Lesson 11 Demo 05

Implementing Terraform Built-in Functions

Objective: To implement Terraform built-in functions to manipulate and manage data efficiently in infrastructure configurations

Tools required: Terraform, AWS, and Visual Studio Code

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

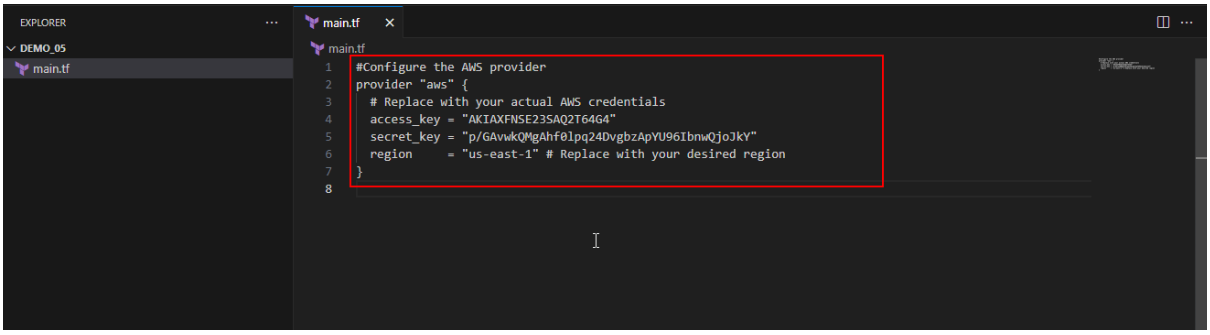
Steps to be followed:

1. Utilize basic numerical functions
2. Manipulate strings using Terraform functions
3. Implement the cidrsubnet function to create subnets

Step 1: Utilize basic numerical functions

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

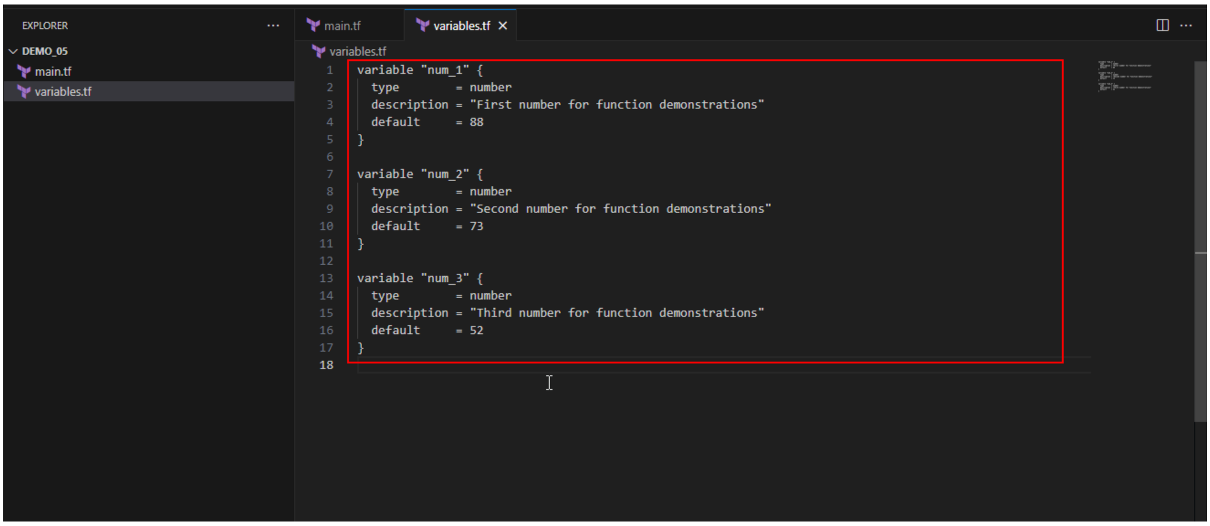
```
#Configure the AWS provider
provider "aws" {
  # Replace with your actual AWS credentials
  access_key = "YOUR_ACCESS_KEY"
  secret_key = "YOUR_SECRET_KEY"
  region     = "us-east-1" # Replace with your desired region
}
```



```
#Configure the AWS provider
provider "aws" {
  # Replace with your actual AWS credentials
  access_key = "AKIAXFNSE23SAQ2T64G4"
  secret_key = "p/GAvwkQMgAhf0lpq24DvgbzApYU96IbnwQjoJkY"
  region     = "us-east-1" # Replace with your desired region
}
```

1.2 Create a file named **variables.tf** with the following numerical variables:

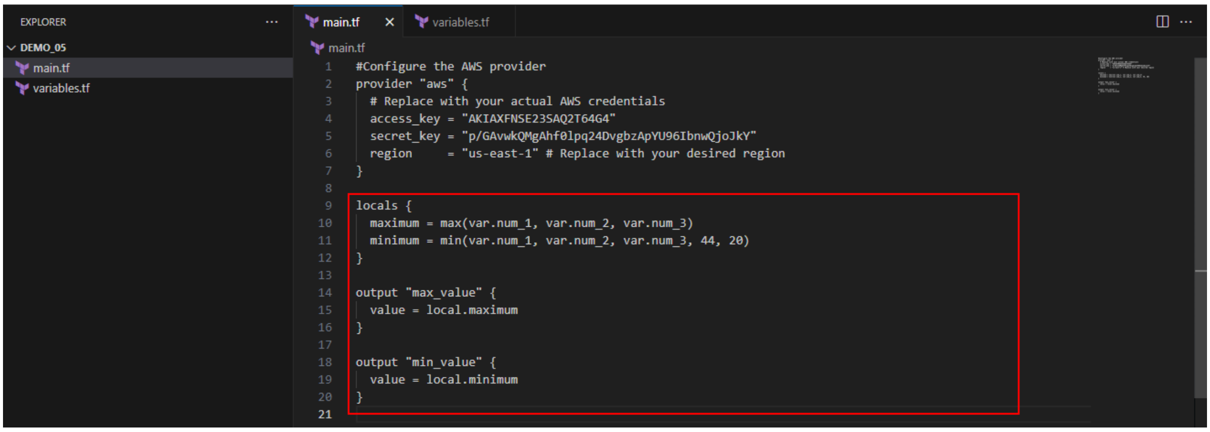
```
variable "num_1" {
  type        = number
  description = "First number for function demonstrations"
  default     = 88
}

variable "num_2" {
  type        = number
  description = "Second number for function demonstrations"
  default     = 73
}

variable "num_3" {
  type        = number
  description = "Third number for function demonstrations"
  default     = 52
}
```



```
variable "num_1" {
  type        = number
  description = "First number for function demonstrations"
  default     = 88
}

variable "num_2" {
  type        = number
  description = "Second number for function demonstrations"
  default     = 73
}

variable "num_3" {
  type        = number
  description = "Third number for function demonstrations"
  default     = 52
}
```

1.3 Add a local variable block in the **main.tf** file to use numerical functions for finding maximum and minimum values as shown in the screenshot below:

