Lesson 11 Demo 06

Working with Dynamic Blocks

Objective: To implement dynamic blocks and local variables in Terraform for efficient and flexible infrastructure configuration management

Tools required: Terraform, AWS, and Visual Studio Code

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

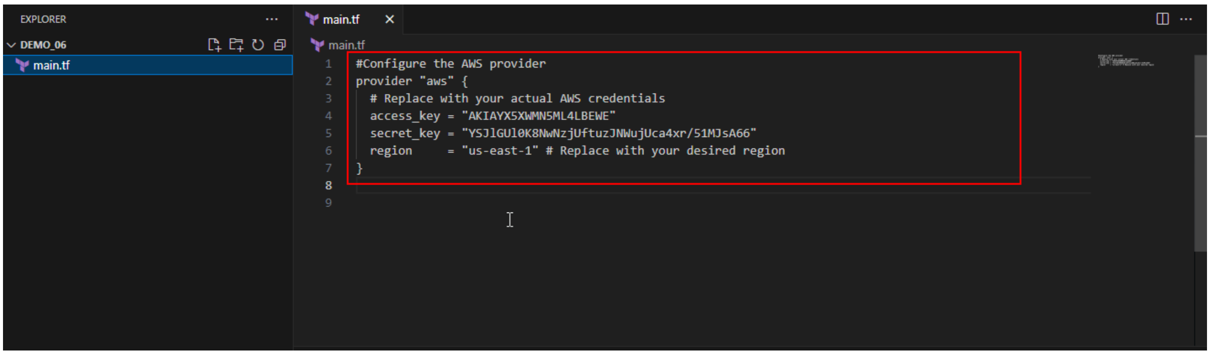
Steps to be followed:

1. Set up basic AWS infrastructure
2. Implement dynamic blocks for security groups
3. Utilize local variables for resource configuration
4. Apply configuration changes

Step 1: Set up basic AWS infrastructure

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

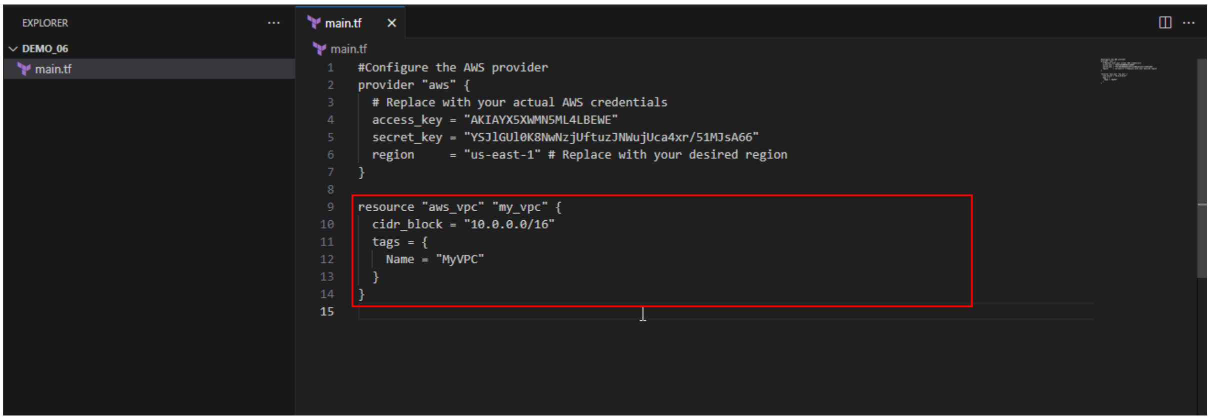
```
#Configure the AWS provider
provider "aws" {
  # Replace with your actual AWS credentials
  access_key = "YOUR_ACCESS_KEY"
  secret_key = "YOUR_SECRET_KEY"
  region     = "us-east-1" # Replace with your desired region
}
```



```
#Configure the AWS provider
provider "aws" {
  # Replace with your actual AWS credentials
  access_key = "AKIAYX5XWMN5ML4LBEWE"
  secret_key = "YSJlGUl0K8NwNzjUftuzJNWujUca4xr/51MJsA66"
  region     = "us-east-1" # Replace with your desired region
}
```

