System Provisioning

Experiment 1-5

Submitted to:

Hitesh Kumar Sharma

Senior Associate Professor

Submitted By:

Name: Binit Bhushan

SAP ID: 500091356

B. tech CSE DevOps B1

Roll no.: R2142210239

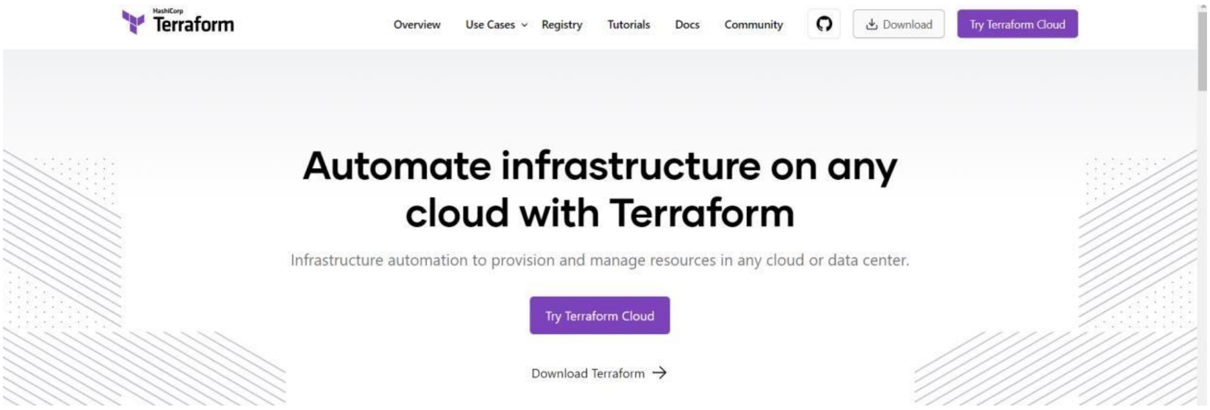
Lab Exercise 1– Install Terraform on Windows

Aim: Installing Terraform on Windows requires you to download the correct Terraform package, unpack, and execute it via the CLI. Follow the instructions below to ensure you do not miss any steps.

Download Terraform File for Windows

To find the latest version of Terraform for Windows:

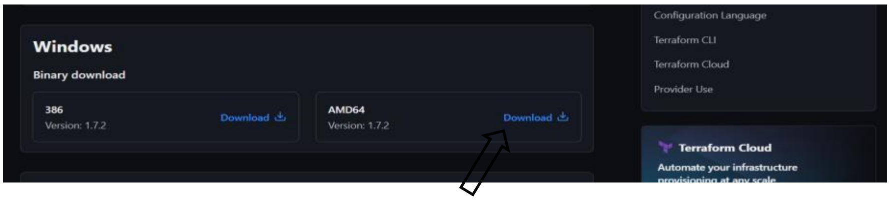
1. **Browse to the Download Terraform page.**



Automate infrastructure on any cloud with Terraform

Infrastructure automation to provision and manage resources in any cloud or data center.

2. **Select the Windows tab under the Operating System heading. The latest versions preselected.**
3. **Choose the binary to download. Select 386 for 32-bit systems or AMD64 for 64-bit systems. Choose the download location for the zip file if the download does not start automatically.**



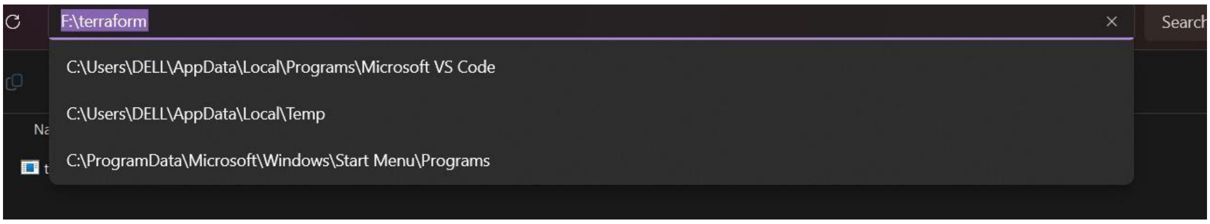
Windows

Binary download

386 Version: 1.7.2 Download

AMD64 Version: 1.7.2 Download

4. **Unzip the downloaded file. For example, use the C:\terraform path. Remember this location so you can add the path to the environment variables.**



F:\terraform

C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code

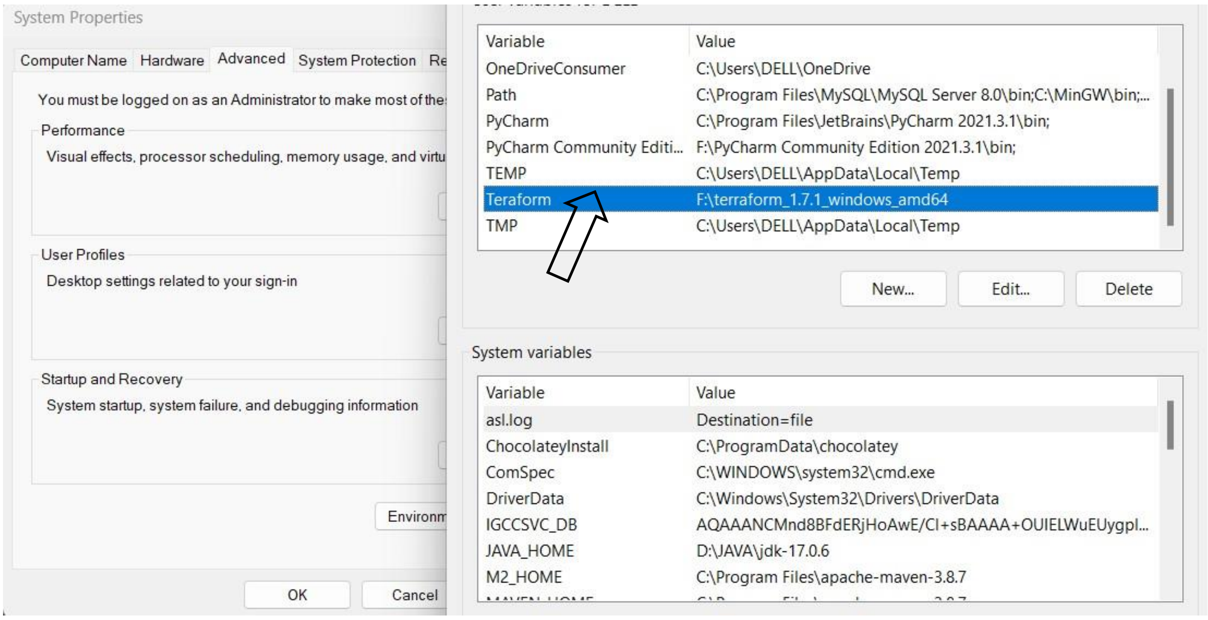
C:\Users\DELL\AppData\Local\Temp

C:\ProgramData\Microsoft\Windows\Start Menu\Programs

Add Terraform Path to System Environment Variables

To add the Terraform executable to the system's global path:

1. **Open the start menu, start typing environment and click Edit system environment variables. The System Properties window opens.**
2. **Click the Environment Variables... button.**
3. **Select the Path variable in the System variables section to add terraform for all accounts. Alternatively, select Path in the User variables section to add terraform for the currently logged-in user only. Click Edit once you select a Path.**
4. **Click New in the edit window and enter the location of the Terraform folder.**



You must be logged on as an Administrator to make most of the

Performance

Visual effects, processor scheduling, memory usage, and virtu

User Profiles

Desktop settings related to your sign-in

Startup and Recovery

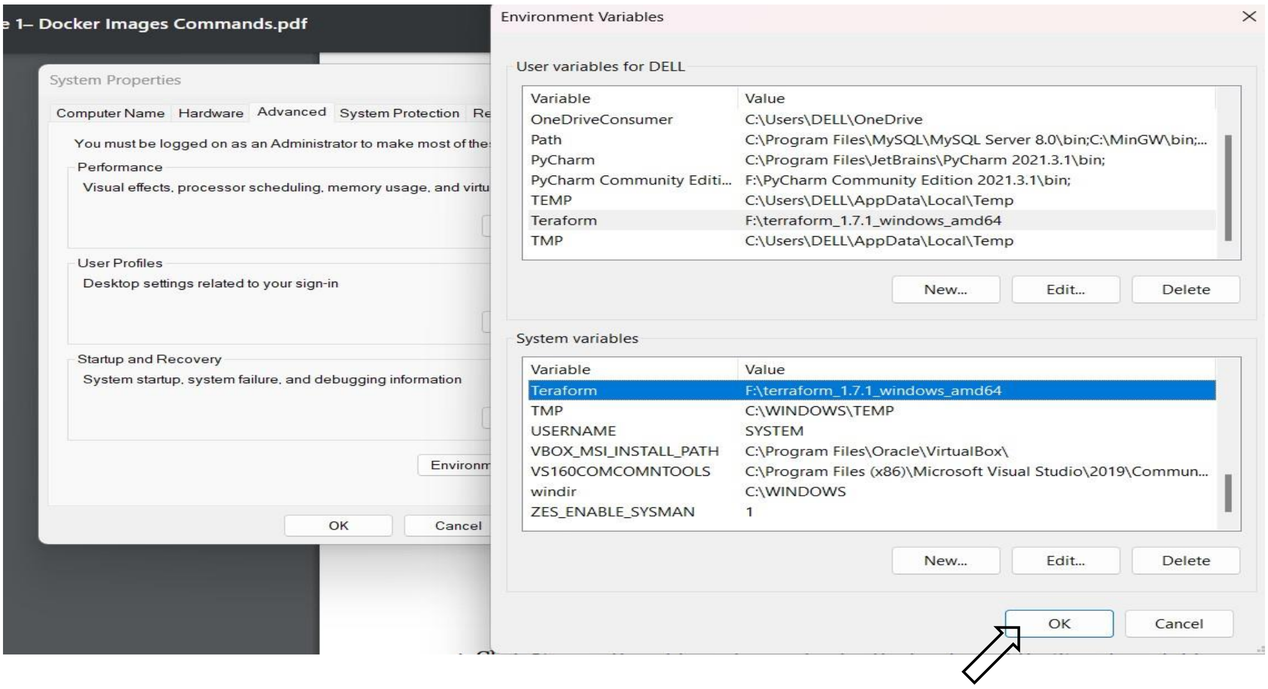
System startup, system failure, and debugging information

Variable	Value
OneDriveConsumer	C:\Users\DELL\OneDrive
Path	C:\Program Files\MySQL\MySQL Server 8.0\bin;C:\MinGW\bin;...
PyCharm	C:\Program Files\JetBrains\PyCharm 2021.3.1\bin;
PyCharm Community Editi...	F:\PyCharm Community Edition 2021.3.1\bin;
TEMP	C:\Users\DELL\AppData\Local\Temp
Teraform	F:\terraform_1.7.1_windows_amd64
TMP	C:\Users\DELL\AppData\Local\Temp

System variables

Variable	Value
asl.log	Destination=file
ChocolateyInstall	C:\ProgramData\chocolatey
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
IGCCSVC_DB	AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAA+OUIELWuEUygpl...
JAVA_HOME	D:\JAVA\jdk-17.0.6
M2_HOME	C:\Program Files\apache-maven-3.8.7

5. **Click OK on all windows to apply the changes.**



You must be logged on as an Administrator to make most of the

Visual effects, processor scheduling, memory usage, and virtu

Desktop settings related to your sign-in

System startup, system failure, and debugging information

User variables for DELL

Variable	Value
OneDriveConsumer	C:\Users\DELL\OneDrive
Path	C:\Program Files\MySQL\MySQL Server 8.0\bin;C:\MinGW\bin;...
PyCharm	C:\Program Files\JetBrains\PyCharm 2021.3.1\bin;
PyCharm Community Editi...	F:\PyCharm Community Edition 2021.3.1\bin;
TEMP	C:\Users\DELL\AppData\Local\Temp
Teraform	F:\terraform_1.7.1_windows_amd64
TMP	C:\Users\DELL\AppData\Local\Temp

System variables

Variable	Value
Teraform	F:\terraform_1.7.1_windows_amd64
TMP	C:\WINDOWS\TEMP
USERNAME	SYSTEM
VBOX_MSI_INSTALL_PATH	C:\Program Files\Oracle\VirtualBox\
VS160COMCOMNTOOLS	C:\Program Files (x86)\Microsoft Visual Studio\2019\Commun...
windir	C:\WINDOWS
ZES_ENABLE_SYSMAN	1