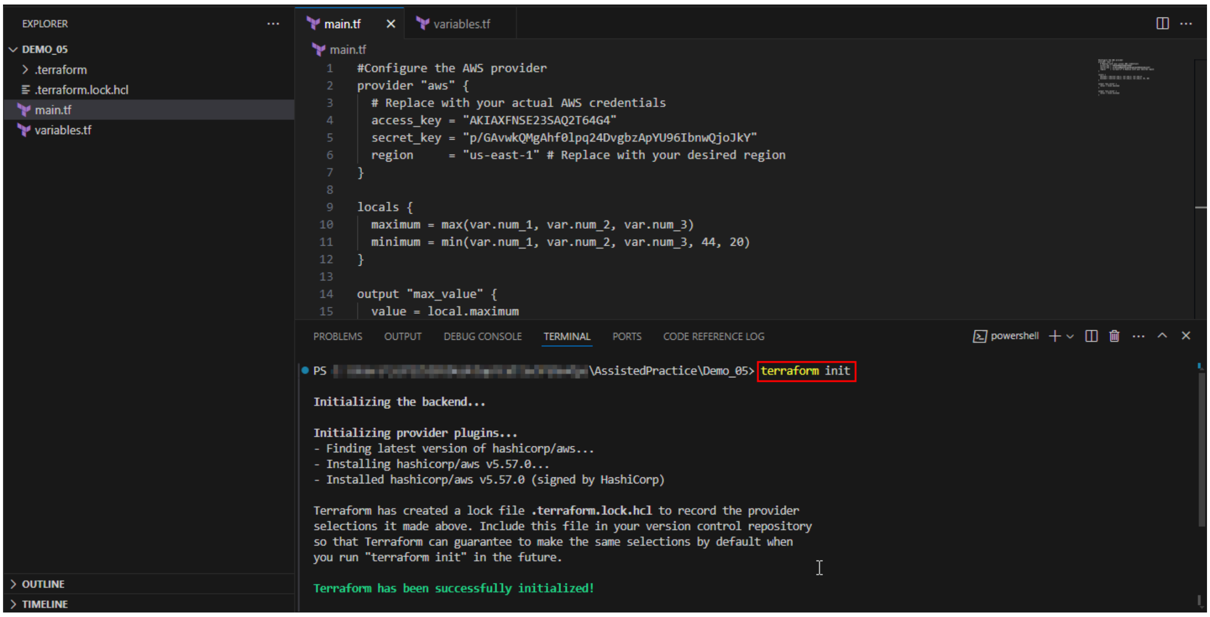
```
locals {
  maximum = max(var.num_1, var.num_2, var.num_3)
  minimum = min(var.num_1, var.num_2, var.num_3, 44, 20)
}

output "max_value" {
  value = local.maximum
}

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



```
#Configure the AWS provider
provider "aws" {
  # Replace with your actual AWS credentials
  access_key = "AKIAXFNSE23SAQ2T64G4"
  secret_key = "p/GAvwkQMgAhf0lpq24DvgbzApYU96IbnwQjoJkY"
  region     = "us-east-1" # Replace with your desired region
}

locals {
  maximum = max(var.num_1, var.num_2, var.num_3)
  minimum = min(var.num_1, var.num_2, var.num_3, 44, 20)
}

output "max_value" {
  value = local.maximum
}

output "min_value" {
  value = local.minimum
}

```

1.4 Initialize the Terraform configuration using the following command:

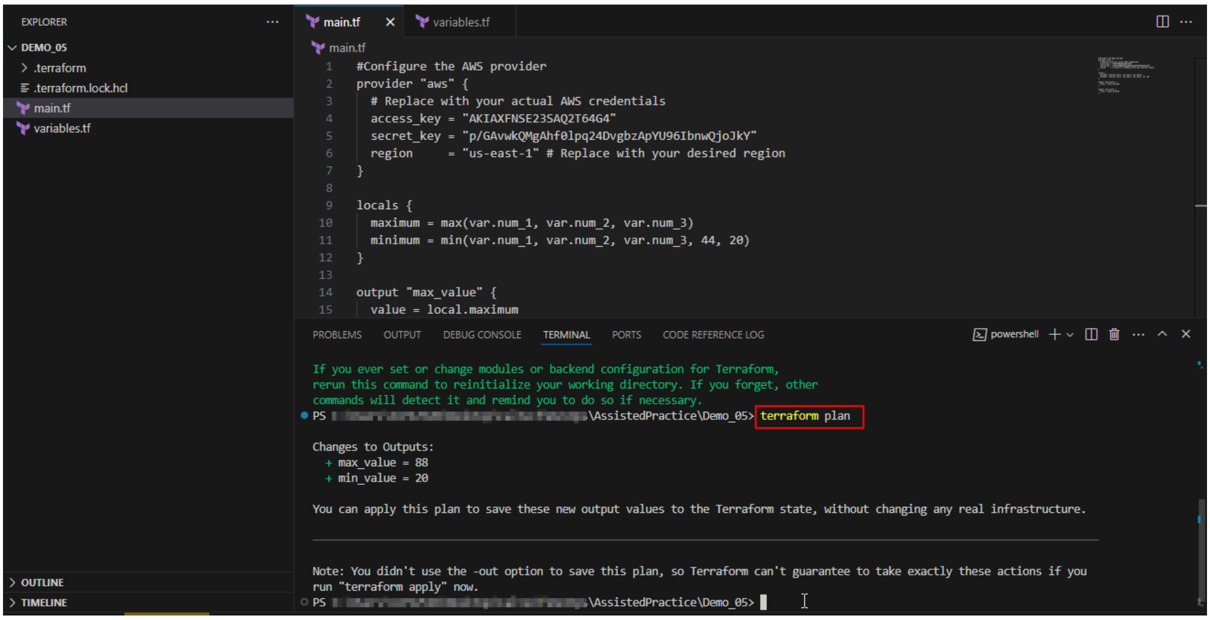
terraform init



```
#Configure the AWS provider
provider "aws" {
  # Replace with your actual AWS credentials
  access_key = "AKIAXFNSE23SAQ2T64G4"
  secret_key = "p/GAvwkQMgAhf0lpq24DvgbzApYU96IbnwQjoJkY"
  region     = "us-east-1" # Replace with your desired region
}

locals {
  maximum = max(var.num_1, var.num_2, var.num_3)
  minimum = min(var.num_1, var.num_2, var.num_3, 44, 20)
}

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

```
PS [redacted]\AssistedPractice\Demo_05> 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
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!
```

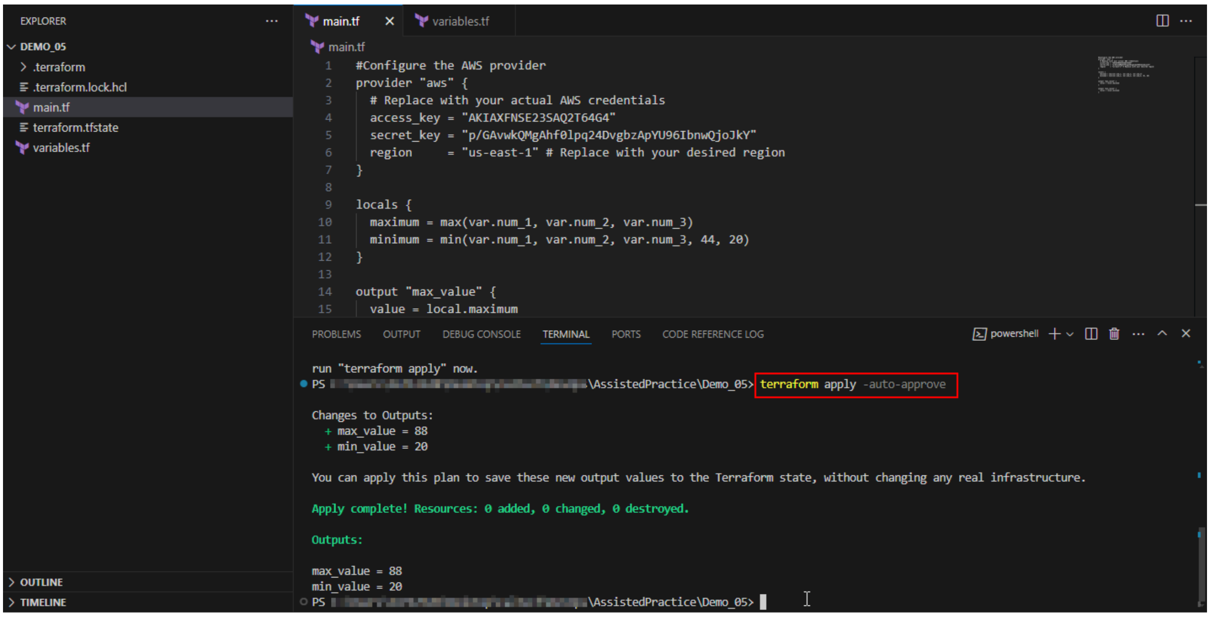
1.5 Plan the changes using the following command as shown in the screenshot below:
terraform plan



```
#Configure the AWS provider
provider "aws" {
  # Replace with your actual AWS credentials
  access_key = "AKIAXFNSE23SAQ2T64G4"
  secret_key = "p/GAvwkQMgAhf0lpq24DvgbzApYU96IbnwQjoJkY"
  region     = "us-east-1" # Replace with your desired region
}

locals {
  maximum = max(var.num_1, var.num_2, var.num_3)
  minimum = min(var.num_1, var.num_2, var.num_3, 44, 20)
}

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