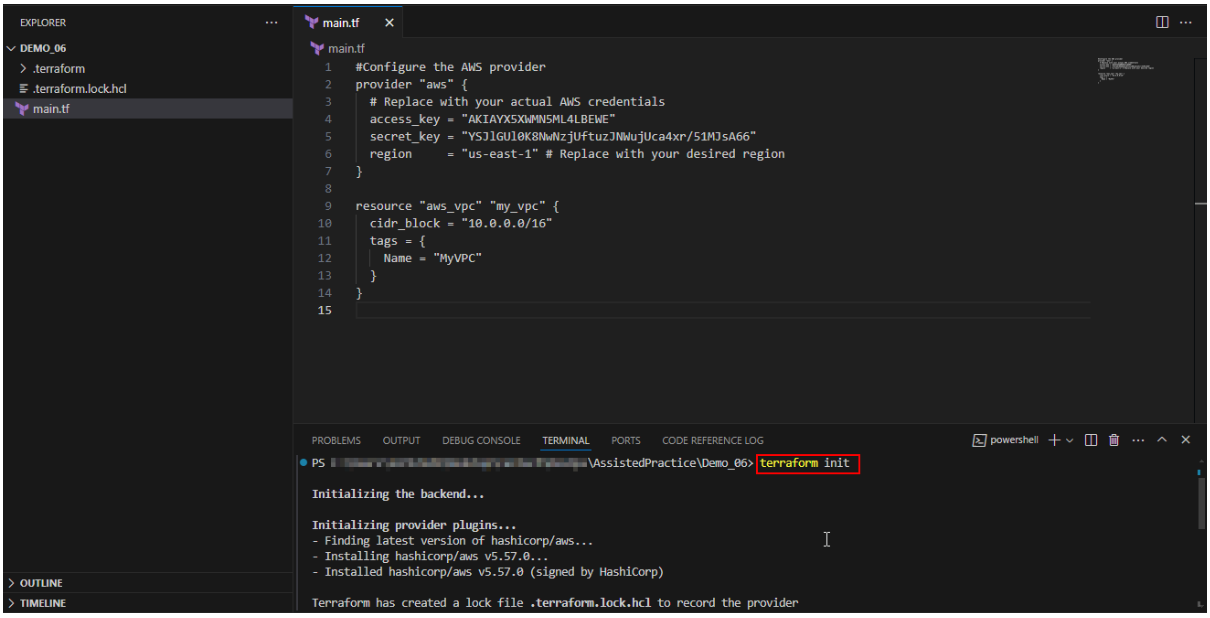
1.2 Define a new VPC to host the network components as shown in the screenshot below:

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



```
#Configure the AWS provider
provider "aws" {
  # Replace with your actual AWS credentials
  access_key = "AKIAYX5XWMN5ML4LBEWE"
  secret_key = "YSJlGUl0K8NwNzjUftuzJNWujUca4xr/51MJsA66"
  region     = "us-east-1" # Replace with your desired region
}

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

1.3 Run **terraform init** to initialize the directory and download the necessary provider plugins as shown in the screenshot below:



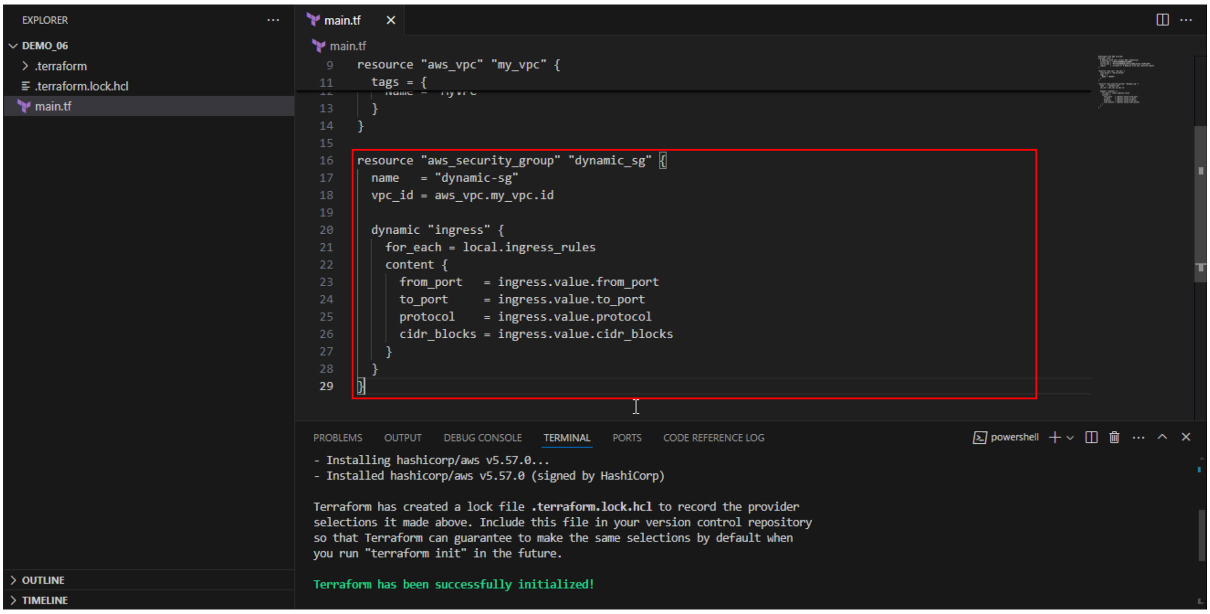
```
#Configure the AWS provider
provider "aws" {
  # Replace with your actual AWS credentials
  access_key = "AKIAYX5XWMN5ML4LBEWE"
  secret_key = "YSJlGUl0K8NwNzjUftuzJNWujUca4xr/51MJsA66"
  region     = "us-east-1" # Replace with your desired region
}