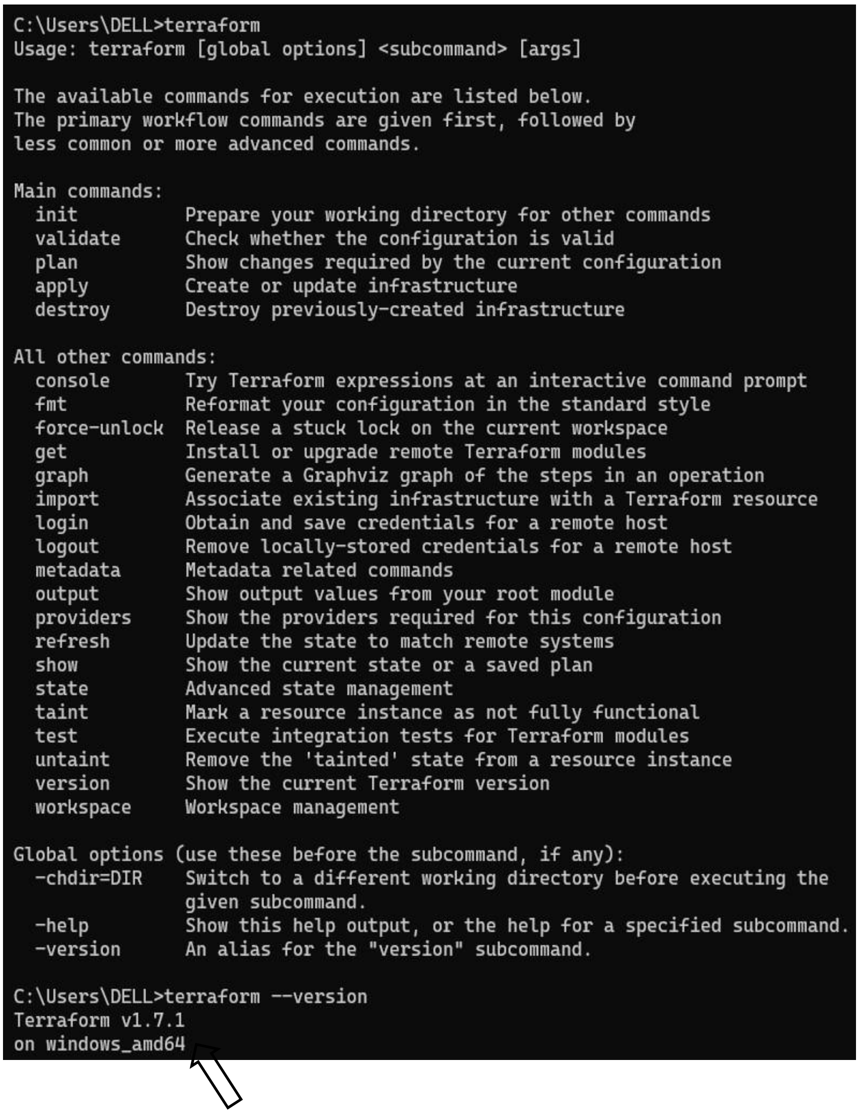
Verify Windows Terraform Installation

To check the Terraform global path configuration:

1. Open a new command-prompt window.

2. Enter the command to check the Terraform version: terraform -version

Cmd= terraform -version



```
C:\Users\DELL>terraform
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
  plan          Show changes required by the current configuration
  apply         Create or update infrastructure
  destroy       Destroy previously-created infrastructure

All other commands:
  console       Try Terraform expressions at an interactive command prompt
  fmt           Reformat your configuration in the standard style
  force-unlock  Release a stuck lock on the current workspace
  get           Install or upgrade remote Terraform modules
  graph         Generate a Graphviz graph of the steps in an operation
  import        Associate existing infrastructure with a Terraform resource
  login         Obtain and save credentials for a remote host
  logout        Remove locally-stored credentials for a remote host
  metadata      Metadata related commands
  output        Show output values from your root module
  providers     Show the providers required for this configuration
  refresh       Update the state to match remote systems
  show          Show the current state or a saved plan
  state         Advanced state management
  taint         Mark a resource instance as not fully functional
  test          Execute integration tests for Terraform modules
  untaint       Remove the 'tainted' state from a resource instance
  version       Show the current Terraform version
  workspace     Workspace management

Global options (use these before the subcommand, if any):
  -chdir=DIR    Switch to a different working directory before executing the
                given subcommand.
  -help         Show this help output, or the help for a specified subcommand.
  -version      An alias for the "version" subcommand.

C:\Users\DELL>terraform --version
Terraform v1.7.1
on windows_amd64
```

The output shows the Terraform version you downloaded and installed on your

Windows machine.

Lab Exercise 2– Terraform AWS Provider and IAM User Setting

Prerequisites: Terraform Installed: Make sure you have Terraform installed on your machine.

Follow the official installation guide if needed.

AWS Credentials: Ensure you have AWS credentials (Access Key ID and Secret Access

Key) configured. You can set them up using the AWS CLI or by setting environment variables.

Exercise Steps:

Step 1: Create a New Directory:

Create a new directory for your Terraform configuration:

mkdir aws-terraform-demo

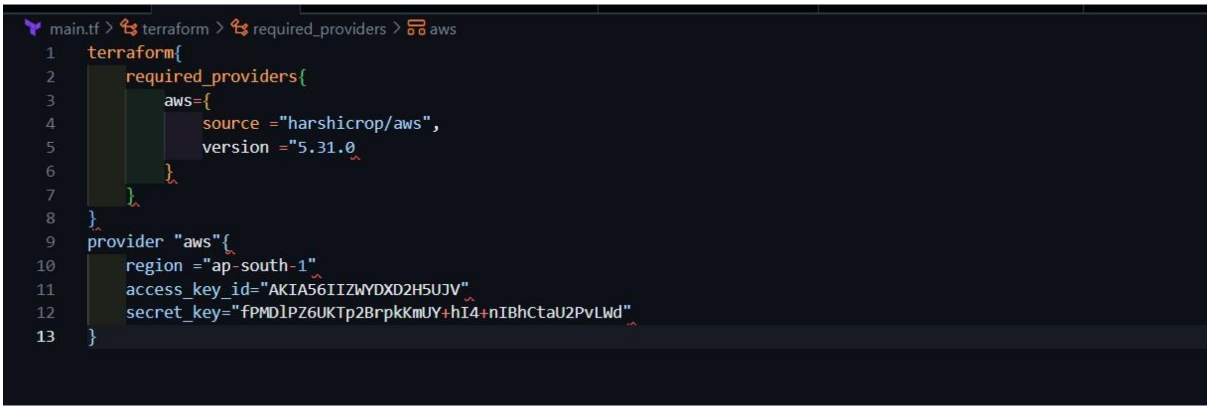
cd aws-terraform-demo

```
2-lab    30-01-2024 09:30 AM    File folder
```

Step 2: Create Terraform Configuration File (main.tf):

```
main.tf    30-01-2024 09:30 AM    TF File
```

Create a file named main.tf with the following content:

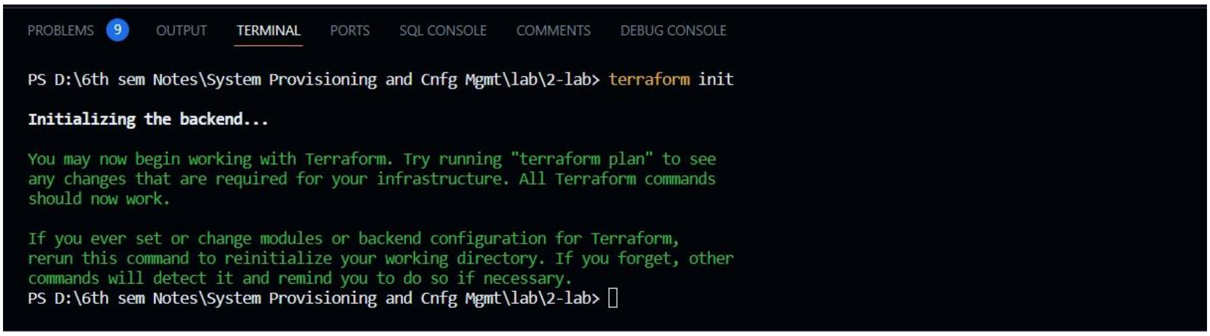


```
terraform{
    required_providers{
        aws={
            source ="harshicrop/aws",
            version ="5.31.0
        }
    }
}
provider "aws"{
    region ="ap-south-1"
    access_key_id="AKIA56IIZWYDXD2H5UJV"
    secret_key="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"
}
```

This script defines an AWS provider and provisions an EC2 instance.

