

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

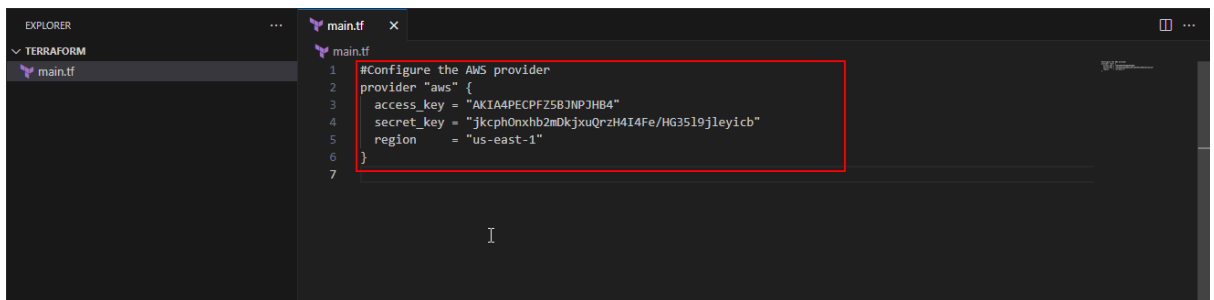
Steps to be followed:

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

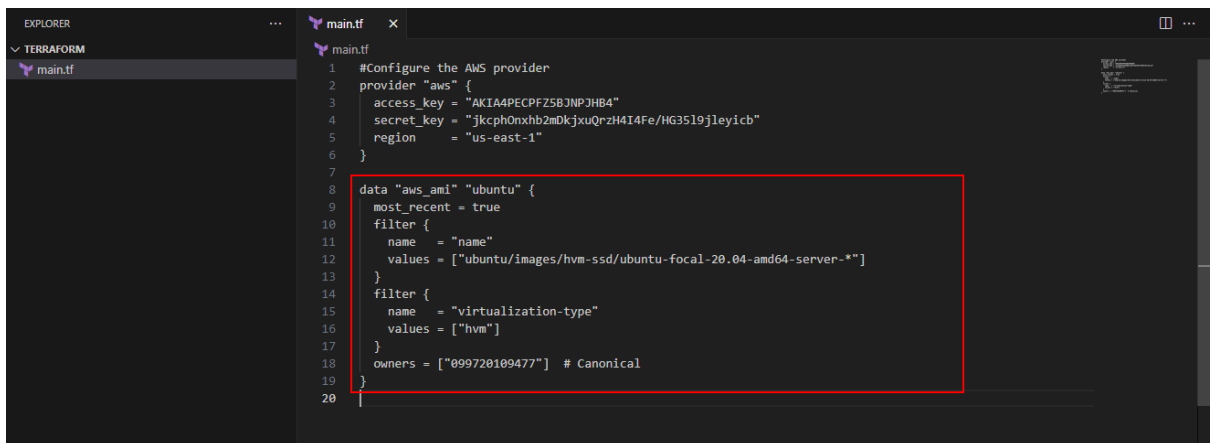
- 1.1 Open your Terraform configuration environment and create a file named **main.tf**.
Add the following configuration block as shown in the screenshot below:

```
#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
}
```



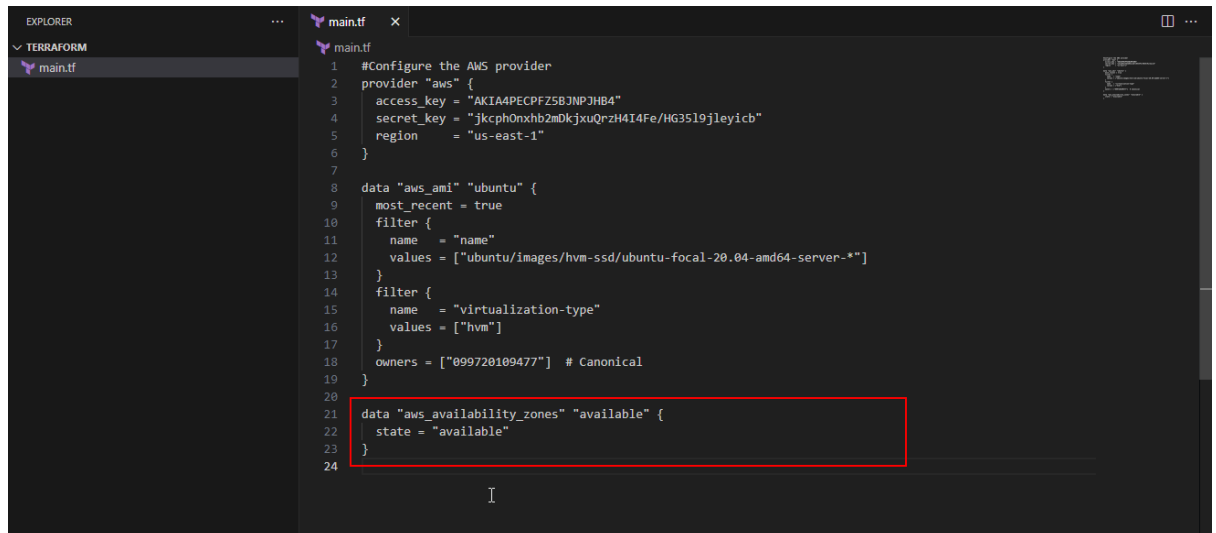
- 1.2 Add the AWS AMI data source as shown in the screenshot below:

```
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
}
```



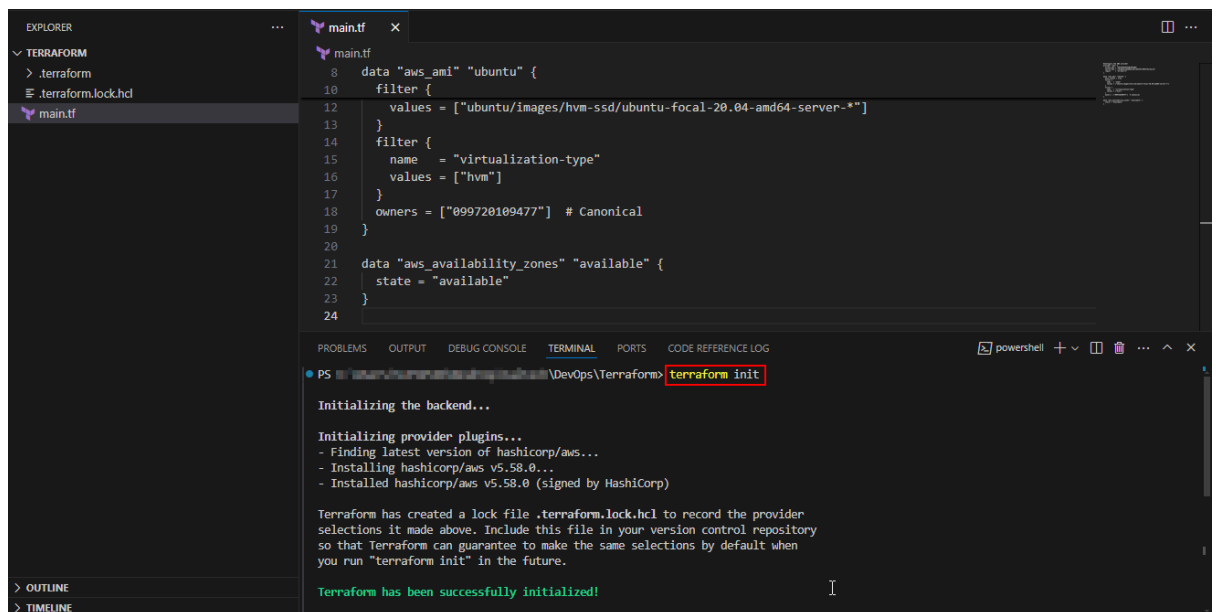
1.3 Declare the AWS availability zones data source as shown in the screenshot below:

```
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



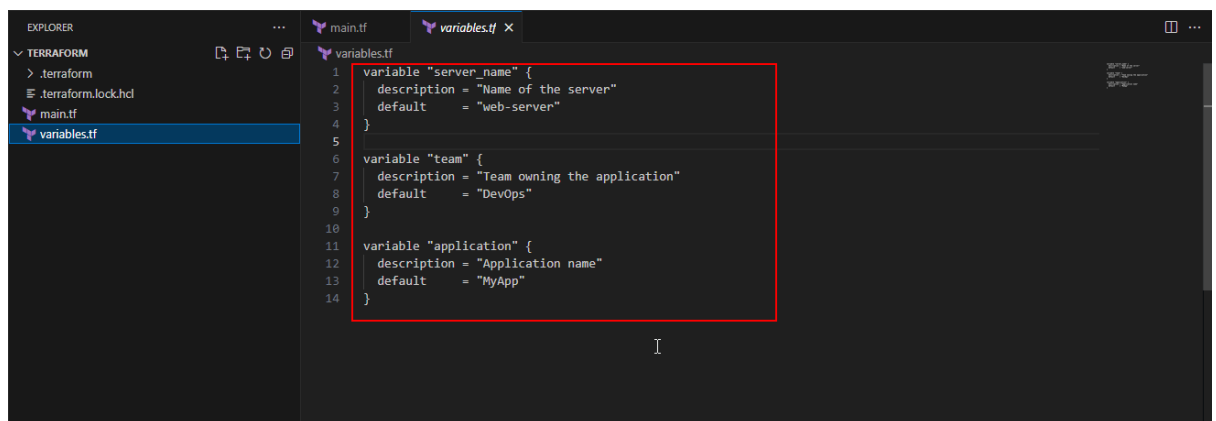
Step 2: Define variables, locals, and outputs

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"  
}
```

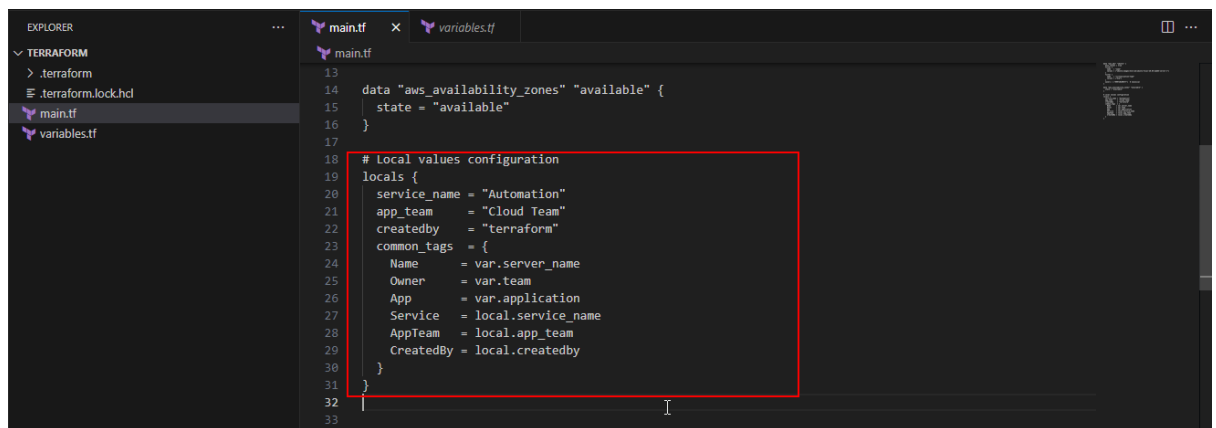
```
variable "team" {  
  description = "Team owning the application"  
  default    = "DevOps"  
}
```