```
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.
PS [redacted]\AssistedPractice\Demo_05> terraform plan

Changes to Outputs:
  + max_value = 88
  + min_value = 20

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

─────────────────────────────────────────────────────────────────────────────

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.
PS [redacted]\AssistedPractice\Demo_05>
```

1.6 Execute the following command to view the results of the numerical functions through outputs:
terraform apply -auto-approve



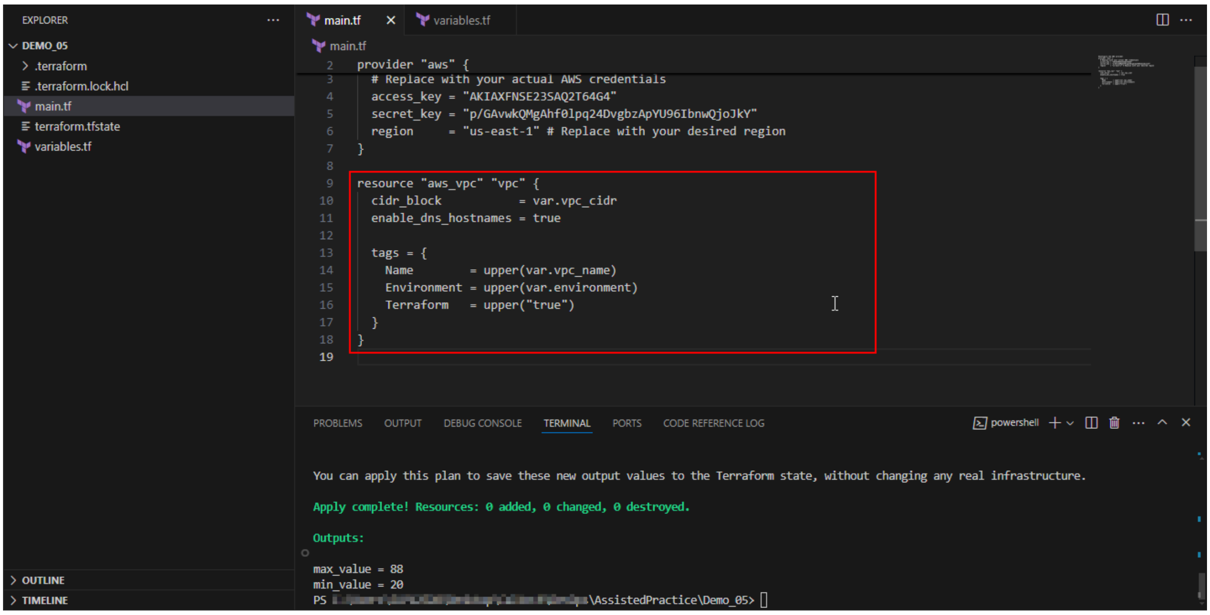
```
#Configure the AWS provider
provider "aws" {
  # Replace with your actual AWS credentials
  access_key = "AKIAXFNSE23SAQ2T64G4"
  secret_key = "p/GAvwkQMgAhf0lpq24DvgbzApYU96IbnwQjoJkY"
  region     = "us-east-1" # Replace with your desired region
}

locals {
  maximum = max(var.num_1, var.num_2, var.num_3)
  minimum = min(var.num_1, var.num_2, var.num_3, 44, 20)
}

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

```
run "terraform apply" now.
PS [redacted]\AssistedPractice\Demo_05> terraform apply -auto-approve

Changes to Outputs:
  + max_value = 88
  + min_value = 20

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.

Outputs:

max_value = 88
min_value = 20
PS [redacted]\AssistedPractice\Demo_05>
```

Step 2: Manipulate strings using Terraform functions

2.1 Modify the VPC resource in **main.tf** to use string functions for transforming tag values:

```
resource "aws_vpc" "vpc" {
  cidr_block           = var.vpc_cidr
  enable_dns_hostnames = true

  tags = {
    Name        = upper(var.vpc_name)
    Environment = upper(var.environment)
    Terraform   = upper("true")
  }
}
```

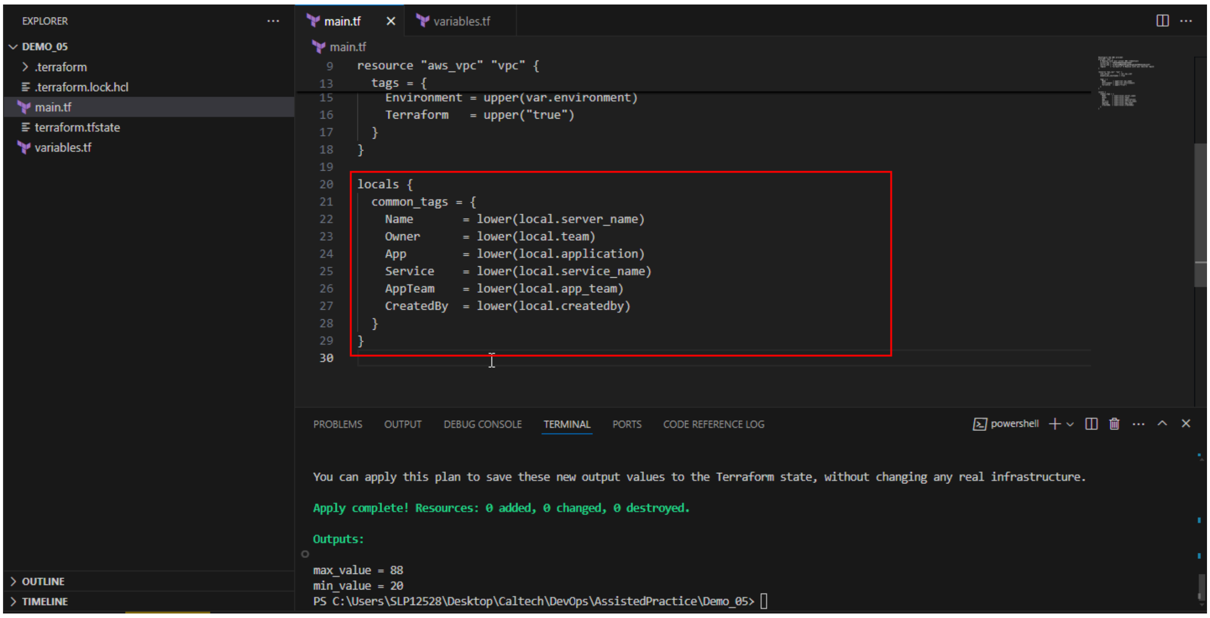


```
provider "aws" {
  # Replace with your actual AWS credentials
  access_key = "AKIAXFNSE23SAQ2T64G4"
  secret_key = "p/GAvwkQMgAhf0lpq24DvgbzApYU96IbnwQjoJkY"
  region     = "us-east-1" # Replace with your desired region
}

resource "aws_vpc" "vpc" {
  cidr_block           = var.vpc_cidr
  enable_dns_hostnames = true

  tags = {
    Name        = upper(var.vpc_name)
    Environment = upper(var.environment)
    Terraform   = upper("true")
  }
}
```

```
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.

Outputs:

max_value = 88
min_value = 20
PS [redacted]\AssistedPractice\Demo_05>
```