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

```
PS [redacted]\AssistedPractice\Demo_06> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.57.0...
- Installed hashicorp/aws v5.57.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
```

Step 2: Implement dynamic blocks for security groups

2.1 Create a security group within the VPC and use dynamic blocks to handle multiple ingress rules as shown in the screenshot below:

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

  dynamic "ingress" {
    for_each = local.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
    }
  }
}
```

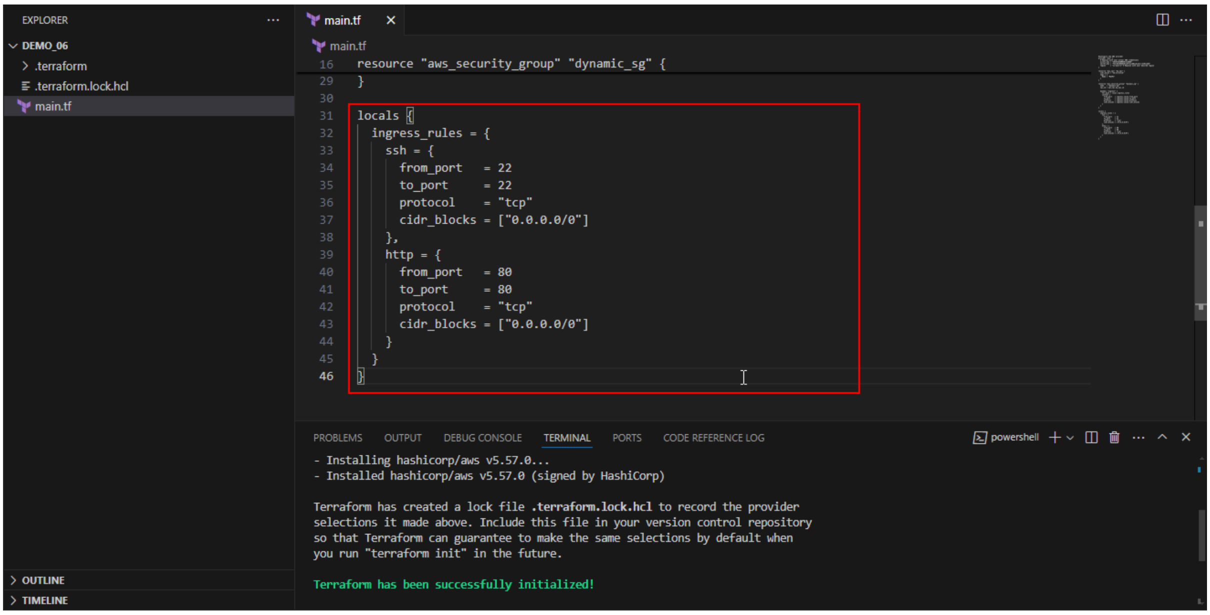


```
resource "aws_vpc" "my_vpc" {
  tags = {
  }
}

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

  dynamic "ingress" {
    for_each = local.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
    }
  }
}
```

```
- Installing hashicorp/aws v5.57.0...
- Installed hashicorp/aws v5.57.0 (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!
```

2.2 Define **locals** in **main.tf** that include ingress rule configurations:

```
locals {
  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"]
    }
  }
}
```



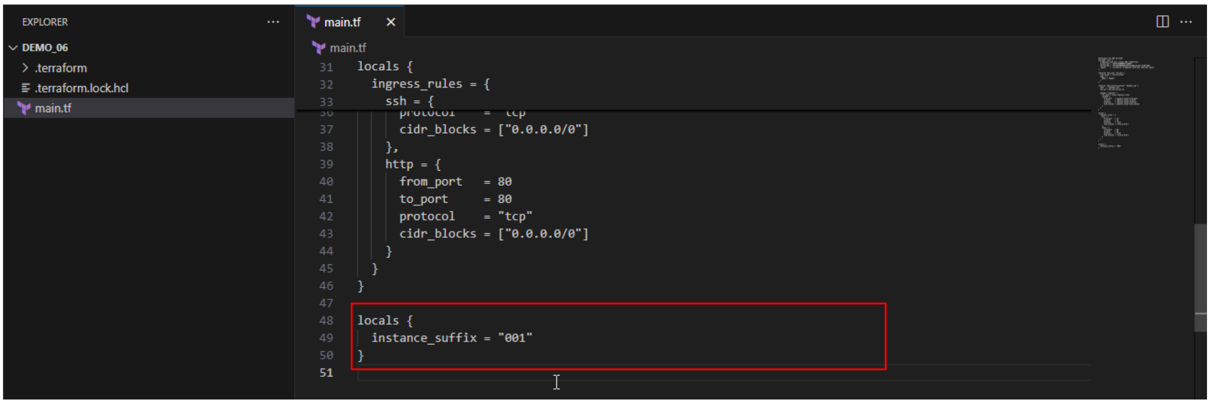
```
resource "aws_security_group" "dynamic_sg" {
}

locals {
  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"]
    }
  }
}
```