Step 3: Initialize Terraform:

Run the following command to initialize your Terraform working directory:



```
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\2-lab> terraform init

Initializing the backend...

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

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\2-lab> 
```

Lab Exercise 3–Provisioning an EC2 Instance on AWS

Prerequisites: Terraform Installed: Make sure you have Terraform installed on your machine.

Follow the official installation guide if needed.

AWS Credentials: Ensure you have AWS credentials (Access Key ID and Secret Access Key) configured. You can set them up using the AWS CLI or by setting environment variables.

Exercise Steps:

Step 1: Create a New Directory:

Create a new directory for your Terraform configuration:

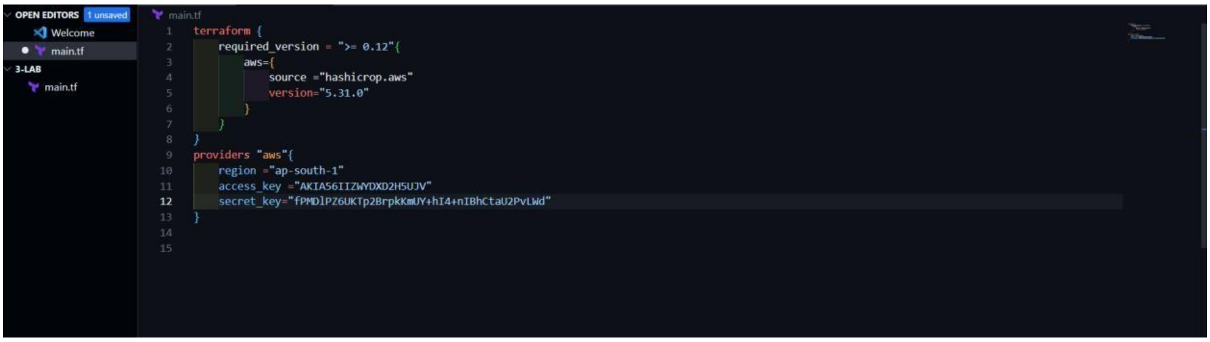
mkdir aws-terraform-demo

cd aws-terraform-demo

3-lab 30-01-2024 09:52 AM File folder

Step 2: Create Terraform Configuration File (main.tf):

Create a file named main.tf with the following content:



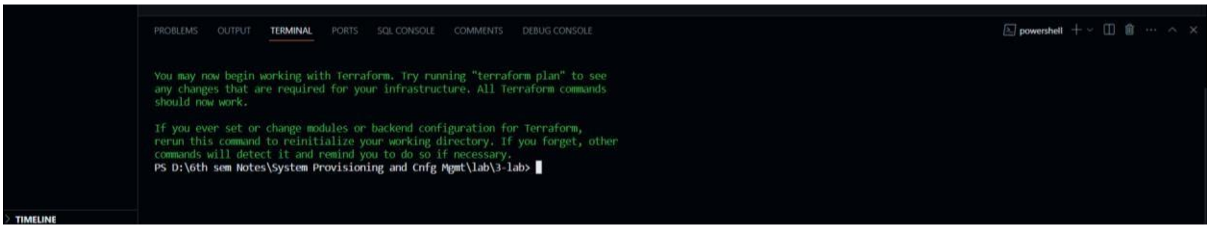
```
terraform {
    required_version = ">= 0.12"{
        aws={
            source ="hashicrop.aws"
            version="5.31.0"
        }
    }
}
providers "aws"{
    region ="ap-south-1"
    access_key ="AKIA56IIZWYDXD2H5UJV"
    secret_key="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"
}
```

This script defines an AWS provider and provisions an EC2 instance.

Step 3: Initialize Terraform:

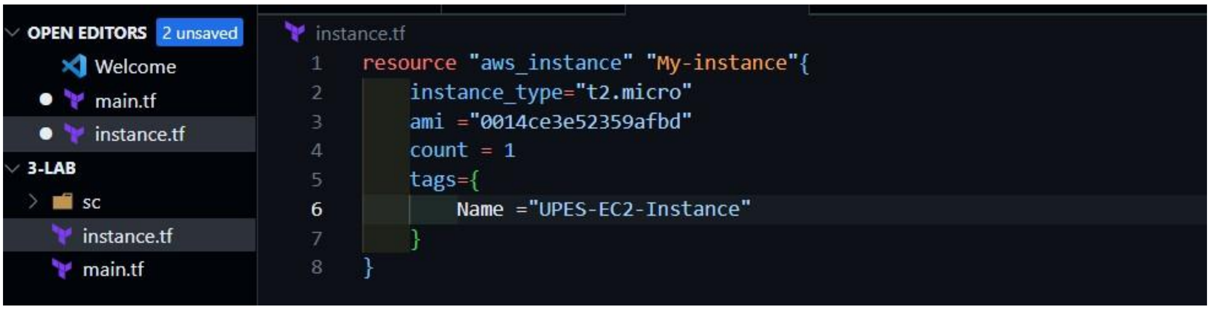
Run the following command to initialize your Terraform working directory:

terraform init



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

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\3-lab>
```

Step 4: Create Terraform Configuration File for EC2 instance (instance.tf):

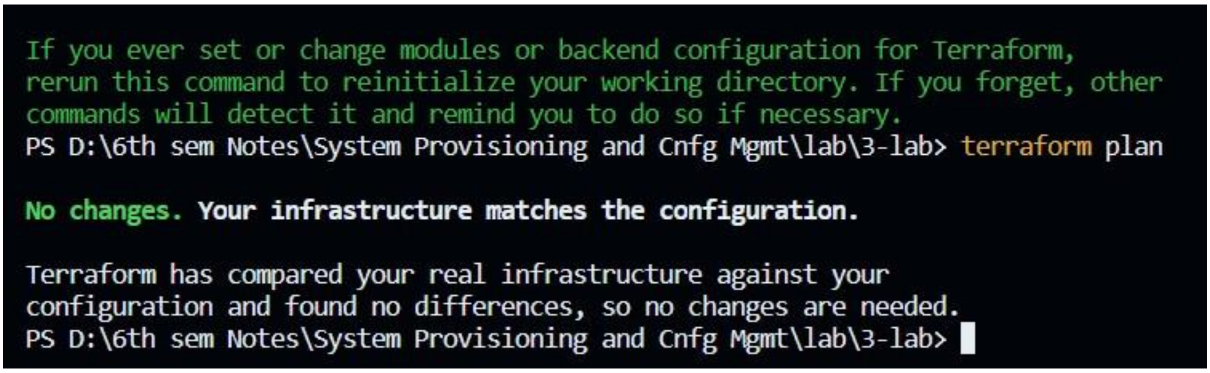