2.2 Adjust the locals block in **main.tf** to ensure all tags are in lowercase, making use of the **lower** string function:

```
locals {
  common_tags = {
    Name       = lower(local.server_name)
    Owner      = lower(local.team)
    App        = lower(local.application)
    Service    = lower(local.service_name)
    AppTeam    = lower(local.app_team)
    CreatedBy  = lower(local.createdby)
  }
}
```

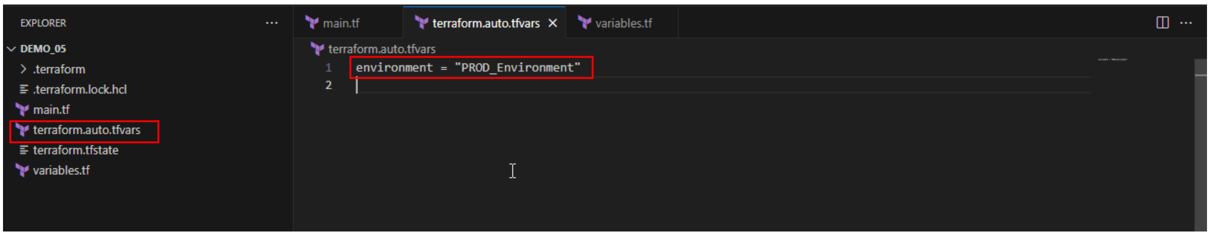


```
resource "aws_vpc" "vpc" {
  tags = {
    Environment = upper(var.environment)
    Terraform   = upper("true")
  }
}

locals {
  common_tags = {
    Name        = lower(local.server_name)
    Owner       = lower(local.team)
    App         = lower(local.application)
    Service     = lower(local.service_name)
    AppTeam     = lower(local.app_team)
    CreatedBy   = lower(local.createdby)
  }
}
```

```
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.

Outputs:

max_value = 88
min_value = 20
PS C:\Users\SLP12528\Desktop\Caltech\DevOps\AssistedPractice\Demo_05>
```

2.3 Create the **terraform.auto.tfvars** file in the working directory, and add the following key-value pair as shown in the screenshot below:

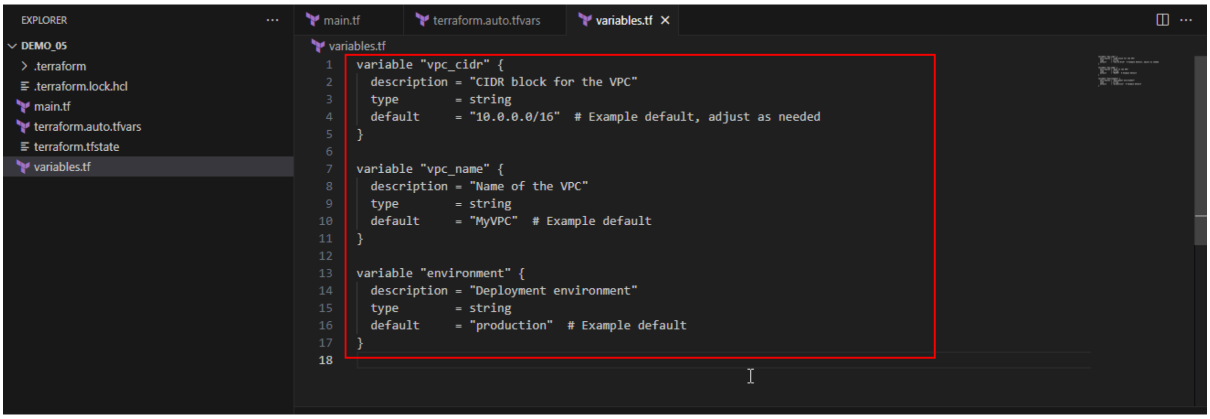
```
environment = "PROD_Environment"
```



```
environment = "PROD_Environment"
```

2.4 Update the **variables.tf** file with the following block as shown in the screenshot below:

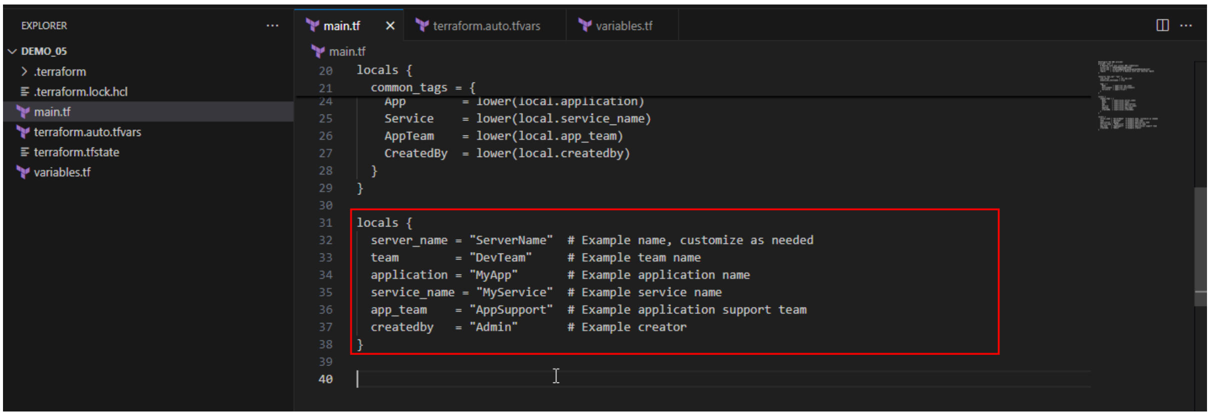
```
variable "vpc_cidr" {
  description = "CIDR block for the VPC"
  type        = string
  default     = "10.0.0.0/16"  # Example default, adjust as needed
}

variable "vpc_name" {
  description = "Name of the VPC"
  type        = string
  default     = "MyVPC"  # Example default
}

variable "environment" {
  description = "Deployment environment"
  type        = string
  default     = "production"  # Example default
}
```



```
variable "vpc_cidr" {
  description = "CIDR block for the VPC"
  type        = string
  default     = "10.0.0.0/16"  # Example default, adjust as needed
}

variable "vpc_name" {
  description = "Name of the VPC"
  type        = string
  default     = "MyVPC"  # Example default
}

variable "environment" {
  description = "Deployment environment"
  type        = string
  default     = "production"  # Example default
}
```