```
- Installing hashicorp/aws v5.57.0...
- Installed hashicorp/aws v5.57.0 (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!
```

Step 3: Utilize local variables for resource configuration

3.1 Add a local variable to define a suffix or any configuration detail dynamically as shown in the screenshot below:

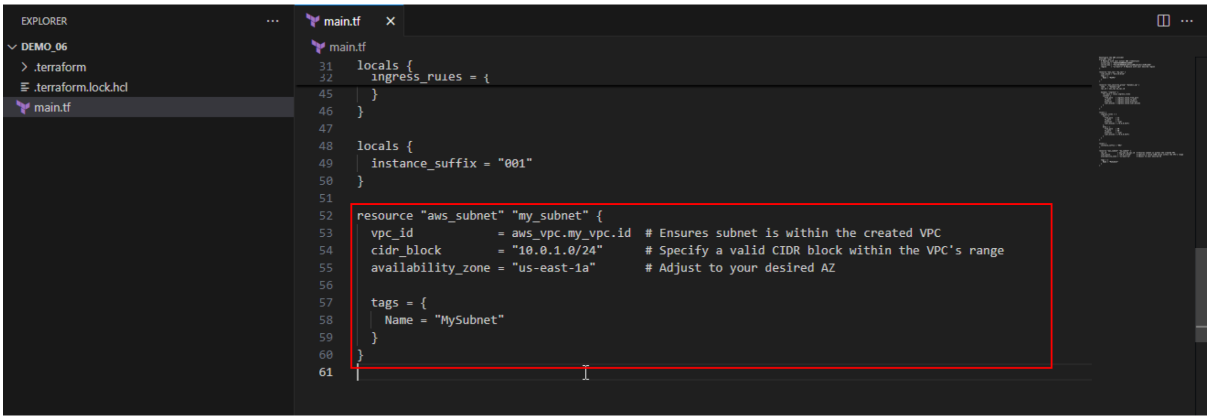
```
locals {
  instance_suffix = "001"
}
```



```
locals {
  ingress_rules = {
    ssh = {
      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"]
    }
  }
}

locals {
  instance_suffix = "001"
}
```

3.2 Declare the subnet in **main.tf** as shown in the screenshot below:

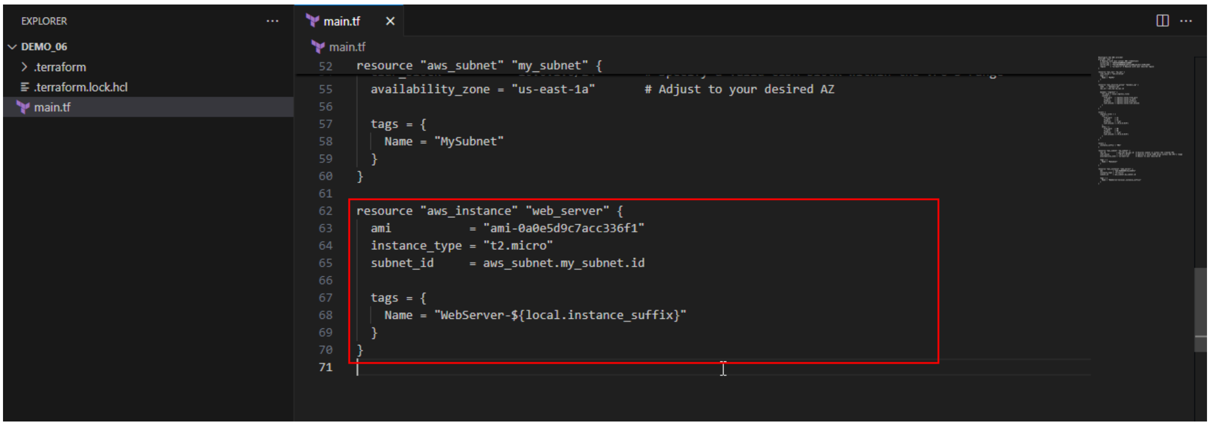
```
resource "aws_subnet" "my_subnet" {
  vpc_id            = aws_vpc.my_vpc.id  # Ensures subnet is within the created VPC
  cidr_block        = "10.0.1.0/24"      # Specify a valid CIDR block within the VPC's range
  availability_zone = "us-east-1a"       # Adjust to your desired AZ

  tags = {
    Name = "MySubnet"
  }
}
```



```
locals {
  ingress_rules = {
  }
}

locals {
  instance_suffix = "001"
}

resource "aws_subnet" "my_subnet" {
  vpc_id            = aws_vpc.my_vpc.id  # Ensures subnet is within the created VPC
  cidr_block        = "10.0.1.0/24"      # Specify a valid CIDR block within the VPC's range
  availability_zone = "us-east-1a"       # Adjust to your desired AZ

  tags = {
    Name = "MySubnet"
  }
}
```

3.3 Define an AWS EC2 instance within the declared subnet specifying the AMI, instance type, and referencing the subnet ID as shown in the screenshot below:

```
resource "aws_instance" "web_server" {
  ami           = "ami-0a0e5d9c7acc336f1"
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.my_subnet.id

  tags = {
    Name = "WebServer-${local.instance_suffix}"
  }
}
```

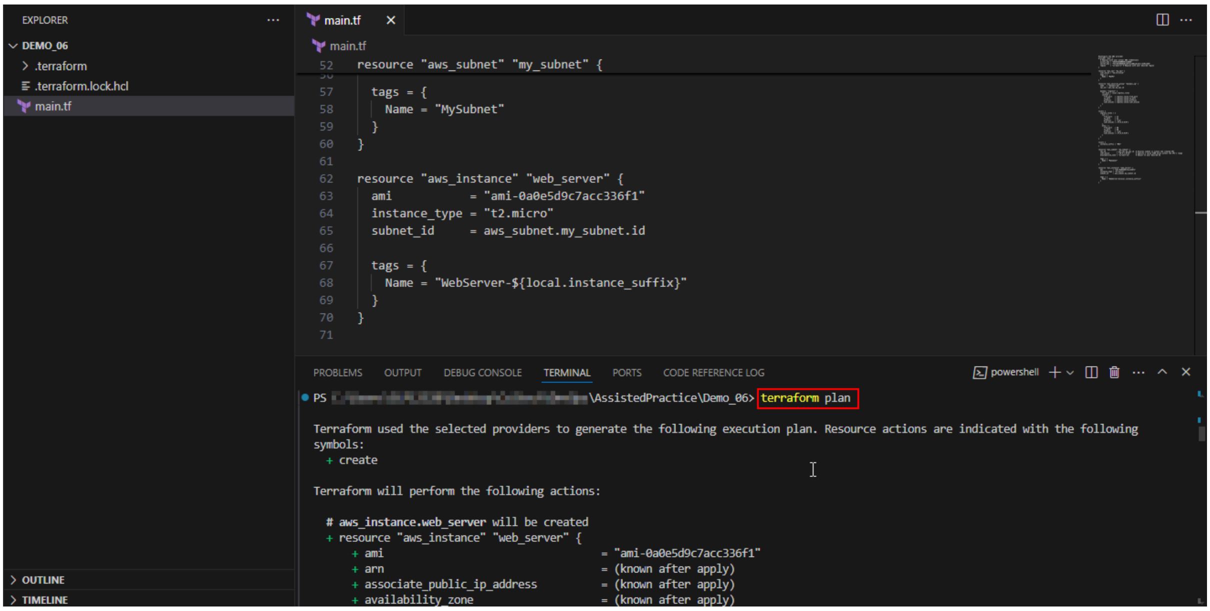


```
resource "aws_subnet" "my_subnet" {
  availability_zone = "us-east-1a"       # Adjust to your desired AZ

  tags = {
    Name = "MySubnet"
  }
}

resource "aws_instance" "web_server" {
  ami           = "ami-0a0e5d9c7acc336f1"
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.my_subnet.id