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



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

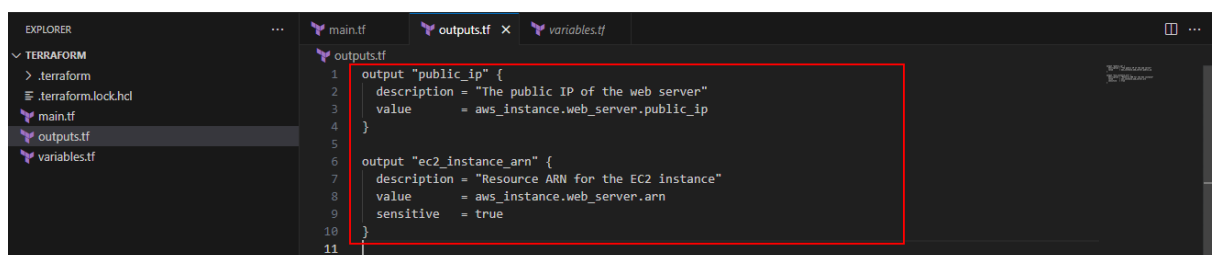
```
# Local values configuration
locals {
  service_name = "Automation"
  app_team     = "Cloud Team"
  createdby    = "terraform"
  common_tags = {
    Name     = var.server_name
    Owner    = var.team
    App      = var.application
    Service  = local.service_name
    AppTeam  = local.app_team
 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

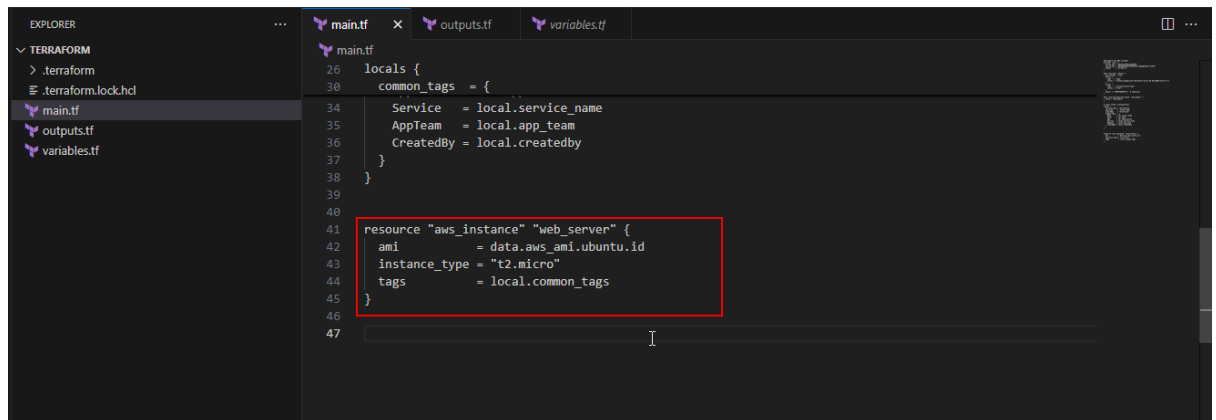
output "ec2_instance_arn" {
  description = "Resource ARN for the EC2 instance"
  value      = aws_instance.web_server.arn
  sensitive  = true
}
```



Step 3: Implement resources with variables and locals

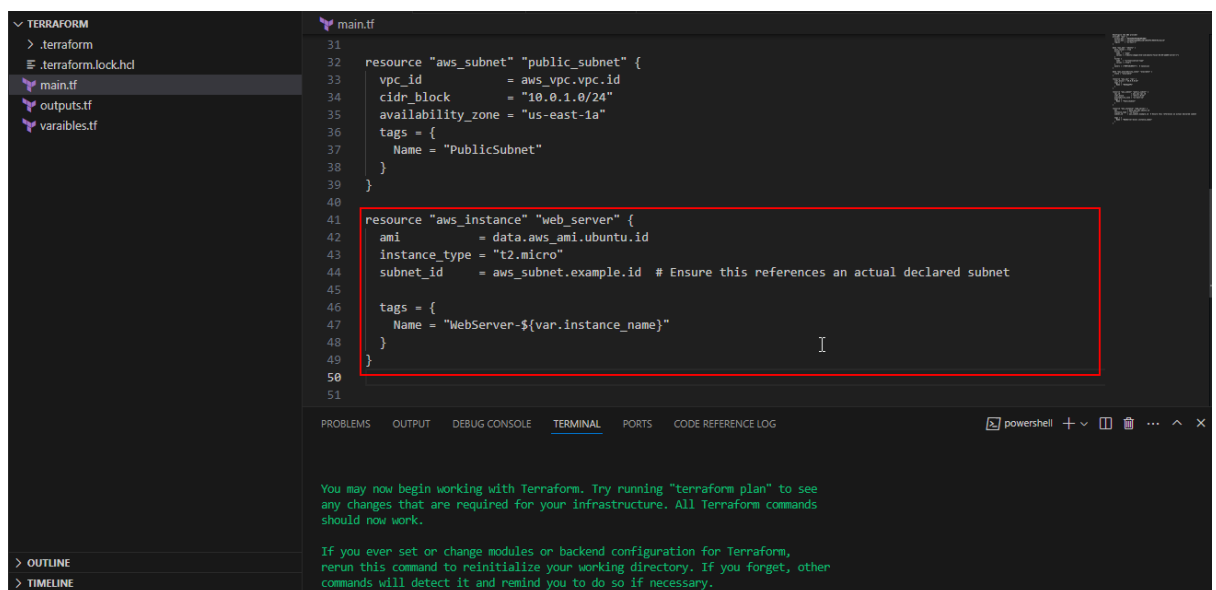
3.1 Add an AWS instance resource in **main.tf** as shown in the screenshot below:

```
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:

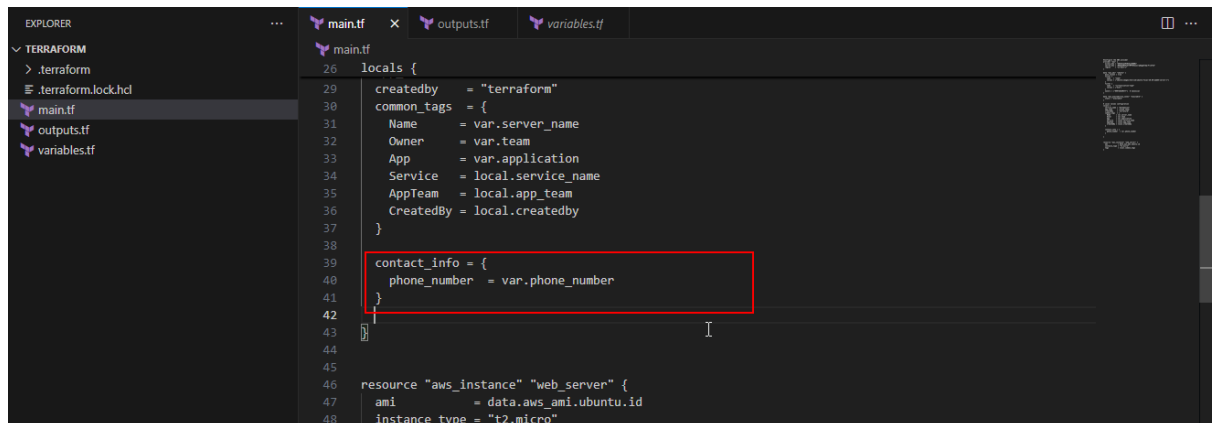
```
resource "aws_instance" "web_server" {  
  ami      = data.aws_ami.ubuntu.id  
  instance_type = "t2.micro"  
  tags     = local.common_tags  
}
```



Step 4: Secure and manage sensitive data

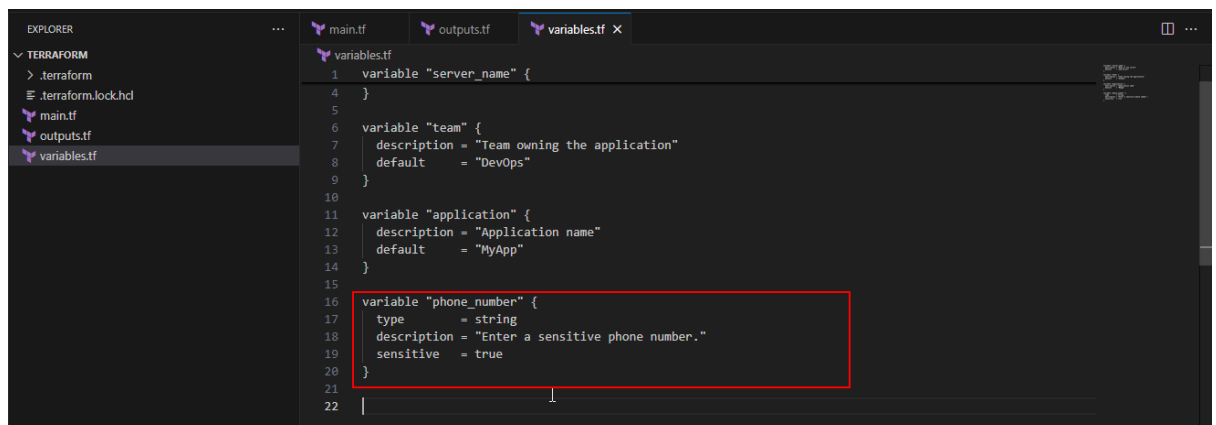
- 4.1 Add sensitive information such as phone number inside the **locals** block in **main.tf** as shown in the screenshot below:

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



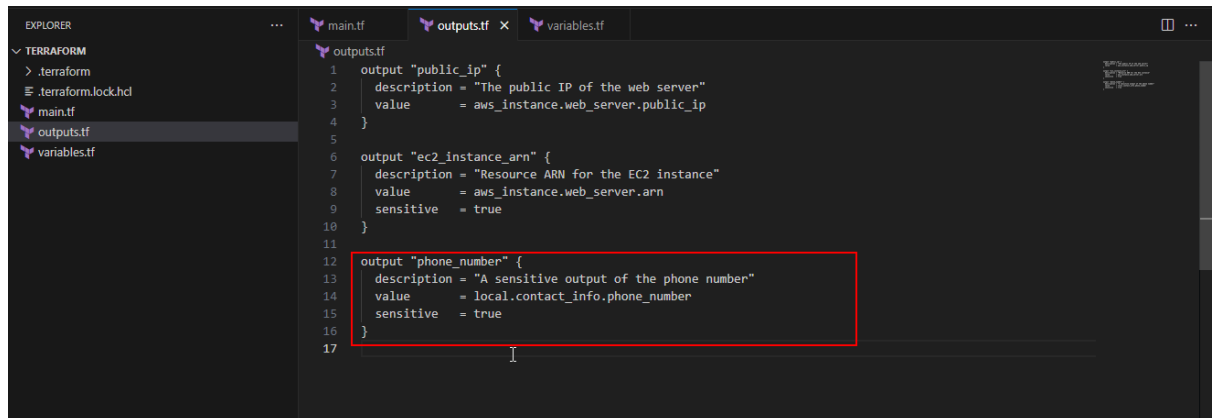
- 4.2 Add the variable for contact information as shown in the screenshot below:

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

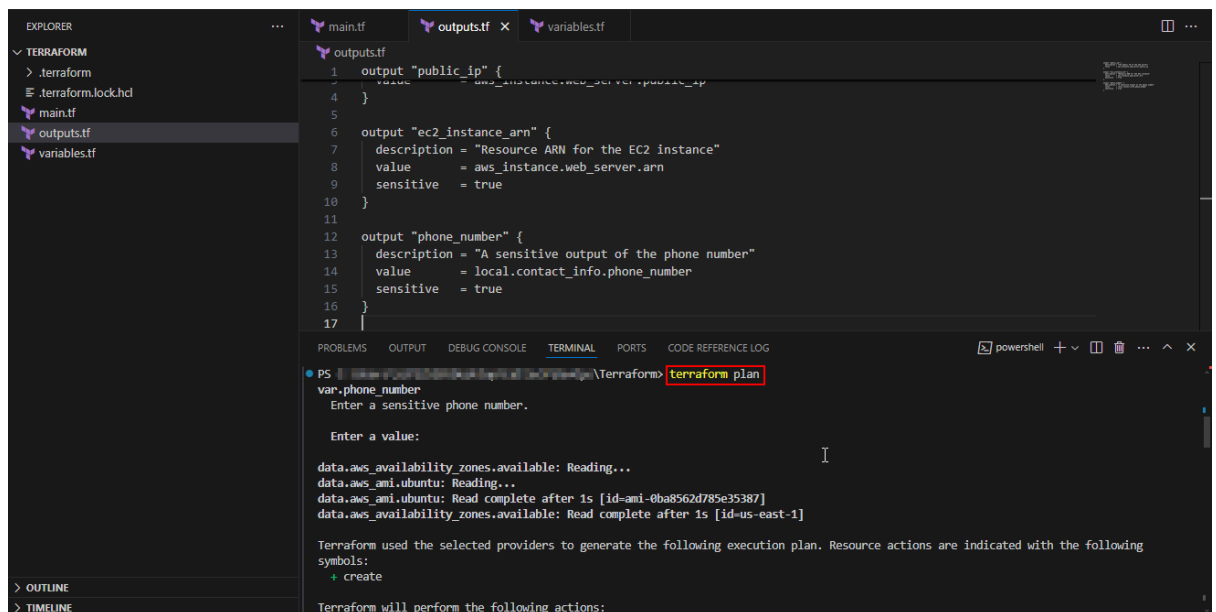


- 4.3 Add the following **output** block with **sensitive = true** in **outputs.tf** to securely output the phone number while keeping it hidden from logs:

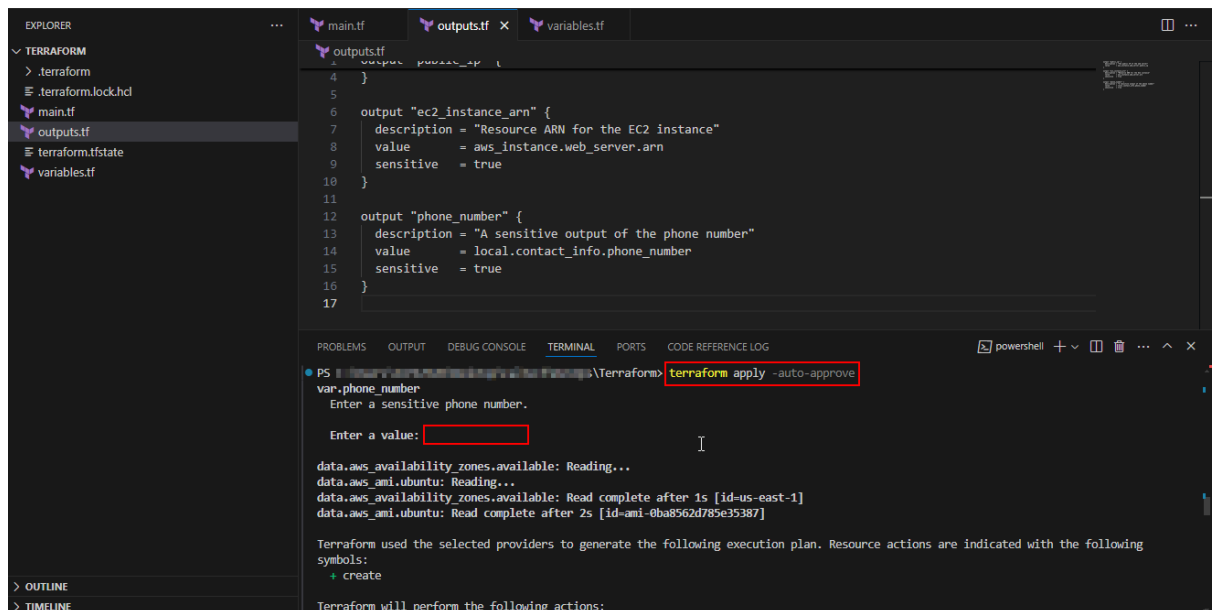
```
output "phone_number" {  
  description = "A sensitive output of the phone number"  
  value      = local.contact_info.phone_number  
  sensitive  = true  
}
```



- 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



The screenshot shows the VS Code interface with the Terraform configuration files in the Explorer on the left. The `outputs.tf` file is open in the editor, showing two output blocks: `ec2_instance_arn` and `phone_number`. The terminal at the bottom shows the command `terraform apply -auto-approve` being executed. It prompts for a sensitive phone number, which is entered as `100.24.31.205`. The terminal output shows the Terraform execution plan and the start of the resource creation process.