```
resource "aws_instance" "My-instance"{
    instance_type="t2.micro"
    ami ="0014ce3e52359afbd"
    count = 1
    tags={
        Name ="UPES-EC2-Instance"
    }
}
```

Step 5: Review Plan:

Run the following command to see what Terraform will do: terraform

plan

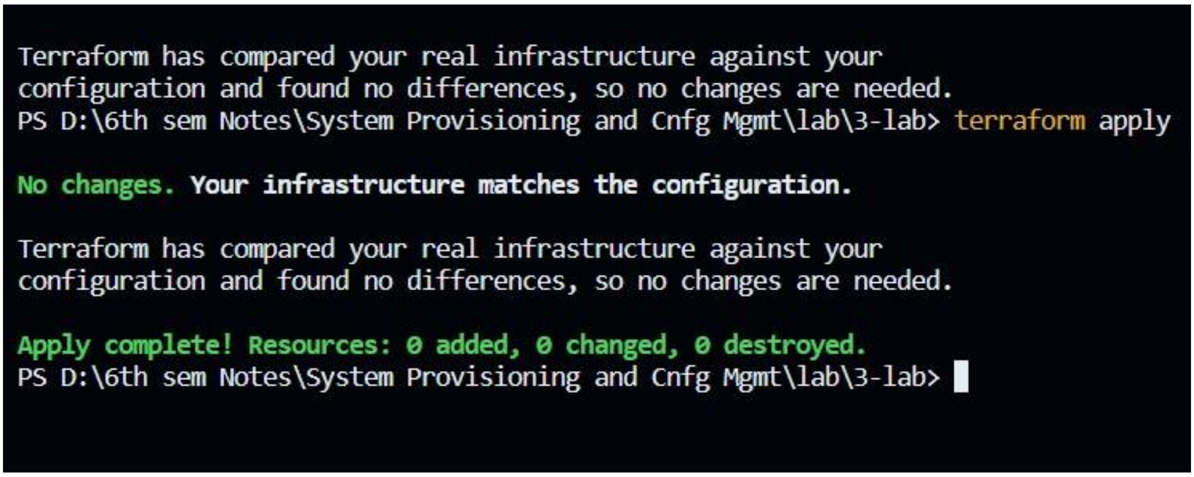


```
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\3-lab> terraform plan

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your
configuration and found no differences, so no changes are needed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\3-lab>
```

Step 6: Apply Changes:

Apply the changes to create the AWS resources:

terraform apply

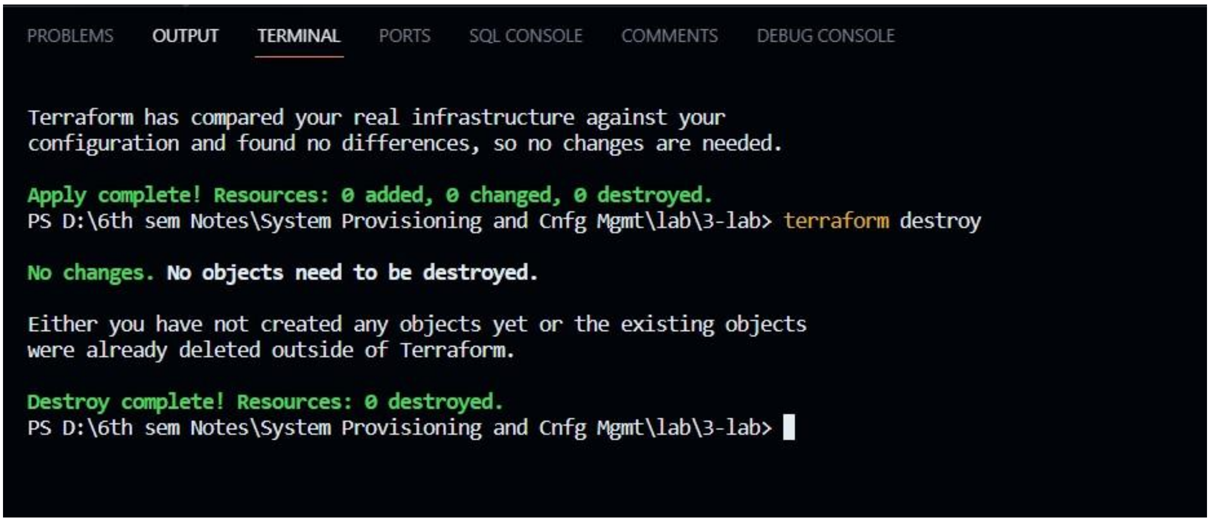


```
Terraform has compared your real infrastructure against your
configuration and found no differences, so no changes are needed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\3-lab> terraform apply

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your
configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\3-lab>
```

Step 7: Verify Resources:

After the terraform apply command completes, log in to your AWS Management Console and navigate to the EC2 dashboard. Verify that the EC2 instance has been created. Step 8: Cleanup Resources:

When you are done experimenting, run the following command to destroy the created: terraform destroy



```
Terraform has compared your real infrastructure against your
configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\3-lab> terraform destroy

No changes. No objects need to be destroyed.

Either you have not created any objects yet or the existing objects
were already deleted outside of Terraform.