  tags = {
    Name = "WebServer-${local.instance_suffix}"
  }
}
```

Step 4: Apply configuration changes

4.1 Plan the deployment using the following command to see the proposed changes:

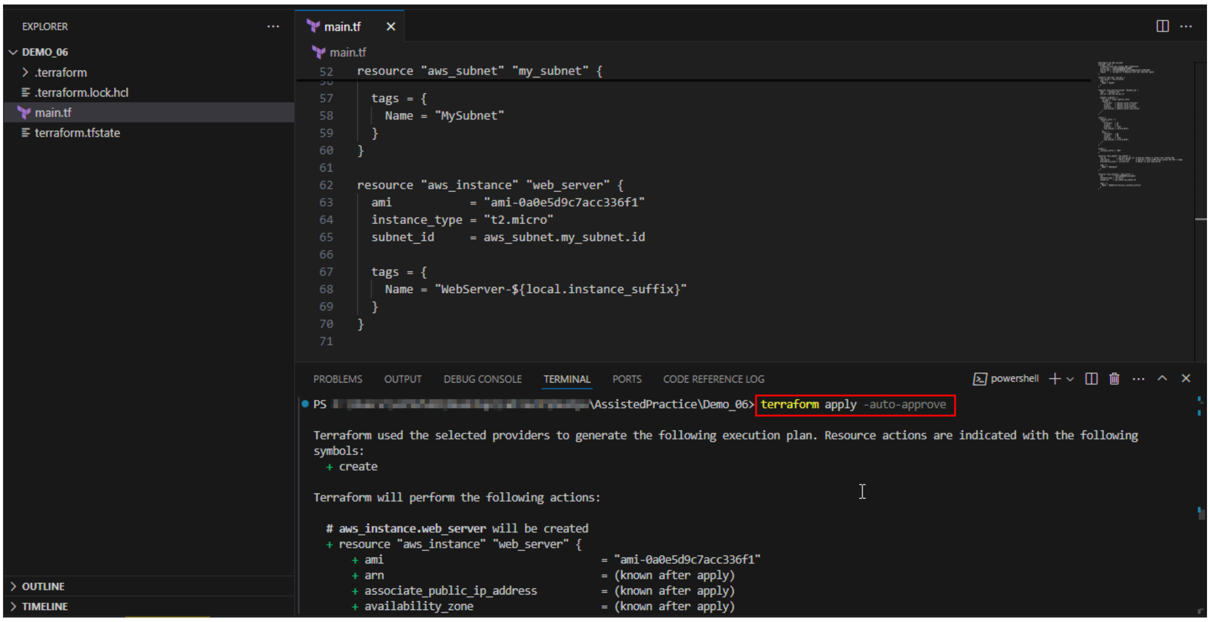
terraform plan



```
resource "aws_subnet" "my_subnet" {

  tags = {
    Name = "MySubnet"
  }
}

resource "aws_instance" "web_server" {
  ami           = "ami-0a0e5d9c7acc336f1"
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.my_subnet.id

  tags = {
    Name = "WebServer-${local.instance_suffix}"
  }
}
```

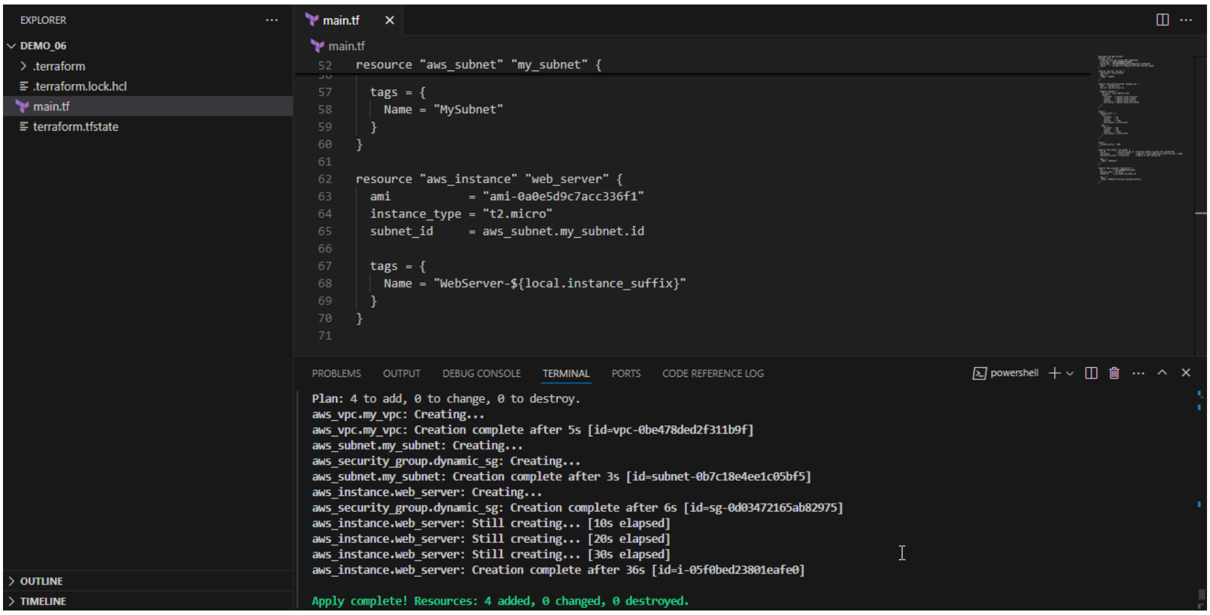
```
PS [redacted]\AssistedPractice\Demo_06> terraform plan

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_instance.web_server will be created
  + resource "aws_instance" "web_server" {
      + ami                                  = "ami-0a0e5d9c7acc336f1"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
```

4.2 Apply the configuration using the following command to deploy the changes as shown in the screenshot below:
terraform apply -auto-approve