```
1 output "ec2_instance_arn" {
2   description = "Resource ARN for the EC2 instance"
3   value       = aws_instance.web_server.arn
4   sensitive   = true
5 }
6
7 output "phone_number" {
8   description = "A sensitive output of the phone number"
9   value       = local.contact_info.phone_number
10  sensitive   = true
11 }
12
13
14
15
16
17
```

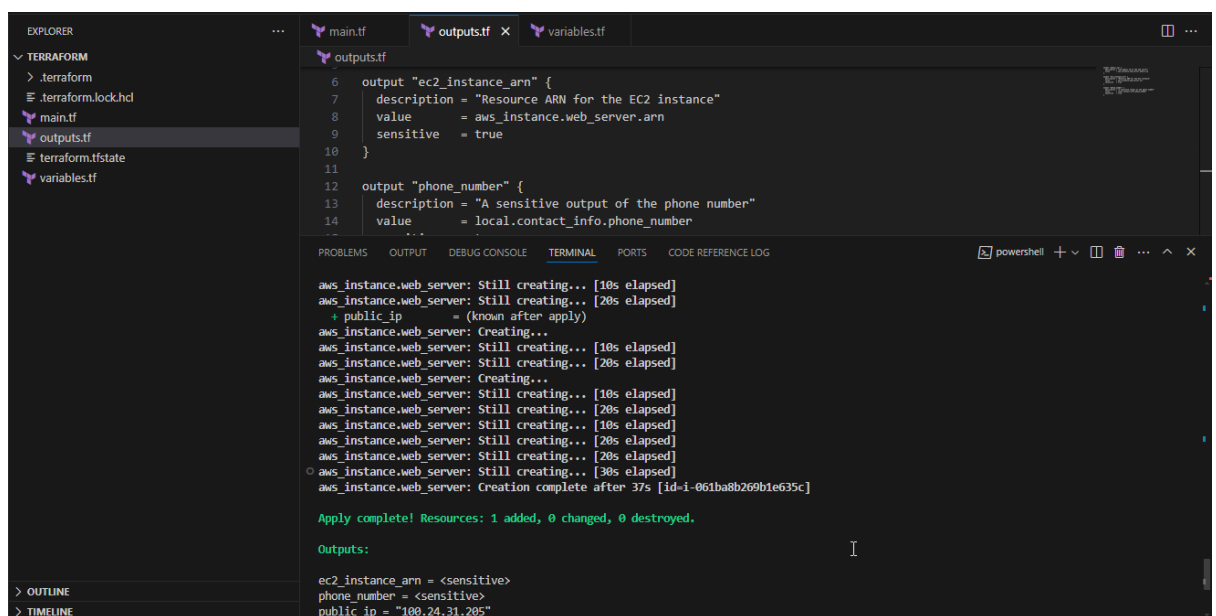
```
PS C:\Terraform> terraform apply -auto-approve
var.phone_number
Enter a sensitive phone number.

Enter a value: 100.24.31.205

data.aws_availability_zones.available: Reading...
data.aws_ami.ubuntu: Reading...
data.aws_availability_zones.available: Read complete after 1s [id=us-east-1]
data.aws_ami.ubuntu: Read complete after 2s [id=ami-0ba8562d785e35387]

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:
```



The screenshot shows the same VS Code interface, but the terminal output has progressed. It shows the `aws_instance.web_server` resource being created. The terminal output indicates that the resource is still creating, with progress updates every 10 seconds. Finally, the resource is created successfully, and the apply process is complete. The terminal output shows the final state of the resources and the public IP address.