Destroy complete! Resources: 0 destroyed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\3-lab>
```

Lab Exercise 4– Terraform Variables

Objective: Learn how to define and use variables in Terraform configuration.

Prerequisites: • Install Terraform on your machine.

Steps:

1. Create a Terraform Directory:

• Create a new directory for your Terraform project. mkdir

terraform-variables

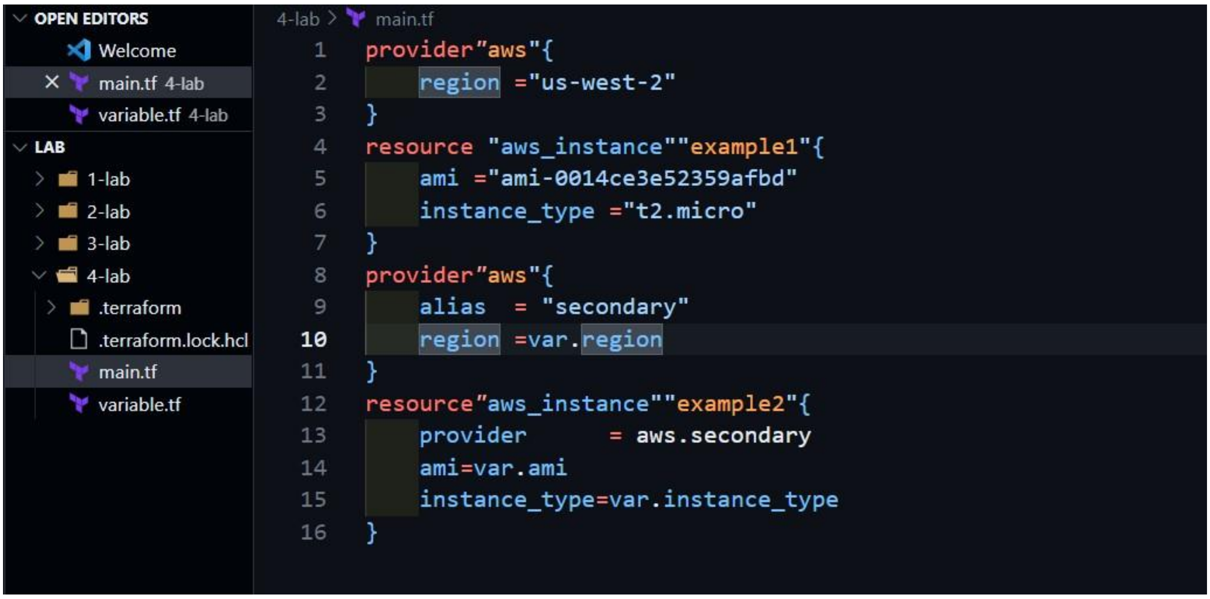
cd terraform-variables

4-lab   06-02-2024 09:48 AM   File folder

2. Create a Terraform Configuration File:

• Create a file named main.tf within your project directory. #

main.tf

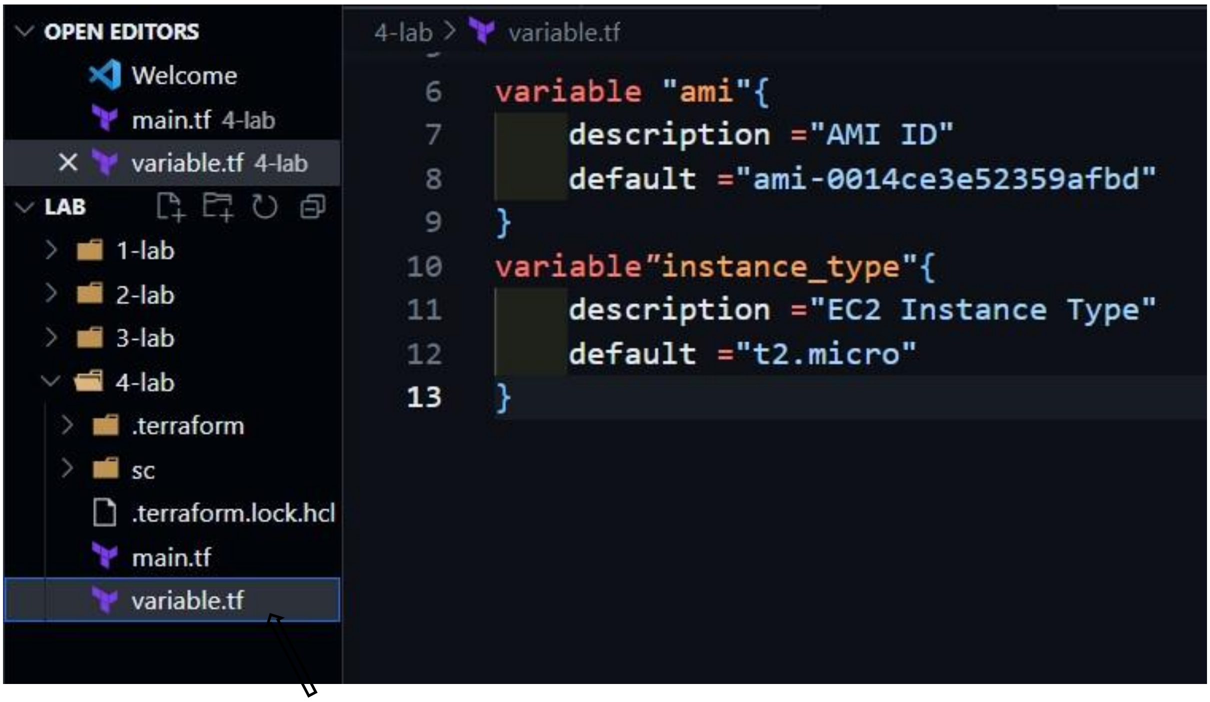


```
provider"aws"{
    region ="us-west-2"
}
resource "aws_instance""example1"{
    ami ="ami-0014ce3e52359afbd"
    instance_type ="t2.micro"
}
provider"aws"{
    alias  = "secondary"
    region =var.region
}
resource"aws_instance""example2"{
    provider      = aws.secondary
    ami=var.ami
    instance_type=var.instance_type
}
```

3. Define Variables:

• Open a new file named variables.tf. Define variables for region, ami, and instance_type.

# variables.tf



```
variable "ami"{
    description ="AMI ID"
    default ="ami-0014ce3e52359afbd"
}
variable"instance_type"{
    description ="EC2 Instance Type"
    default ="t2.micro"
}
```

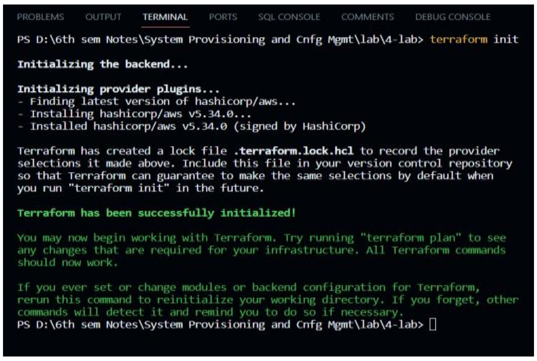
4. Use Variables in main.tf:

• Modify main.tf to use the variables.

# main.tf

5. Initialize and Apply:

• Run the following Terraform commands to initialize and apply the configuration. terraform init terraform apply



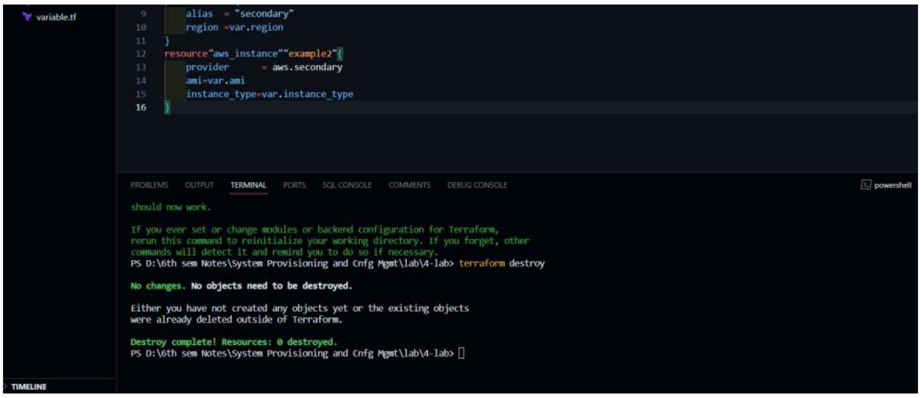
```
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\4-lab> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.34.0...
- Installed hashicorp/aws v5.34.0 (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!