2.5 Declare local variables in **main.tf** file as shown in the screenshot below:

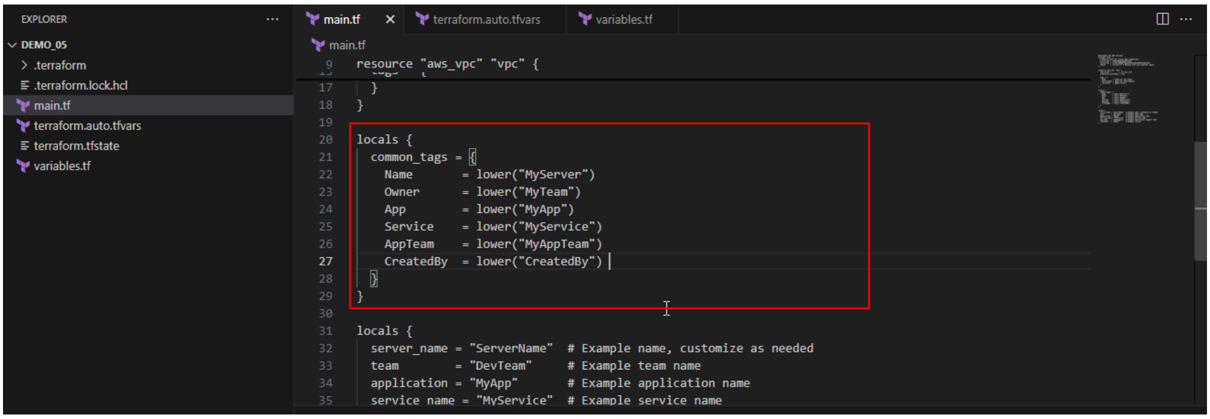
```
locals {
  server_name = "ServerName"  # Example name, customize as needed
  team        = "DevTeam"     # Example team name
  application = "MyApp"       # Example application name
  service_name = "MyService"  # Example service name
  app_team    = "AppSupport"  # Example application support team
  createdby   = "Admin"       # Example creator
}
```



```
locals {
  common_tags = {
    App         = lower(local.application)
    Service     = lower(local.service_name)
    AppTeam     = lower(local.app_team)
    CreatedBy   = lower(local.createdby)
  }
}

locals {
  server_name = "ServerName"  # Example name, customize as needed
  team        = "DevTeam"     # Example team name
  application = "MyApp"       # Example application name
  service_name = "MyService"  # Example service name
  app_team    = "AppSupport"  # Example application support team
  createdby   = "Admin"       # Example creator
}

```

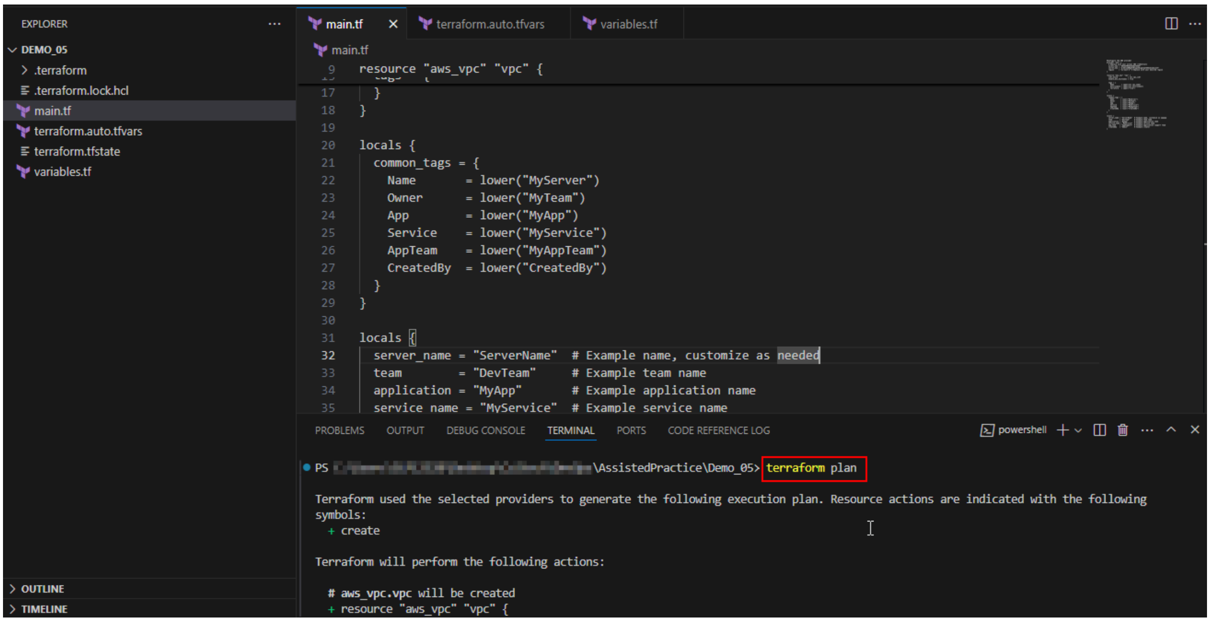
2.6 Update the locals referencing the values in **main.tf** as shown in the screenshot below:

```
locals {
  common_tags = {
    Name       = lower("MyServer")
    Owner      = lower("MyTeam")
    App        = lower("MyApp")
    Service    = lower("MyService")
    AppTeam    = lower("MyAppTeam")
    CreatedBy  = lower("CreatedBy")
  }
}
```



```
resource "aws_vpc" "vpc" {
  }
}

locals {
  common_tags = {
    Name        = lower("MyServer")
    Owner       = lower("MyTeam")
    App         = lower("MyApp")
    Service     = lower("MyService")
    AppTeam     = lower("MyAppTeam")
    CreatedBy   = lower("CreatedBy")
  }
}

locals {
  server_name = "ServerName"  # Example name, customize as needed
  team        = "DevTeam"     # Example team name
  application = "MyApp"       # Example application name
  service_name = "MyService"  # Example service name
```

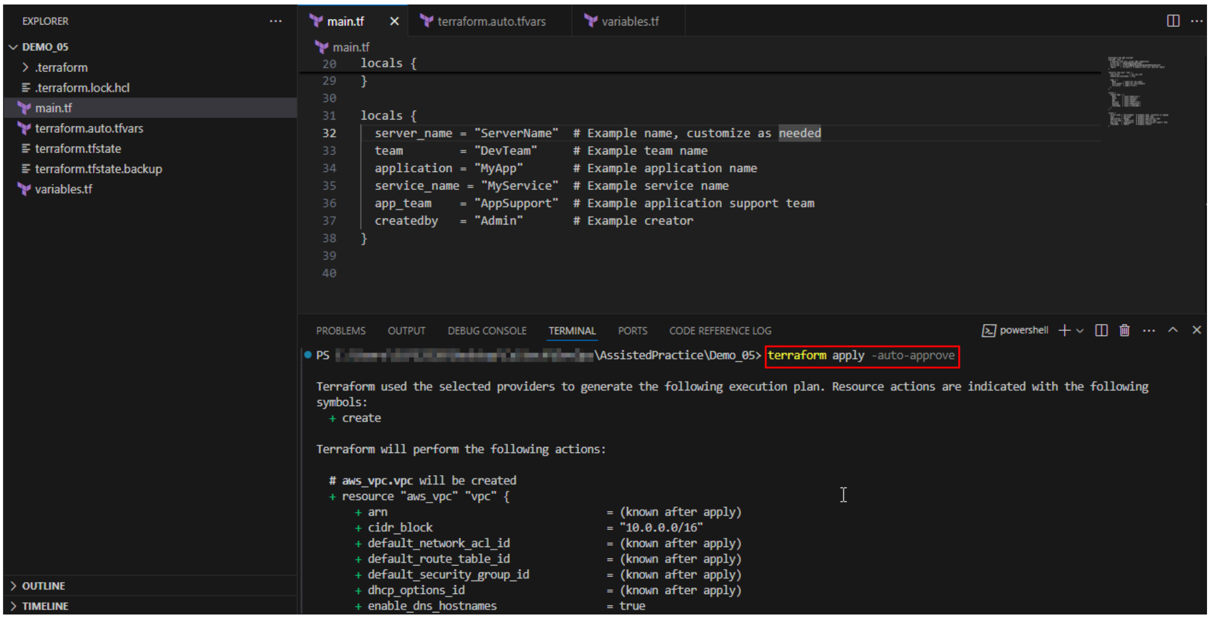
2.7 Plan the deployment using the following command to see the proposed changes:
terraform plan



```
resource "aws_vpc" "vpc" {
  }
}

locals {
  common_tags = {
    Name       = lower("MyServer")
    Owner      = lower("MyTeam")
    App        = lower("MyApp")
    Service    = lower("MyService")
    AppTeam    = lower("MyAppTeam")
    CreatedBy  = lower("CreatedBy")
  }
}

locals {
  server_name = "ServerName"  # Example name, customize as needed
  team        = "DevTeam"     # Example team name
  application = "MyApp"       # Example application name
  service_name = "MyService"  # Example service name
```