```
6 output "ec2_instance_arn" {
7   description = "Resource ARN for the EC2 instance"
8   value       = aws_instance.web_server.arn
9   sensitive   = true
10 }
11
12 output "phone_number" {
13   description = "A sensitive output of the phone number"
14   value       = local.contact_info.phone_number
15 }
16
17
```

```
aws_instance.web_server: Still creating... [10s elapsed]
aws_instance.web_server: Still creating... [20s elapsed]
+ public_ip = (known after apply)
aws_instance.web_server: Creating...
aws_instance.web_server: Still creating... [10s elapsed]
aws_instance.web_server: Still creating... [20s elapsed]
aws_instance.web_server: Creating...
aws_instance.web_server: Still creating... [10s elapsed]
aws_instance.web_server: Still creating... [20s elapsed]
aws_instance.web_server: Still creating... [10s elapsed]
aws_instance.web_server: Still creating... [20s elapsed]
aws_instance.web_server: Still creating... [30s elapsed]
aws_instance.web_server: Creation complete after 37s [id=i-061ba8b269b1e635c]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:
ec2_instance_arn = <sensitive>
phone_number = <sensitive>
public_ip = "100.24.31.205"
```

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  
    }  
  }  
}
```

}

The screenshot shows the VS Code interface with the Explorer on the left and the main editor on the right. The Explorer shows a project structure under 'TERRAFORM' with files: .terraform, .terraform.lock.hcl, main.tf, outputs.tf, terraform.tfstate, and variables.tf. The main editor displays the contents of variables.tf. A red box highlights the definition of the 'ip' variable, which is a map of strings with two entries: 'prod' and 'dev'.

```
16 variable "phone_number" {
17   type        = string
18   description = "Enter a sensitive phone number."
19   sensitive    = true
20 }
21
22 variable "us-east-1-azs" {
23   type        = list(string)
24   default     = ["us-east-1a", "us-east-1b", "us-east-1c", "us-east-1d", "us-east-1e"]
25 }
26
27 variable "ip" {
28   type        = map(string)
29   default     = {
30     prod = "10.0.150.0/24",
31     dev  = "10.0.250.0/24"
32   }
33 }
34
35 variable "env" {
36   type        = map(any)
37   default     = {
38     prod = {
39       ip = "10.0.150.0/24",
40       az = "us-east-1a"
41     },
42     dev = {
43       ip = "10.0.250.0/24",
44       az = "us-east-1e"
45     }
46   }
47 }
```

The screenshot shows the VS Code interface with the Explorer on the left and the main editor on the right. The Explorer shows a project structure under 'TERRAFORM' with files: .terraform, .terraform.lock.hcl, main.tf, outputs.tf, terraform.tfstate, terraform.tfstate.backup, and variables.tf. The main editor displays the contents of variables.tf. A red box highlights the definition of the 'private_subnets' variable, which is a map of objects with two entries: 'subnet1' and 'subnet2'.

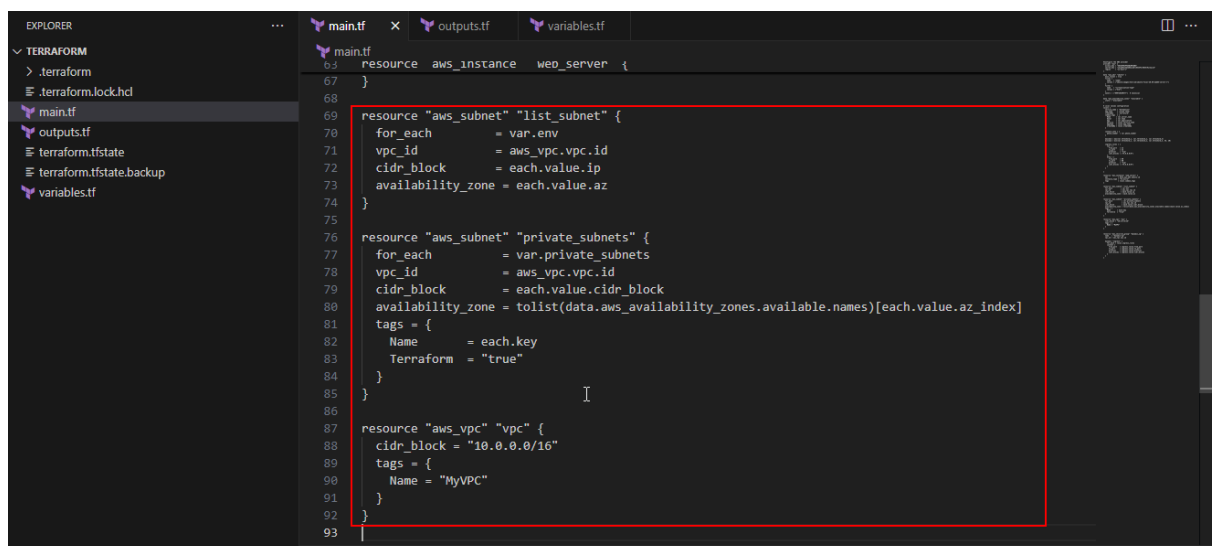
```
35 variable "env" {
36   type        = map(any)
37   default     = {
38     prod = {
39       ip = "10.0.150.0/24",
40       az = "us-east-1a"
41     },
42     dev = {
43       ip = "10.0.250.0/24",
44       az = "us-east-1e"
45     }
46   }
47 }
48
49 variable "private_subnets" {
50   description = "Map of private subnets keyed by name with AZ index as value"
51   type        = map(object({
52     cidr_block = string
53     az_index   = number
54   }))
55   default     = {
56     "subnet1" = {
57       cidr_block = "10.0.1.0/24",
58       az_index   = 0
59     },
60     "subnet2" = {
61       cidr_block = "10.0.2.0/24",
62       az_index   = 1
63     }
64   }
65 }
66 }
```

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

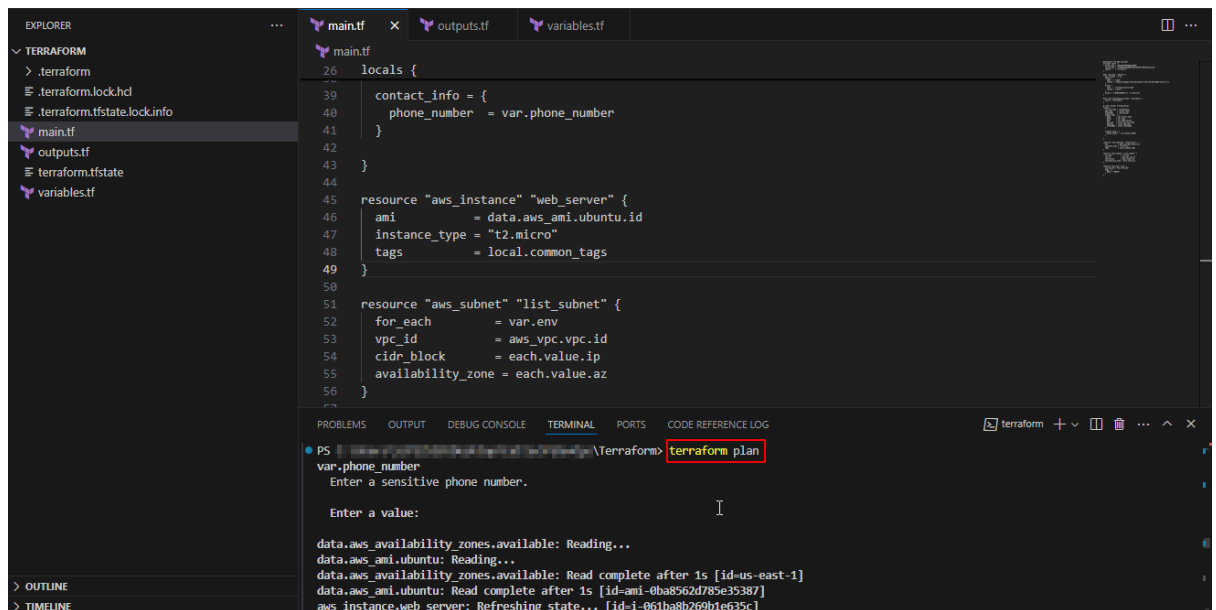
```
resource "aws_subnet" "list_subnet" {
  for_each      = var.env
  vpc_id        = aws_vpc.vpc.id
  cidr_block    = each.value.ip
  availability_zone = each.value.az
}

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



The screenshot shows the VS Code interface with the Terraform Explorer on the left. The main editor displays the `main.tf` file, which defines a web server instance and a subnet. The terminal at the bottom shows the execution of the `terraform plan` command, which prompts for a phone number and displays the planned changes for the AWS resources.

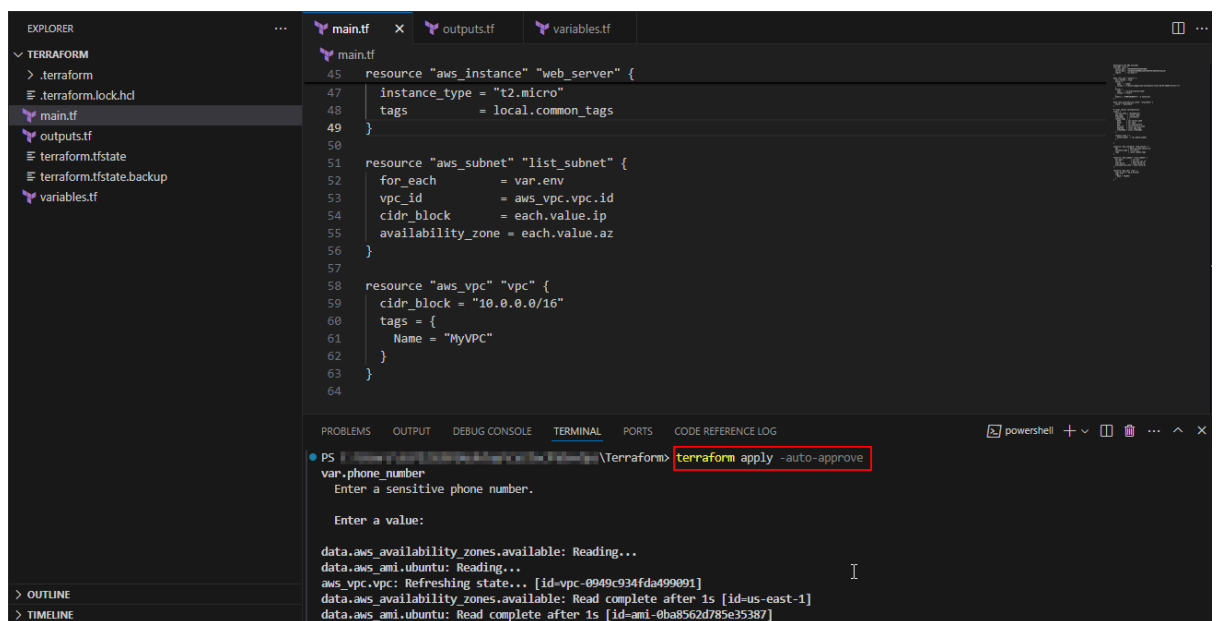
```
main.tf
26 locals {
27   contact_info = {
28     phone_number = var.phone_number
29   }
30 }
31
32 resource "aws_instance" "web_server" {
33   ami           = data.aws_ami.ubuntu.id
34   instance_type = "t2.micro"
35   tags          = local.common_tags
36 }
37
38 resource "aws_subnet" "list_subnet" {
39   for_each = var.env
40   vpc_id    = aws_vpc.vpc.id
41   cidr_block = each.value.ip
42   availability_zone = each.value.az
43 }
```

```
PS C:\Users\user\Documents\Terraform> terraform plan
var.phone_number
Enter a sensitive phone number.