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

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\4-lab> 
```

Observe how the region changes based on the variable override.

6. Clean Up:

After testing, you can clean up resources. terraform

destroy



```
    alias  = "secondary"
    region =var.region
}
resource"aws_instance""example2"{
    provider      = aws.secondary
    ami=var.ami
    instance_type=var.instance_type
}
```

```
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\4-lab> terraform destroy

No changes. No objects need to be destroyed.

Either you have not created any objects yet or the existing objects
were already deleted outside of Terraform.

Destroy complete! Resources: 0 destroyed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\4-lab> 
```

Confirm the destruction by typing yes.

7. Conclusion:

This lab exercise introduces you to Terraform variables and demonstrates how to use

them in your configurations. Experiment with different variable values and

overrides to understand their impact on the infrastructure provisioning

process.

Lab Exercise 5– Terraform Variables with Command Line Arguments

Objective: Learn how to pass values to Terraform variables using command line arguments.

Prerequisites: • Terraform installed on your machine.

• Basic knowledge of Terraform variables.

Steps:

1. Create a Terraform Directory: mkdir

terraform-cli-variables

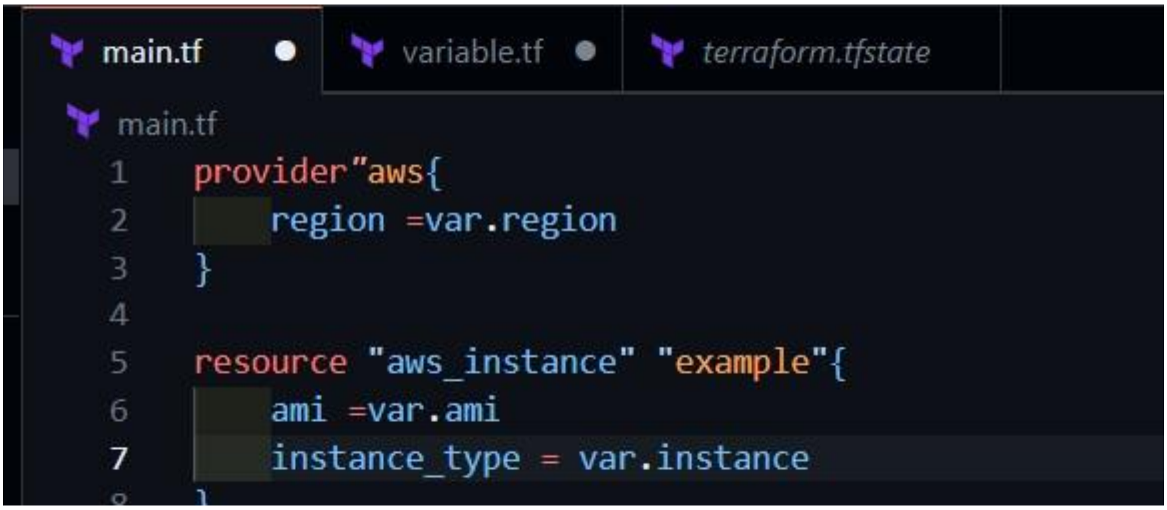
cd terraform-cli-variables

```
5-lab                11-02-2024 01:11 PM      File folder
```

2. Create Terraform Configuration Files:

• Create a file named main.tf:

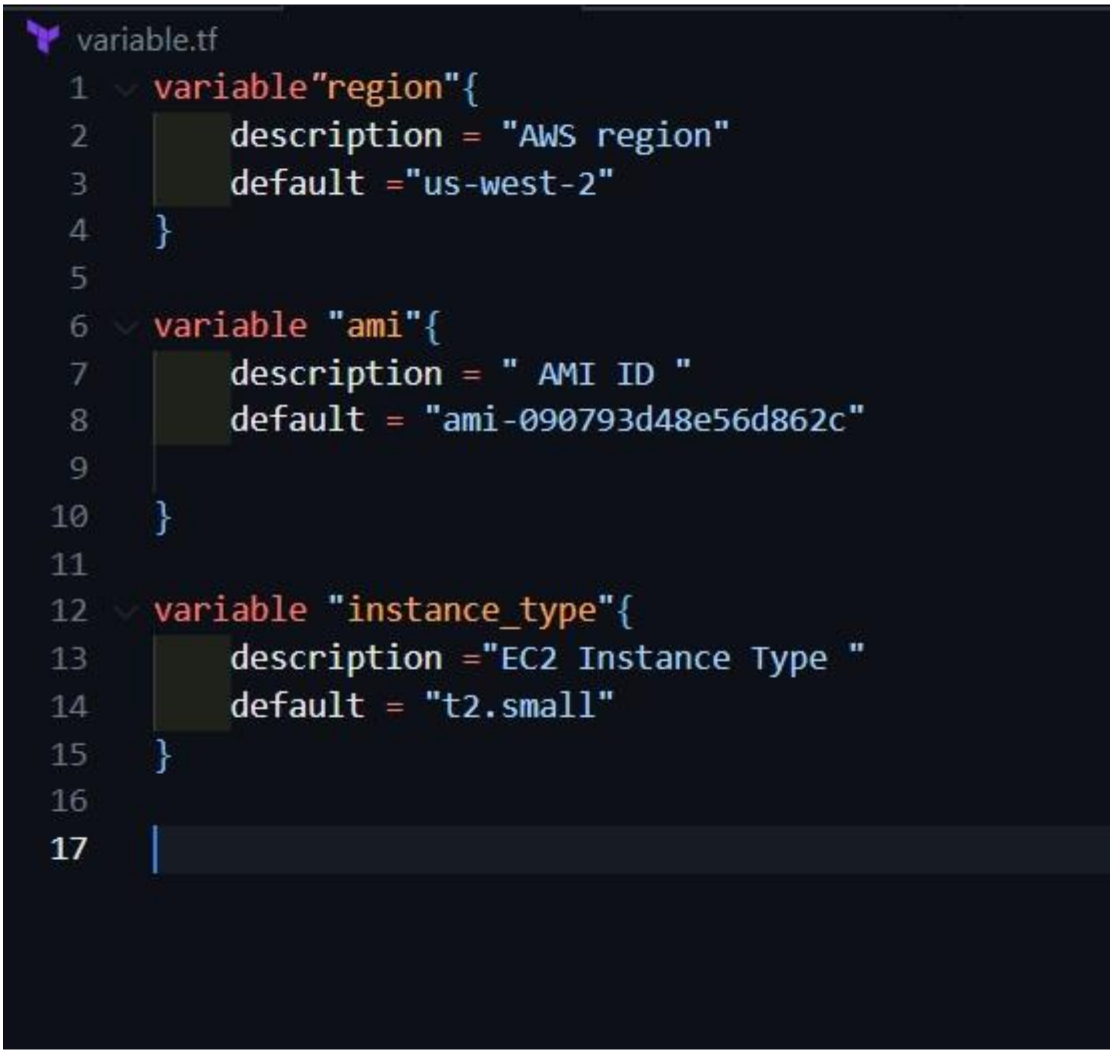
# main.tf