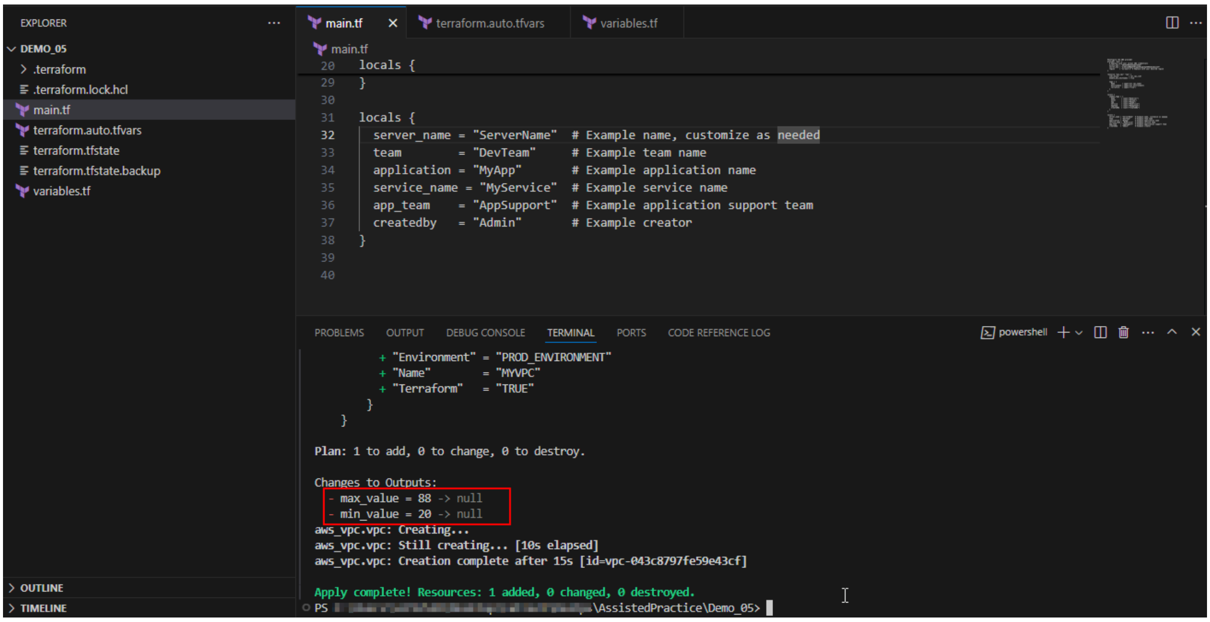
```
PS [redacted]\AssistedPractice\Demo_05> terraform plan

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_vpc.vpc will be created
  + resource "aws_vpc" "vpc" {
```

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



```
locals {
}

locals {
  server_name = "ServerName"  # Example name, customize as needed
  team        = "DevTeam"     # Example team name
  application = "MyApp"       # Example application name
  service_name = "MyService"  # Example service name
  app_team    = "AppSupport"  # Example application support team
  createdby   = "Admin"       # Example creator
}
```

```
PS [redacted]\AssistedPractice\Demo_05> 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_vpc.vpc will be created
  + resource "aws_vpc" "vpc" {
      + arn                                  = (known after apply)
      + cidr_block                           = "10.0.0.0/16"
      + default_network_acl_id               = (known after apply)
      + default_route_table_id               = (known after apply)
      + default_security_group_id            = (known after apply)
      + dhcp_options_id                      = (known after apply)
      + enable_dns_hostnames                 = true
```



```
locals {
}

locals {
  server_name = "ServerName"  # Example name, customize as needed
  team        = "DevTeam"     # Example team name
  application = "MyApp"       # Example application name
  service_name = "MyService"  # Example service name
  app_team    = "AppSupport"  # Example application support team
  createdby   = "Admin"       # Example creator
}
```

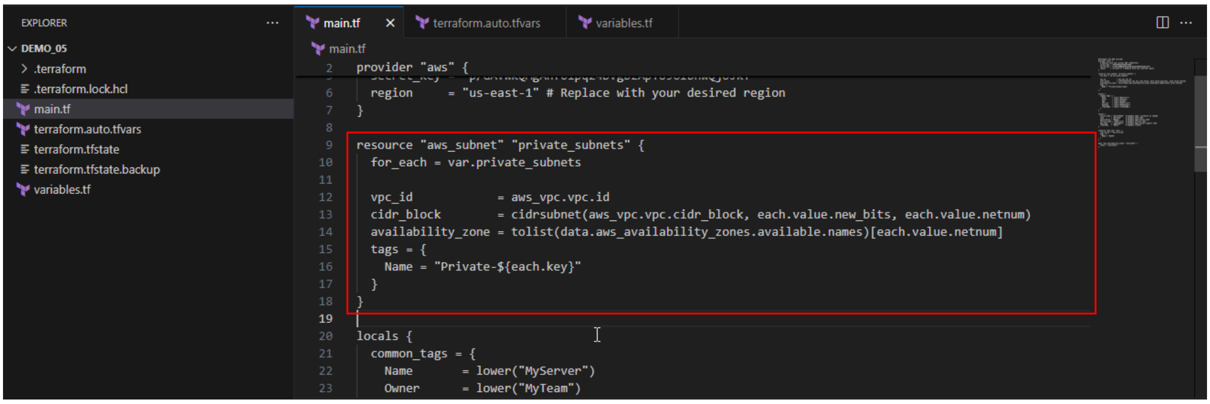
```
          + "Environment" = "PROD_ENVIRONMENT"
          + "Name"        = "MYVPC"
          + "Terraform"   = "TRUE"
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  - max_value = 88 -> null
  - min_value = 20 -> null
aws_vpc.vpc: Creating...
aws_vpc.vpc: Still creating... [10s elapsed]
aws_vpc.vpc: Creation complete after 15s [id=vpc-043c8797fe59e43cf]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
PS [redacted]\AssistedPractice\Demo_05>
```

Step 3: Implement the cidrsubnet function to create subnets

3.1 Modify the subnet creation block in the **main.tf** file to demonstrate the cidrsubnet function:

resource "aws_subnet" "private_subnets" {
for_each = var.private_subnets

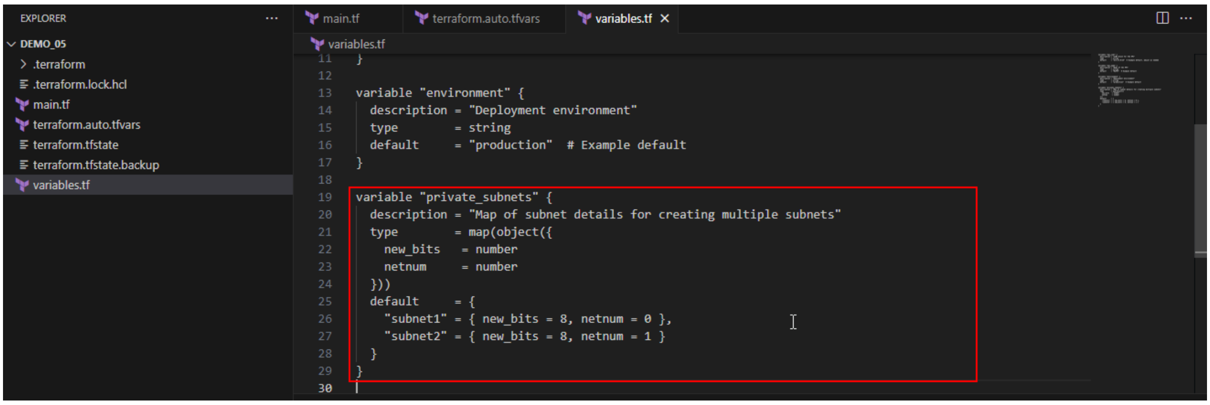
vpc_id = aws_vpc.vpc.id
cidr_block = cidrsubnet(aws_vpc.vpc.cidr_block, each.value.new_bits, each.value.netnum)
availability_zone = tolist(data.aws_availability_zones.available.names)[each.value.netnum]
tags = {
Name = "Private-${each.key}"
}
}
}



```
provider "aws" {
  region     = "us-east-1" # Replace with your desired region
}

resource "aws_subnet" "private_subnets" {
  for_each = var.private_subnets

  vpc_id            = aws_vpc.vpc.id
  cidr_block        = cidrsubnet(aws_vpc.vpc.cidr_block, each.value.new_bits, each.value.netnum)
  availability_zone = tolist(data.aws_availability_zones.available.names)[each.value.netnum]
  tags = {
    Name = "Private-${each.key}"
  }
}

locals {
  common_tags = {
    Name       = lower("MyServer")
    Owner      = lower("MyTeam")
```