Enter a value:

data.aws_availability_zones.available: Reading...
data.aws_ami.ubuntu: Reading...
data.aws_availability_zones.available: Read complete after 1s [id=us-east-1]
data.aws_ami.ubuntu: Read complete after 1s [id=ami-0ba8562d785e35387]
aws_instance.web_server: Refreshing state... [id=i-061ba8b269b1e635c]
```

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



The screenshot shows the VS Code interface with the Terraform Explorer on the left. The main editor displays the `main.tf` file, which defines a web server instance, a subnet, and a VPC. The terminal at the bottom shows the execution of the `terraform apply -auto-approve` command, which prompts for a phone number and displays the progress of applying the configuration.

```
main.tf
45 resource "aws_instance" "web_server" {
46   instance_type = "t2.micro"
47   tags          = local.common_tags
48 }
49
50
51 resource "aws_subnet" "list_subnet" {
52   for_each = var.env
53   vpc_id    = aws_vpc.vpc.id
54   cidr_block = each.value.ip
55   availability_zone = each.value.az
56 }
57
58 resource "aws_vpc" "vpc" {
59   cidr_block = "10.0.0.0/16"
60   tags = {
61     Name = "MyVPC"
62   }
63 }
64
```

```
PS C:\Users\user\Documents\Terraform> terraform apply -auto-approve
var.phone_number
Enter a sensitive phone number.

Enter a value:

data.aws_availability_zones.available: Reading...
data.aws_ami.ubuntu: Reading...
aws_vpc.vpc: Refreshing state... [id=vpc-0949c934fda499091]
data.aws_availability_zones.available: Read complete after 1s [id=us-east-1]
data.aws_ami.ubuntu: Read complete after 1s [id=ami-0ba8562d785e35387]
```

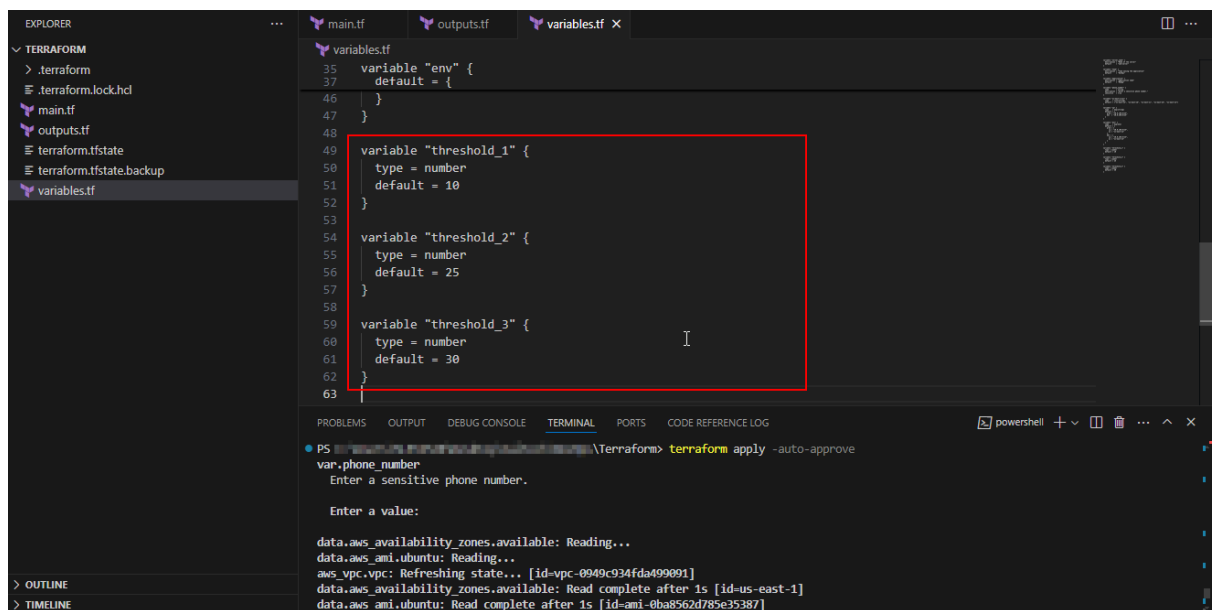
Step 6: Utilize Terraform built-in functions and dynamic blocks

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

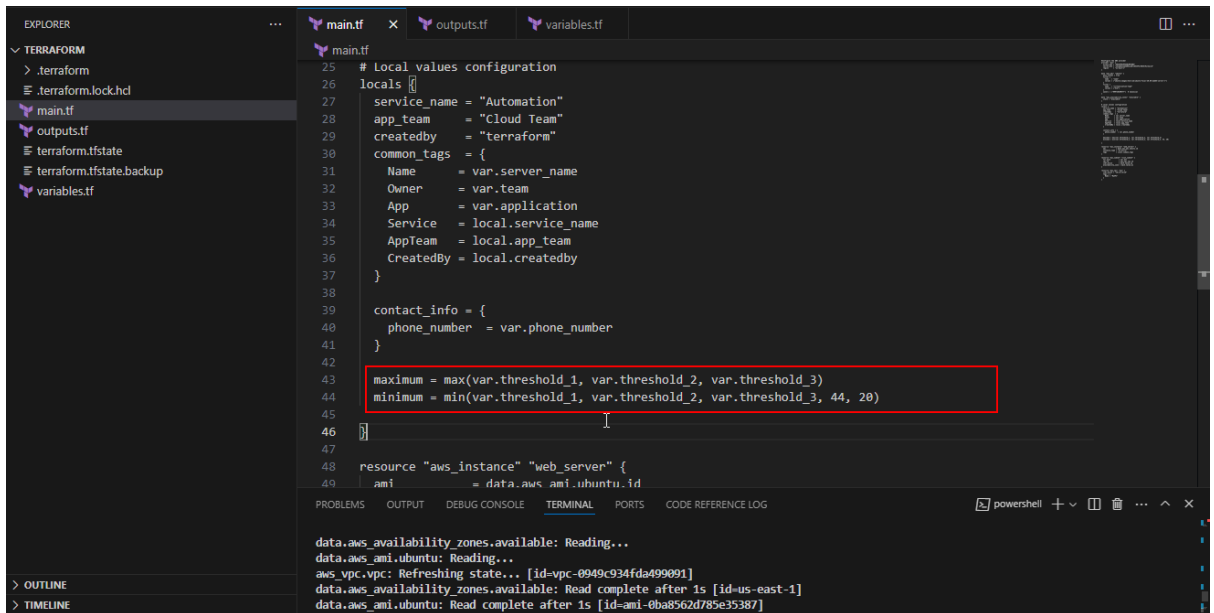
```
variable "threshold_1" {  
  type = number  
  default = 10  
}
```

```
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)
```



The screenshot shows the VS Code editor with the **main.tf** file open. The file contains a `locals` block with the following configuration:

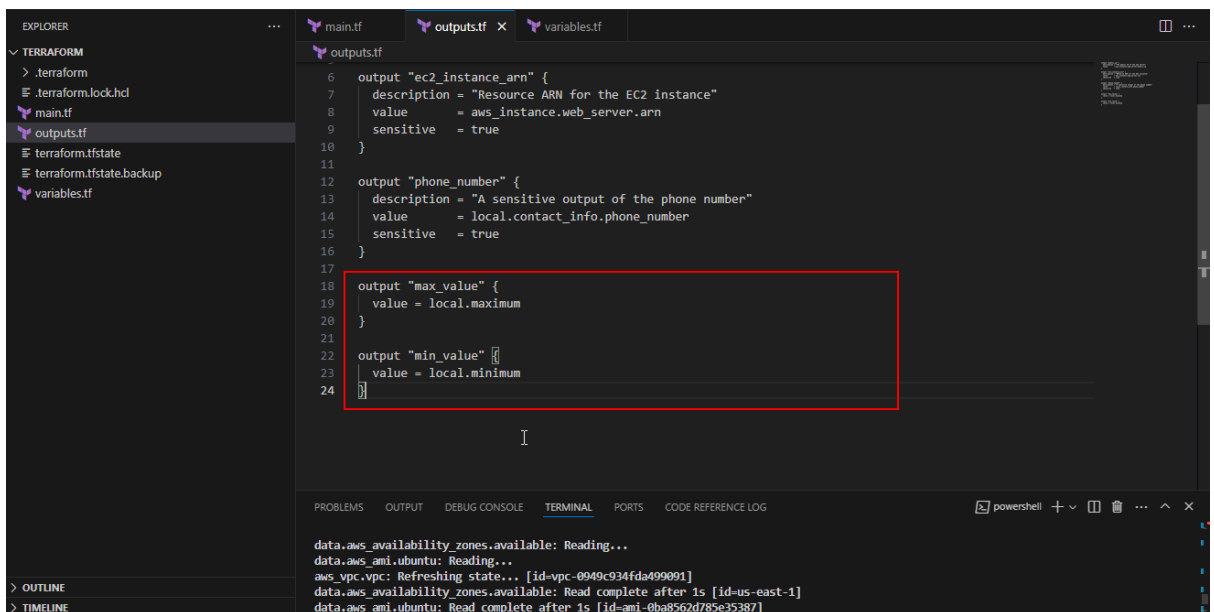
```
25 # Local values configuration
26 locals {
27 service_name = "Automation"
28 app_team = "Cloud Team"
29 createdby = "terraform"
30 common_tags = {
31 Name = var.server_name
32 Owner = var.team
33 App = var.application
34 Service = local.service_name
35 AppTeam = local.app_team
36 CreatedBy = local.createdby
37 }
38
39 contact_info = {
40 phone_number = var.phone_number
41 }
42
43 maximum = max(var.threshold_1, var.threshold_2, var.threshold_3)
44 minimum = min(var.threshold_1, var.threshold_2, var.threshold_3, 44, 20)
45 }
46
47 resource "aws_instance" "web_server" {
48 ami = data.aws_ami.ubuntu.id
49 availability_zones = data.aws_availability_zones.available.names
50 instance_type = var.instance_type
51 tags = common_tags
52 }
```

The lines for `maximum` and `minimum` are highlighted with a red box. The terminal at the bottom shows the status of data sources being read.

- 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
}
```