```
resource "aws_subnet" "my_subnet" {
  tags = {
    Name = "MySubnet"
  }
}

resource "aws_instance" "web_server" {
  ami           = "ami-0a0e5d9c7acc336f1"
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.my_subnet.id

  tags = {
    Name = "WebServer-${local.instance_suffix}"
  }
}
```

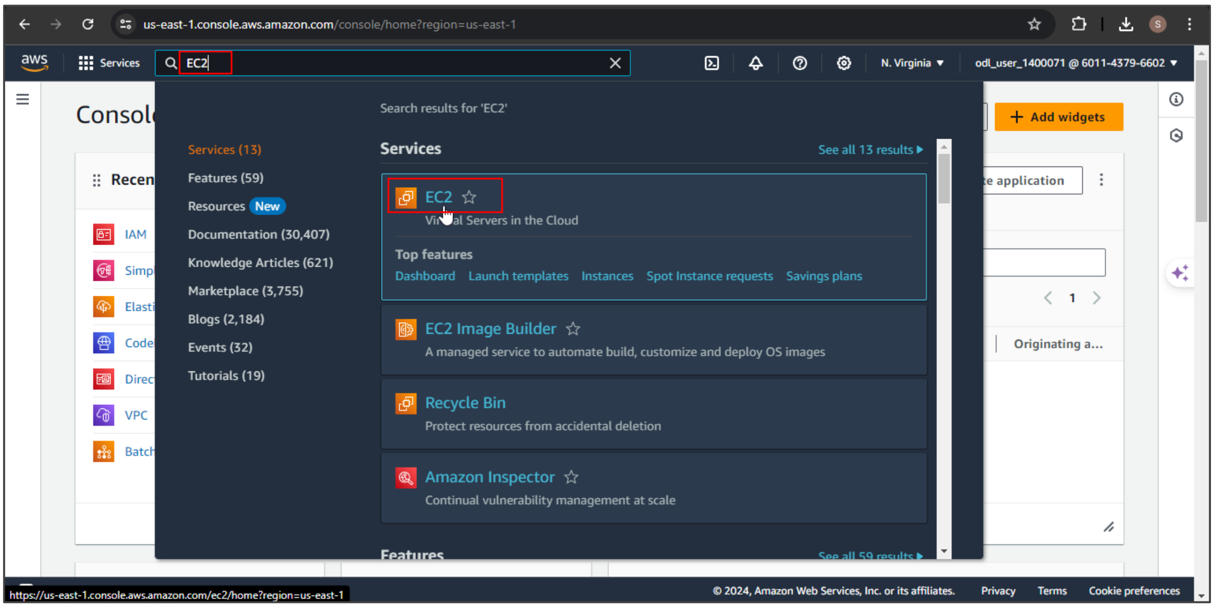
```
PS [redacted]\AssistedPractice\Demo_06> terraform apply -auto-approve

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_instance.web_server will be created
  + resource "aws_instance" "web_server" {
      + ami                                  = "ami-0a0e5d9c7acc336f1"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
```



```
resource "aws_subnet" "my_subnet" {
  tags = {
    Name = "MySubnet"
  }
}

resource "aws_instance" "web_server" {
  ami           = "ami-0a0e5d9c7acc336f1"
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.my_subnet.id

  tags = {
    Name = "WebServer-${local.instance_suffix}"
  }
}
```

```
Plan: 4 to add, 0 to change, 0 to destroy.
aws_vpc.my_vpc: Creating...
aws_vpc.my_vpc: Creation complete after 5s [id=vpc-0be478ded2f311b9f]
aws_subnet.my_subnet: Creating...
aws_security_group.dynamic_sg: Creating...
aws_subnet.my_subnet: Creation complete after 3s [id=subnet-0b7c18e4ee1c05bf5]
aws_instance.web_server: Creating...
aws_security_group.dynamic_sg: Creation complete after 6s [id=sg-0d03472165ab82975]
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 36s [id=i-05f0bed23801eafe0]