3.2 Add this declaration to the **variables.tf** file to declare the **private_subnets** variable as shown in the screenshot below:

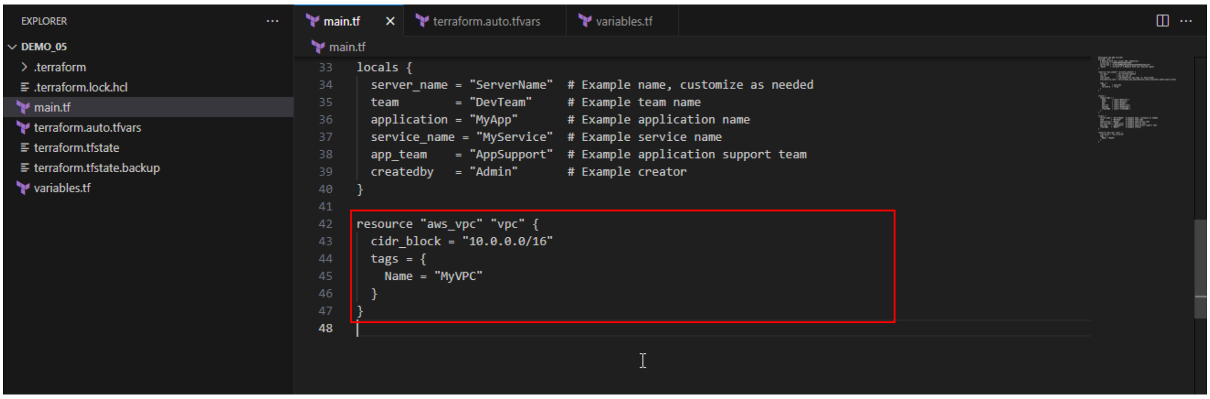
```
variable "private_subnets" {
  description = "Map of subnet details for creating multiple subnets"
  type        = map(object({
    new_bits   = number
    netnum     = number
  }))
  default     = {
    "subnet1" = { new_bits = 8, netnum = 0 },
    "subnet2" = { new_bits = 8, netnum = 1 }
  }
}
```



```
}

variable "environment" {
  description = "Deployment environment"
  type        = string
  default     = "production"  # Example default
}

variable "private_subnets" {
  description = "Map of subnet details for creating multiple subnets"
  type        = map(object({
    new_bits   = number
    netnum     = number
  }))
  default     = {
    "subnet1" = { new_bits = 8, netnum = 0 },
    "subnet2" = { new_bits = 8, netnum = 1 }
  }
}
```

3.3 Add the **aws_vpc** resource in the **main.tf** file as shown in the screenshot below:

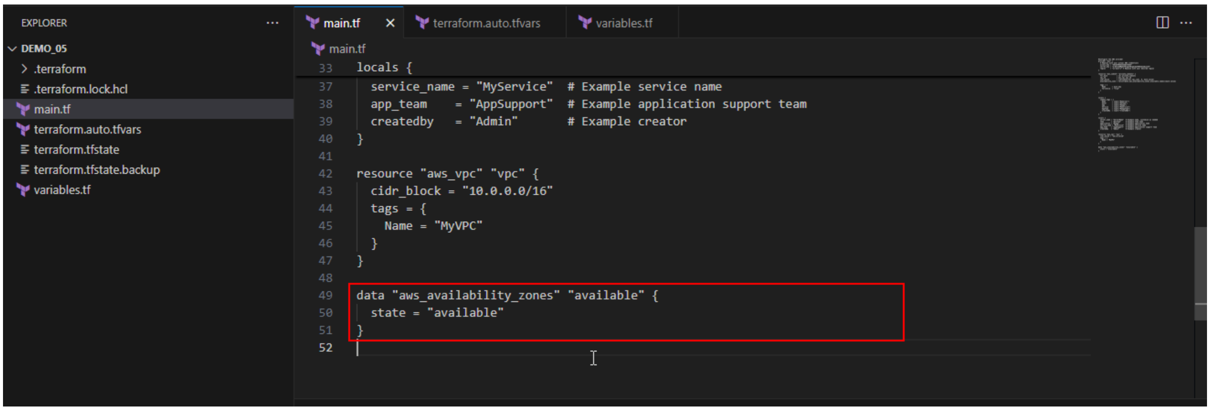
```
resource "aws_vpc" "vpc" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name = "MyVPC"
  }
}
```



```
locals {
  server_name = "ServerName"  # Example name, customize as needed
  team        = "DevTeam"     # Example team name
  application = "MyApp"       # Example application name
  service_name = "MyService"  # Example service name
  app_team    = "AppSupport"  # Example application support team
  createdby   = "Admin"       # Example creator
}

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

3.4 Declare the AWS availability zones data source in the **main.tf** file as shown in the screenshot below:

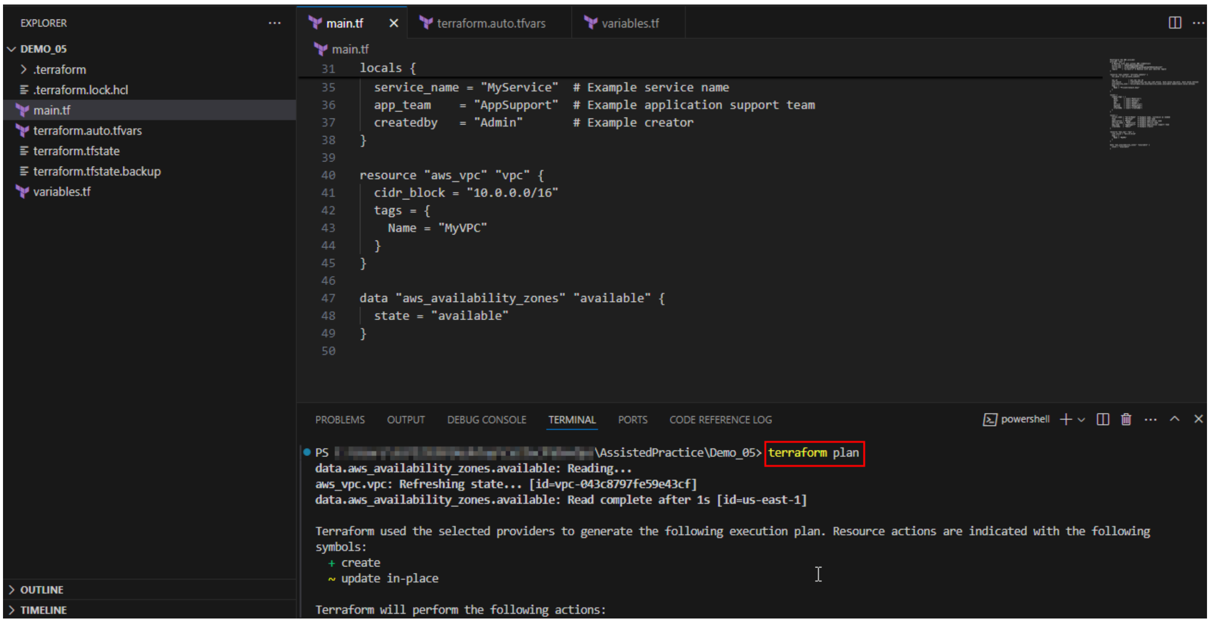
data "aws_availability_zones" "available" {
state = "available"
}