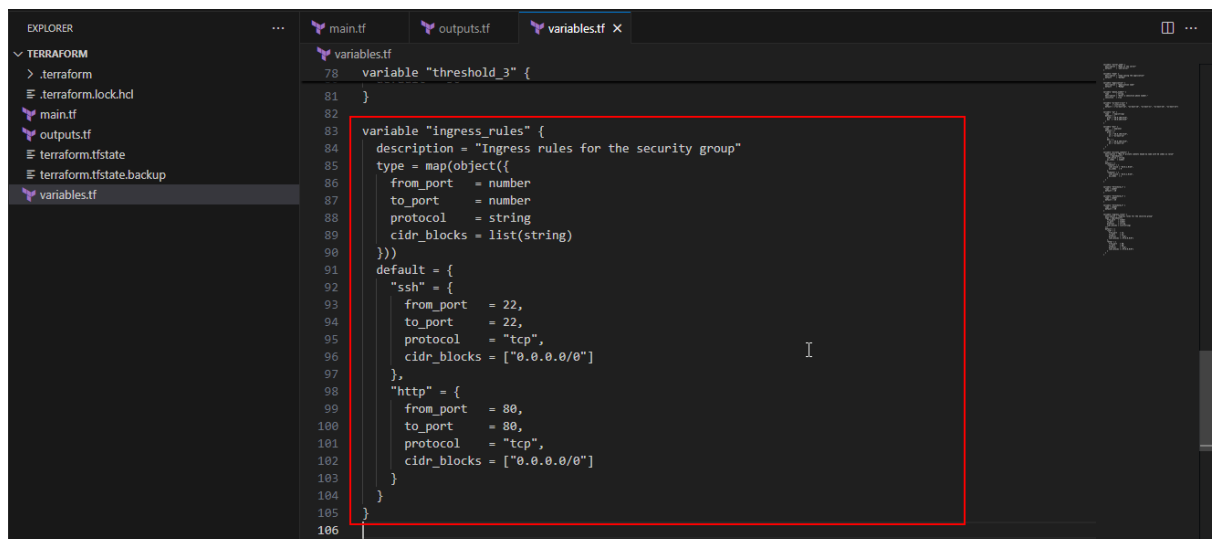
The screenshot shows the VS Code editor with the **output.tf** file open. The file contains the following output blocks:

```
6 output "ec2_instance_arn" {
7 description = "Resource ARN for the EC2 instance"
8 value = aws_instance.web_server.arn
9 sensitive = true
10 }
11
12 output "phone_number" {
13 description = "A sensitive output of the phone number"
14 value = local.contact_info.phone_number
15 sensitive = true
16 }
17
18 output "max_value" {
19 value = local.maximum
20 }
21
22 output "min_value" {
23 value = local.minimum
24 }
```

The `output "max_value"` and `output "min_value"` blocks are highlighted with a red box. The terminal at the bottom shows the status of data sources being read.

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

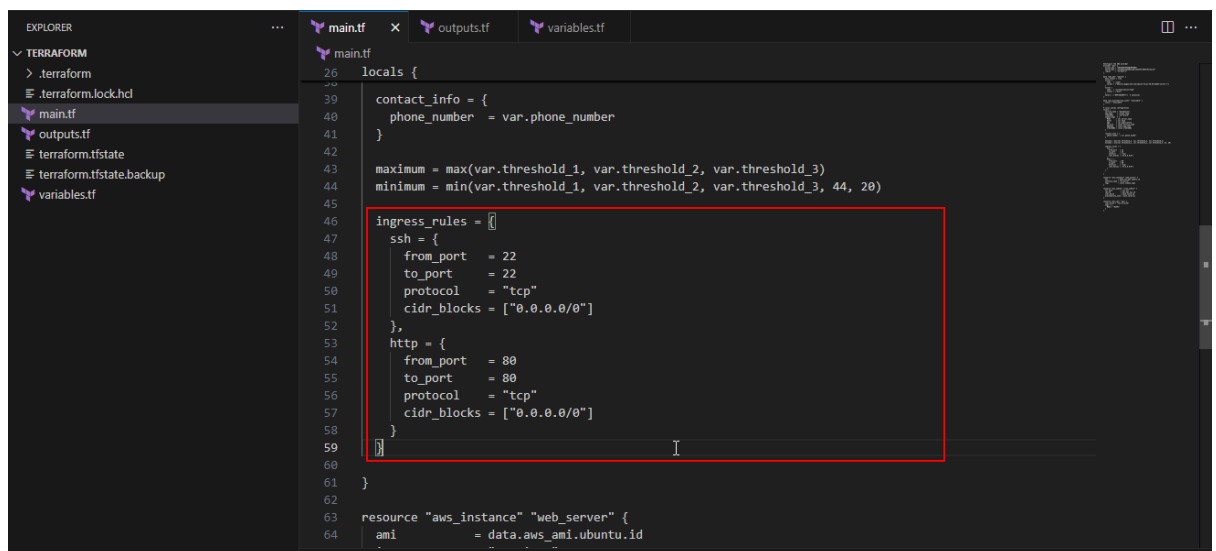
```
variable "ingress_rules" {
 description = "Ingress rules for the security group"
 type = map(object({
 from_port = number
 to_port = number
 protocol = string
 cidr_blocks = list(string)
 }))
 default = {
 "ssh" = {
 from_port = 22,
 to_port = 22,
 protocol = "tcp",
 cidr_blocks = ["0.0.0.0/0"]
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

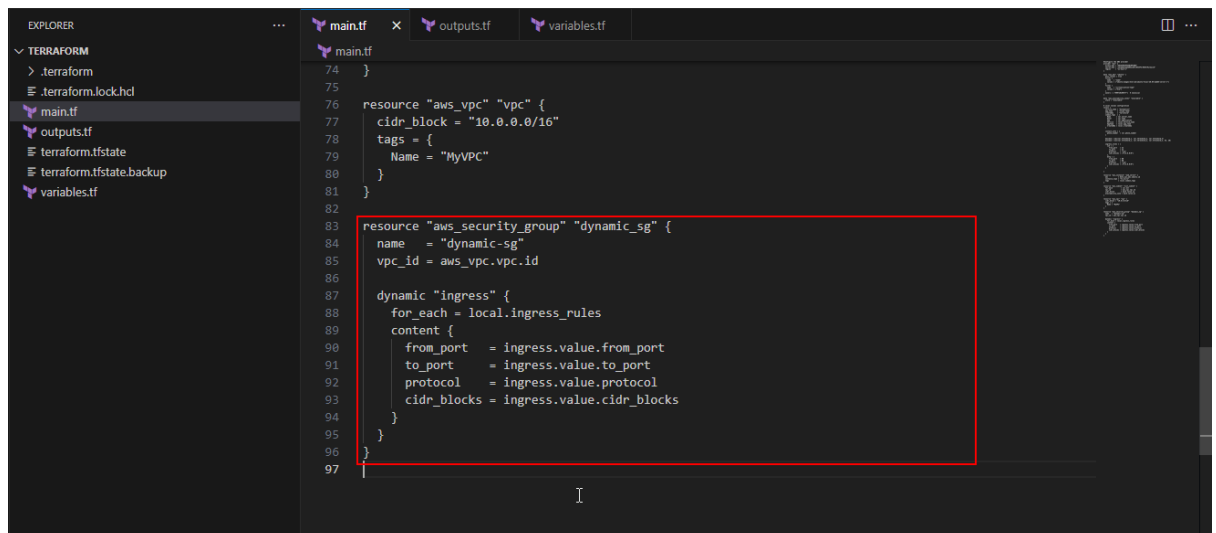
```
ingress_rules = {
 ssh = {
 from_port = 22
 to_port = 22
 protocol = "tcp"
 cidr_blocks = ["0.0.0.0/0"]
 },
 http = {
 from_port = 80
 to_port = 80
 protocol = "tcp"
 cidr_blocks = ["0.0.0.0/0"]
 }
}
```



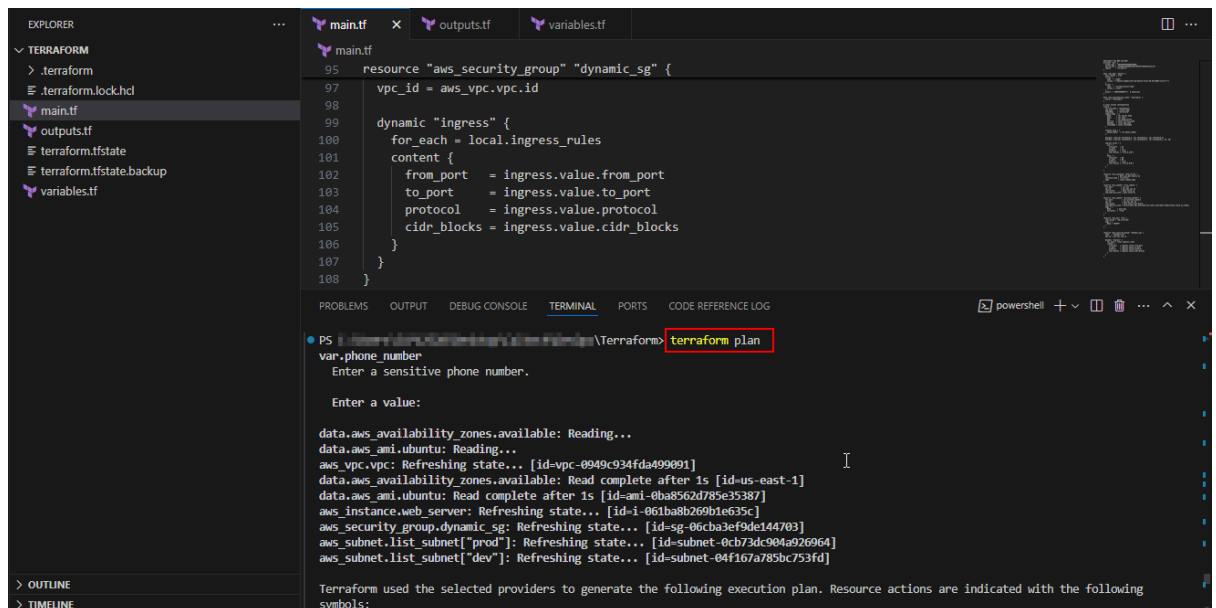
6.6 Add the following resource block in **main.tf** to implement dynamic blocks for security groups as shown in the screenshot below:

```
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**



The screenshot shows the Visual Studio Code interface with the Explorer pane on the left displaying the Terraform project structure. The main editor shows the `main.tf` file with Terraform configuration for an AWS security group. The terminal pane at the bottom shows the execution of the `terraform plan` command in a PowerShell session. The output displays the state of various resources, including availability zones, VPC, subnets, and the security group, indicating that the plan is being generated.

```
main.tf
95 resource "aws_security_group" "dynamic_sg" {
96 vpc_id = aws_vpc.vpc.id
97
98 dynamic "ingress" {
99 for_each = local.ingress_rules
100 content {
101 from_port = ingress.value.from_port
102 to_port = ingress.value.to_port
103 protocol = ingress.value.protocol
104 cidr_blocks = ingress.value.cidr_blocks
105 }
106 }
107 }
108 }
```