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.
```

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



us-east-1.console.aws.amazon.com/console/home?region=us-east-1

Search results for 'EC2'

EC2 — Virtual Servers in the Cloud

Top features: Dashboard, Launch templates, Instances, Spot Instance requests, Savings plans

EC2 Image Builder — A managed service to automate build, customize and deploy OS images

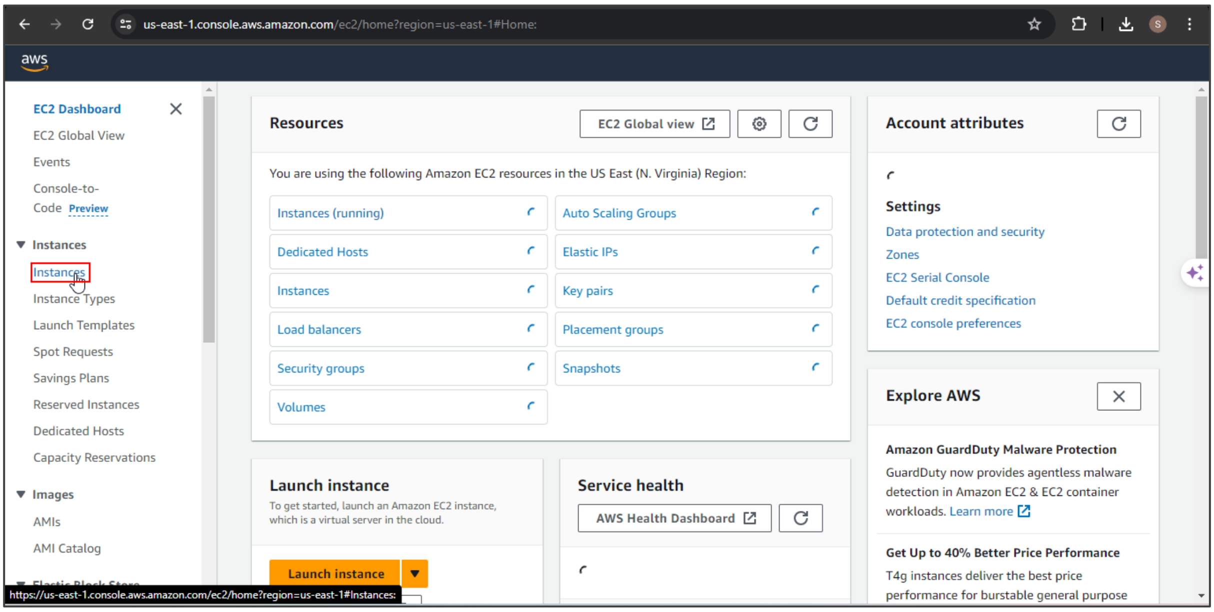
Recycle Bin — Protect resources from accidental deletion

Amazon Inspector — Continual vulnerability management at scale

https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1

© 2024, Amazon Web Services, Inc. or its affiliates.

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



us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Home:

Resources

You are using the following Amazon EC2 resources in the US East (N. Virginia) Region:

Launch instance

To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.

Amazon GuardDuty Malware Protection

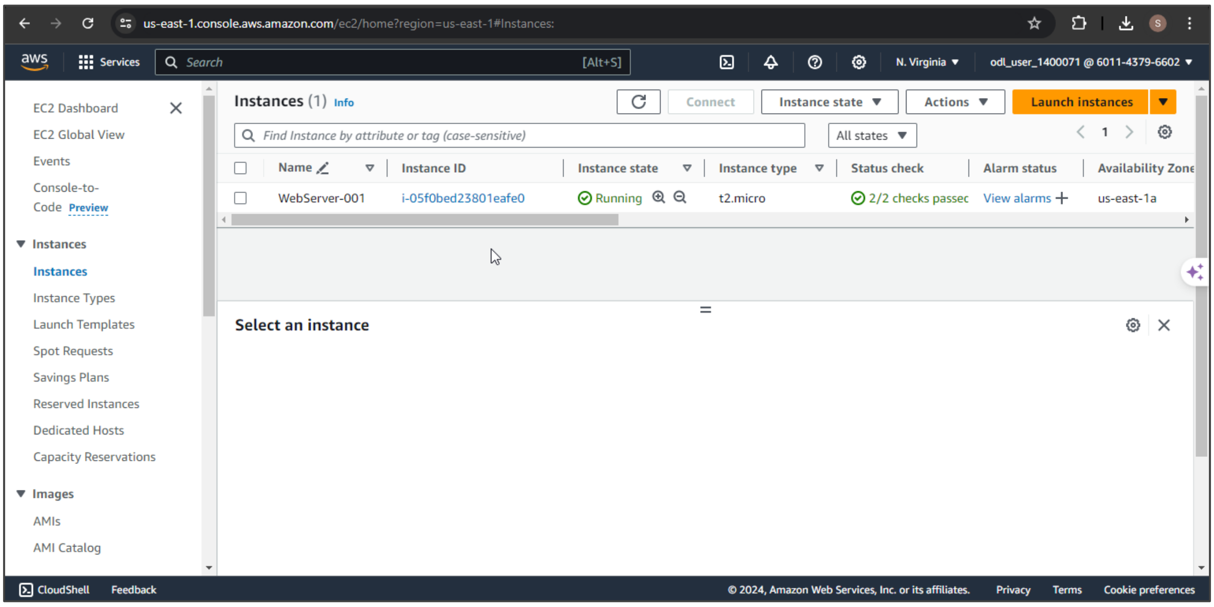
GuardDuty now provides agentless malware detection in Amazon EC2 & EC2 container workloads. Learn more

Get Up to 40% Better Price Performance

T4g instances deliver the best price performance for burstable general purpose

https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:

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



us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:

□	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
□	WebServer-001	i-05f0bed23801eafe0	✓ Running	t2.micro	✓ 2/2 checks passec	View alarms +	us-east-1a

By following these steps, you have successfully implemented dynamic blocks and local variables in Terraform for efficient and flexible infrastructure configuration management.