```
locals {
  service_name = "MyService"  # Example service name
  app_team     = "AppSupport"  # Example application support team
  createdby    = "Admin"       # Example creator
}

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

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

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

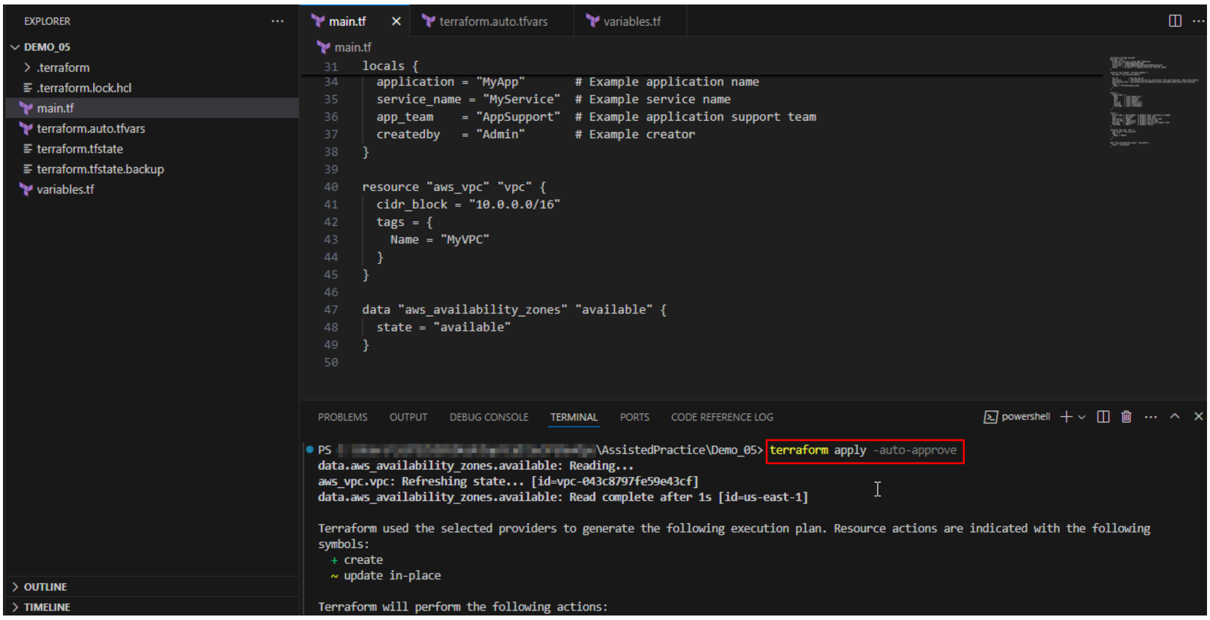
terraform plan



```
locals {
  service_name = "MyService"  # Example service name
  app_team     = "AppSupport"  # Example application support team
  createdby    = "Admin"       # Example creator
}

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

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

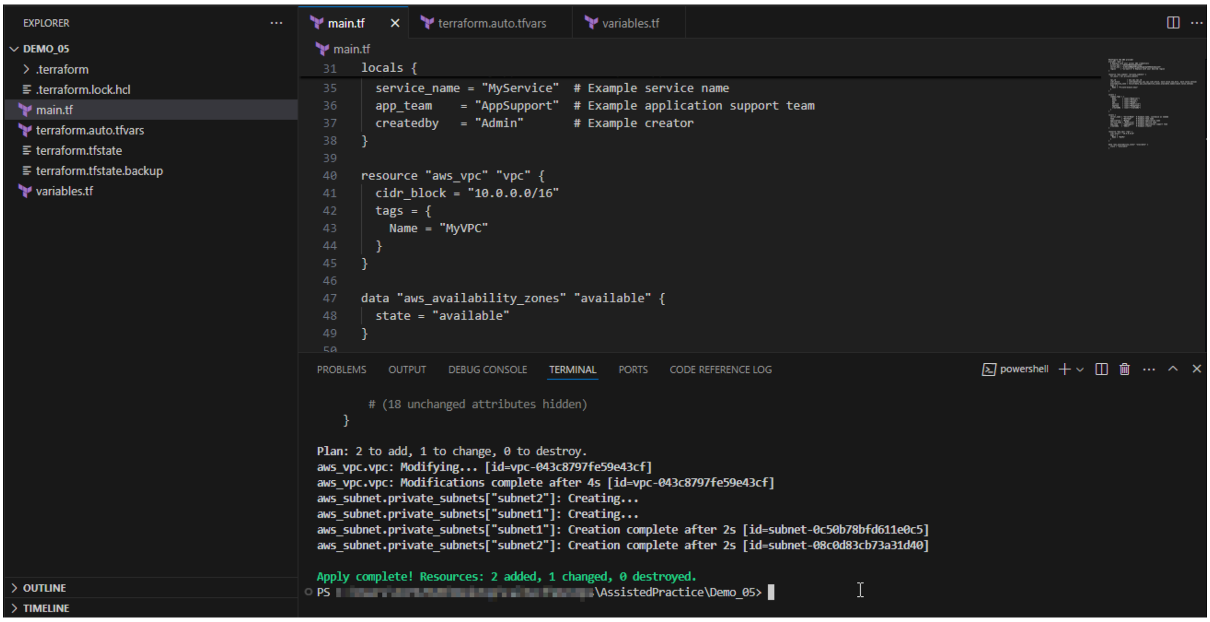
```
PS [redacted]\AssistedPractice\Demo_05> terraform plan
data.aws_availability_zones.available: Reading...
aws_vpc.vpc: Refreshing state... [id=vpc-043c8797fe59e43cf]
data.aws_availability_zones.available: Read complete after 1s [id=us-east-1]

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

Terraform will perform the following actions:
```

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



```
locals {
  application = "MyApp"       # Example application name
  service_name = "MyService"  # Example service name
  app_team     = "AppSupport" # Example application support team
  createdby    = "Admin"      # Example creator
}

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

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

```
PS [redacted]\AssistedPractice\Demo_05> terraform apply -auto-approve
data.aws_availability_zones.available: Reading...
aws_vpc.vpc: Refreshing state... [id=vpc-043c8797fe59e43cf]
data.aws_availability_zones.available: Read complete after 1s [id=us-east-1]

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

Terraform will perform the following actions:
```



```
locals {
  service_name = "MyService"  # Example service name
  app_team     = "AppSupport" # Example application support team
  createdby    = "Admin"      # Example creator
}

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

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

```
        # (18 unchanged attributes hidden)
    }

Plan: 2 to add, 1 to change, 0 to destroy.
aws_vpc.vpc: Modifying... [id=vpc-043c8797fe59e43cf]
aws_vpc.vpc: Modifications complete after 4s [id=vpc-043c8797fe59e43cf]
aws_subnet.private_subnets["subnet2"]: Creating...
aws_subnet.private_subnets["subnet1"]: Creating...
aws_subnet.private_subnets["subnet1"]: Creation complete after 2s [id=subnet-0c50b78bfd611e0c5]
aws_subnet.private_subnets["subnet2"]: Creation complete after 2s [id=subnet-08c0d83cb73a31d40]

Apply complete! Resources: 2 added, 1 changed, 0 destroyed.
PS [redacted]\AssistedPractice\Demo_05>
```

By following these steps, you have successfully implemented Terraform built-in functions to efficiently manipulate and manage data in infrastructure configurations.