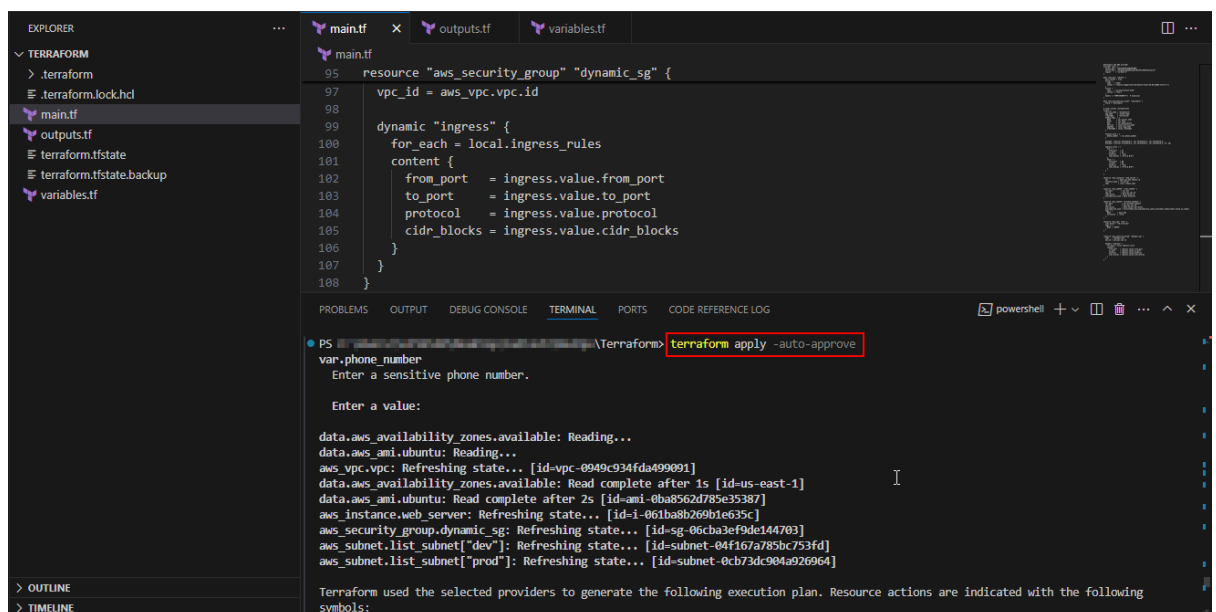
```
PS > terraform plan
var.phone_number
Enter a sensitive phone number.

Enter a value:

data.aws_availability_zones.available: Reading...
data.aws_ami.ubuntu: Reading...
aws_vpc.vpc: Refreshing state... [id=vpc-0949c934fda499091]
data.aws_availability_zones.available: Read complete after 1s [id=us-east-1]
data.aws_ami.ubuntu: Read complete after 1s [id=ami-0ba8562d785e35387]
aws_instance.web_server: Refreshing state... [id=i-061ba8b269b1e635c]
aws_security_group.dynamic_sg: Refreshing state... [id=sg-06cba3ef9de144703]
aws_subnet.list_subnet["prod"]: Refreshing state... [id=subnet-0cb73dc904a926964]
aws_subnet.list_subnet["dev"]: Refreshing state... [id=subnet-04f167a785bc753fd]

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

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



The screenshot shows the Visual Studio Code interface with the same Terraform project structure. The main editor shows the `main.tf` file. The terminal pane at the bottom shows the execution of the `terraform apply -auto-approve` command in a PowerShell session. The output displays the state of various resources, including availability zones, VPC, subnets, and the security group, indicating that the configuration is being applied.

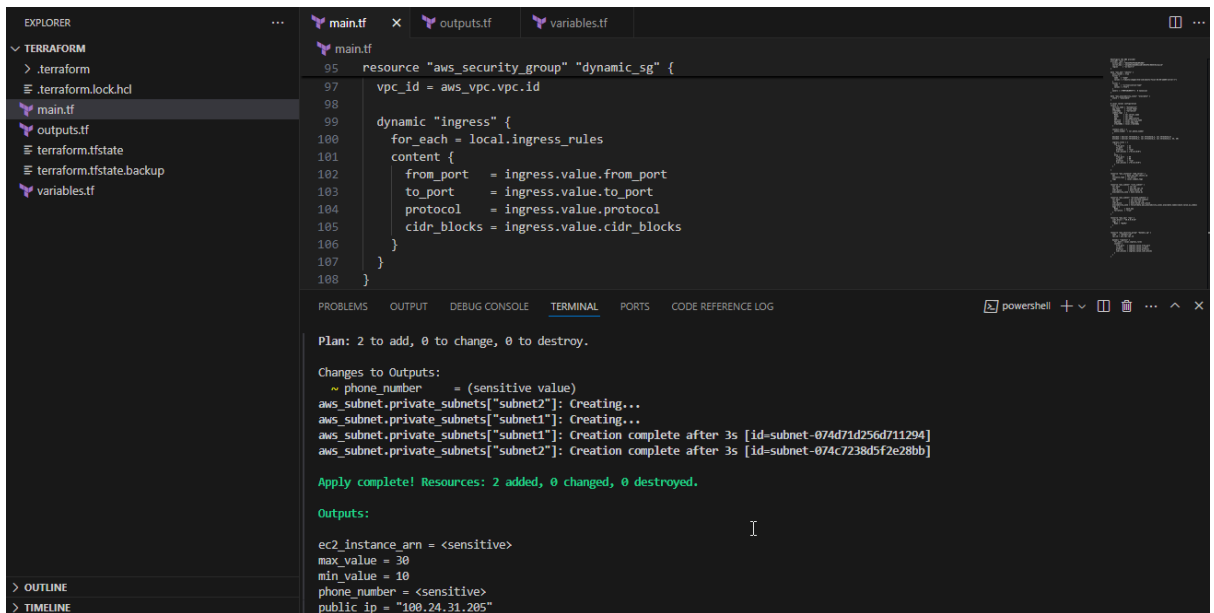
```
main.tf
95 resource "aws_security_group" "dynamic_sg" {
96 vpc_id = aws_vpc.vpc.id
97
98 dynamic "ingress" {
99 for_each = local.ingress_rules
100 content {
101 from_port = ingress.value.from_port
102 to_port = ingress.value.to_port
103 protocol = ingress.value.protocol
104 cidr_blocks = ingress.value.cidr_blocks
105 }
106 }
107 }
108 }
```

```
PS > terraform apply -auto-approve
var.phone_number
Enter a sensitive phone number.

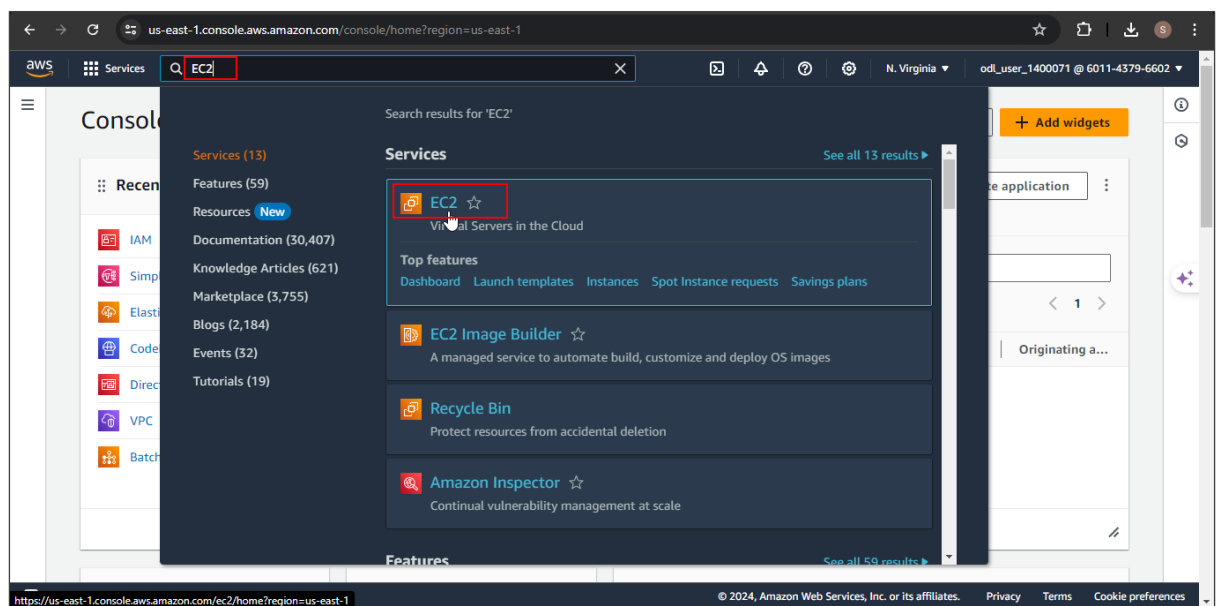
Enter a value:

data.aws_availability_zones.available: Reading...
data.aws_ami.ubuntu: Reading...
aws_vpc.vpc: Refreshing state... [id=vpc-0949c934fda499091]
data.aws_availability_zones.available: Read complete after 1s [id=us-east-1]
data.aws_ami.ubuntu: Read complete after 2s [id=ami-0ba8562d785e35387]
aws_instance.web_server: Refreshing state... [id=i-061ba8b269b1e635c]
aws_security_group.dynamic_sg: Refreshing state... [id=sg-06cba3ef9de144703]
aws_subnet.list_subnet["dev"]: Refreshing state... [id=subnet-04f167a785bc753fd]
aws_subnet.list_subnet["prod"]: Refreshing state... [id=subnet-0cb73dc904a926964]

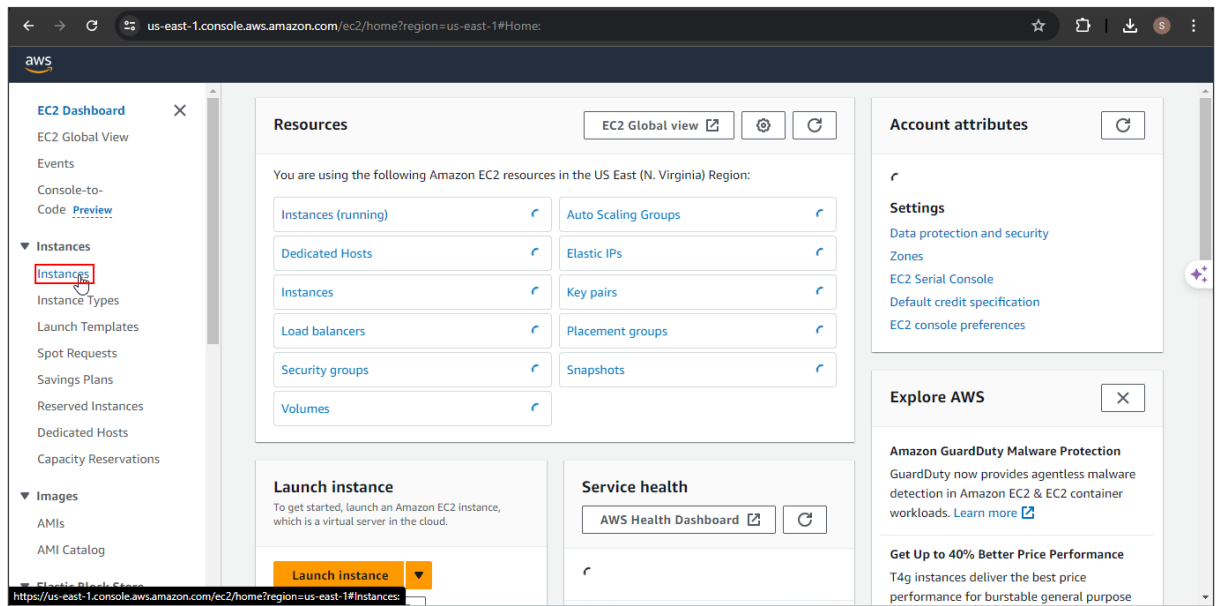
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
```



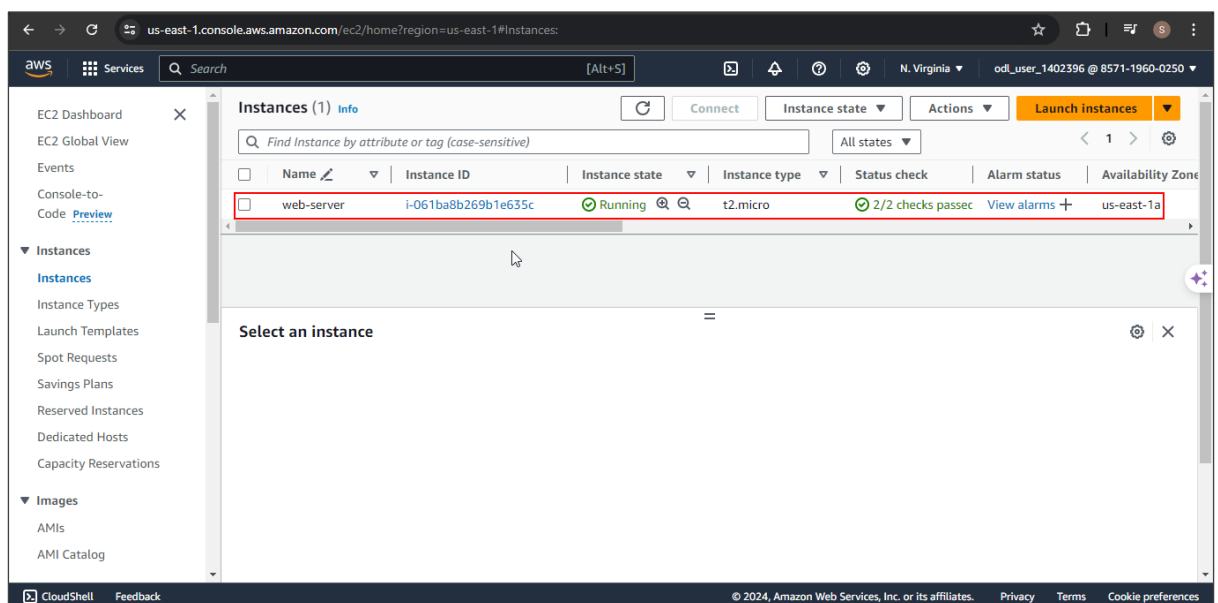
6.9 Navigate to the AWS console home, and search for and click on EC2 as shown in the screenshot below:



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



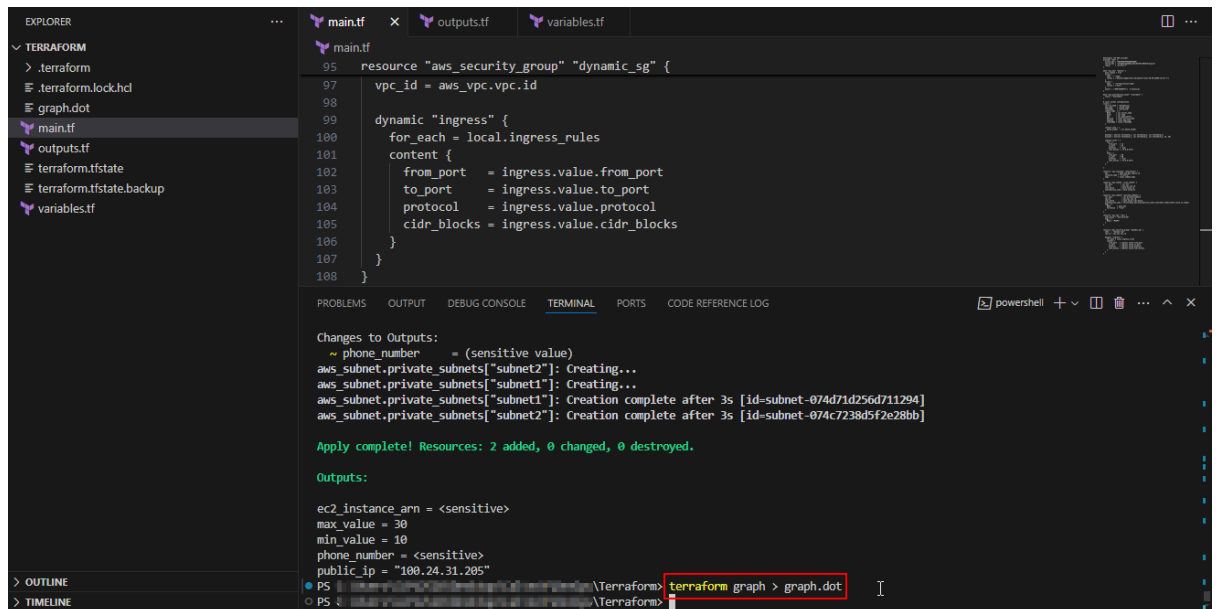
The EC2 instance has been created successfully as shown in the screenshot below:



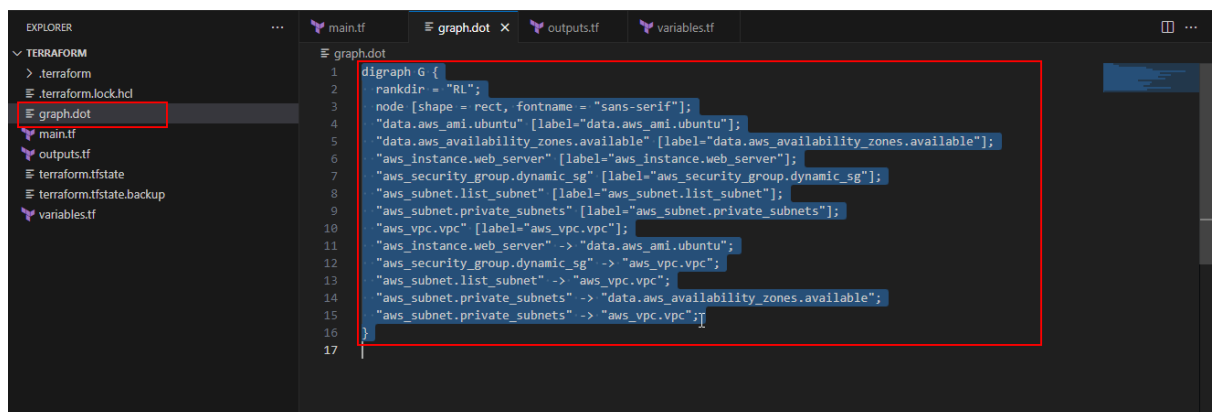
## Step 7: Generate and visualize the resource graph

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

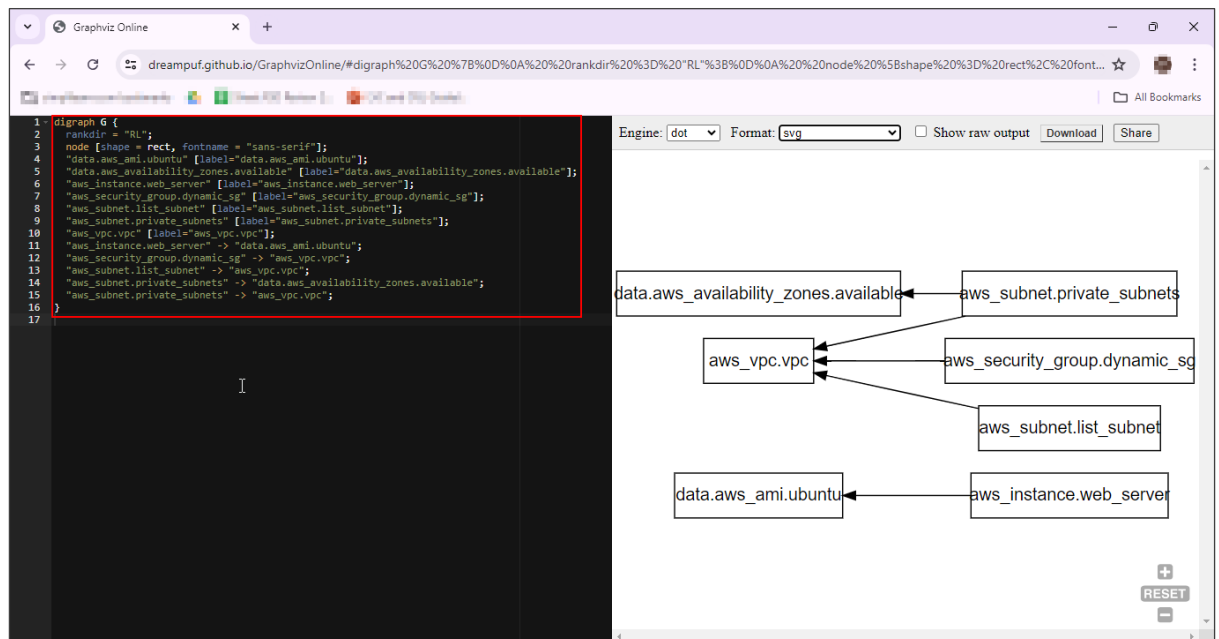
**terraform graph > graph.dot**



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.