```
provider”aws{
    region =var.region
}

resource "aws_instance" "example"{
    ami =var.ami
    instance_type = var.instance
}
```

- Create a file named variables.tf:

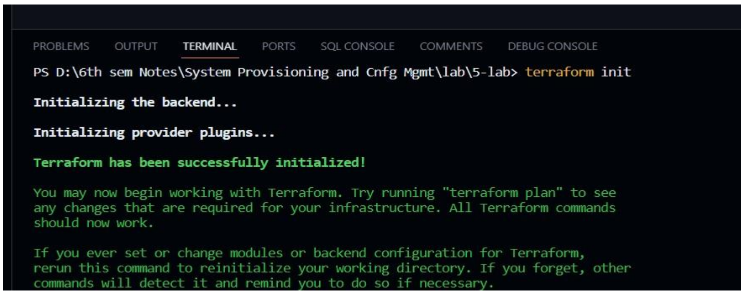
# variables.tf



```
variable"region"{
        description = "AWS region"
        default ="us-west-2"
}

variable "ami"{
        description = " AMI ID "
        default = "ami-090793d48e56d862c"

}

variable "instance_type"{
        description ="EC2 Instance Type "
        default = "t2.small"
}

```

3.

- Open a terminal and navigate to your Terraform project directory.
- Run the terraform init command: terraform init



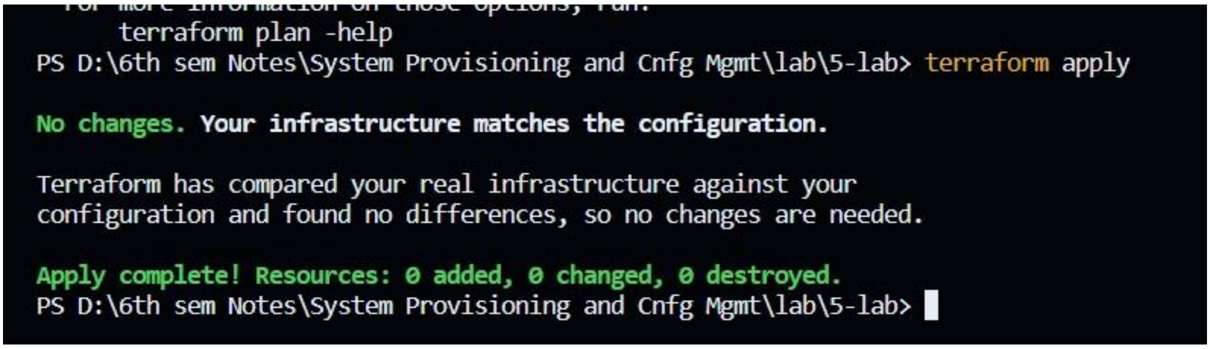
```
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\5-lab> terraform init

Initializing the backend...

Initializing provider plugins...

Terraform has been successfully initialized!

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

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

- Run the terraform apply command with command line arguments to set variable values:

terraform apply -var 'region=us-east-1' -var 'ami=ami-12345678' -var

'instance_type=t2.micro

- Adjust the values based on your preferences.4.Test and Verify:
- Observe how the command line arguments dynamically set the variable values during the apply process.
- Access the AWS Management Console or use the AWS CLI to verify the creation of resources in the specified region.

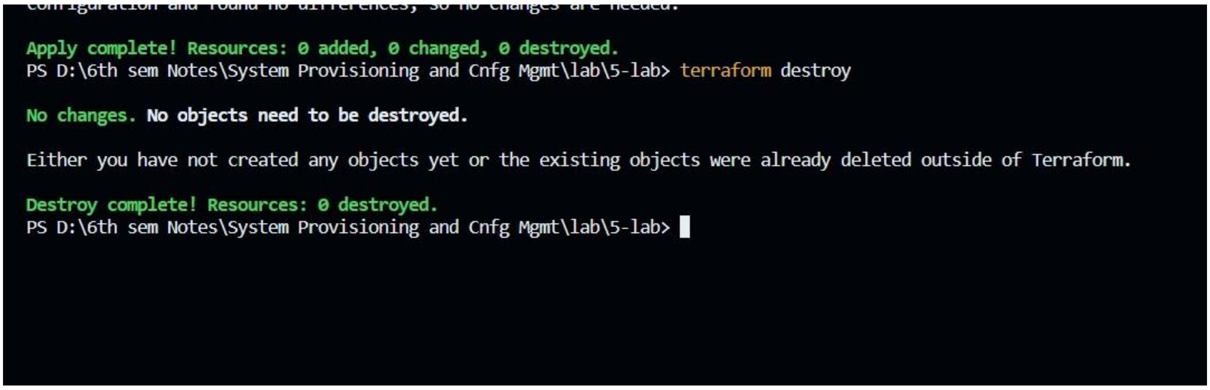


```
   For more information on those options, run:
      terraform plan -help
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\5-lab> terraform apply

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your
configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\5-lab> 
```

5. Clean Up:

After testing, you can clean up resources: terraform

destroy



```
configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\5-lab> terraform destroy

No changes. No objects need to be destroyed.

Either you have not created any objects yet or the existing objects were already deleted outside of Terraform.

Destroy complete! Resources: 0 destroyed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\5-lab> 
```

6. Conclusion:

This lab exercise demonstrates how to use command line arguments to set variablevalues dynamically during the terraform apply process. It allows you to customize your Terraform deployments without modifying the configuration files directly.

Experiment with different variable values and observe how command line arguments impact the infrastructure provisioning process.

Lab Exercise 6– Terraform Multiple tfvars Files

Objective:

Learn how to use multiple tfvars files in Terraform for different environments.

Prerequisites:

- **Terraform installed on your machine.**
- **Basic knowledge of Terraform configuration and variables.**

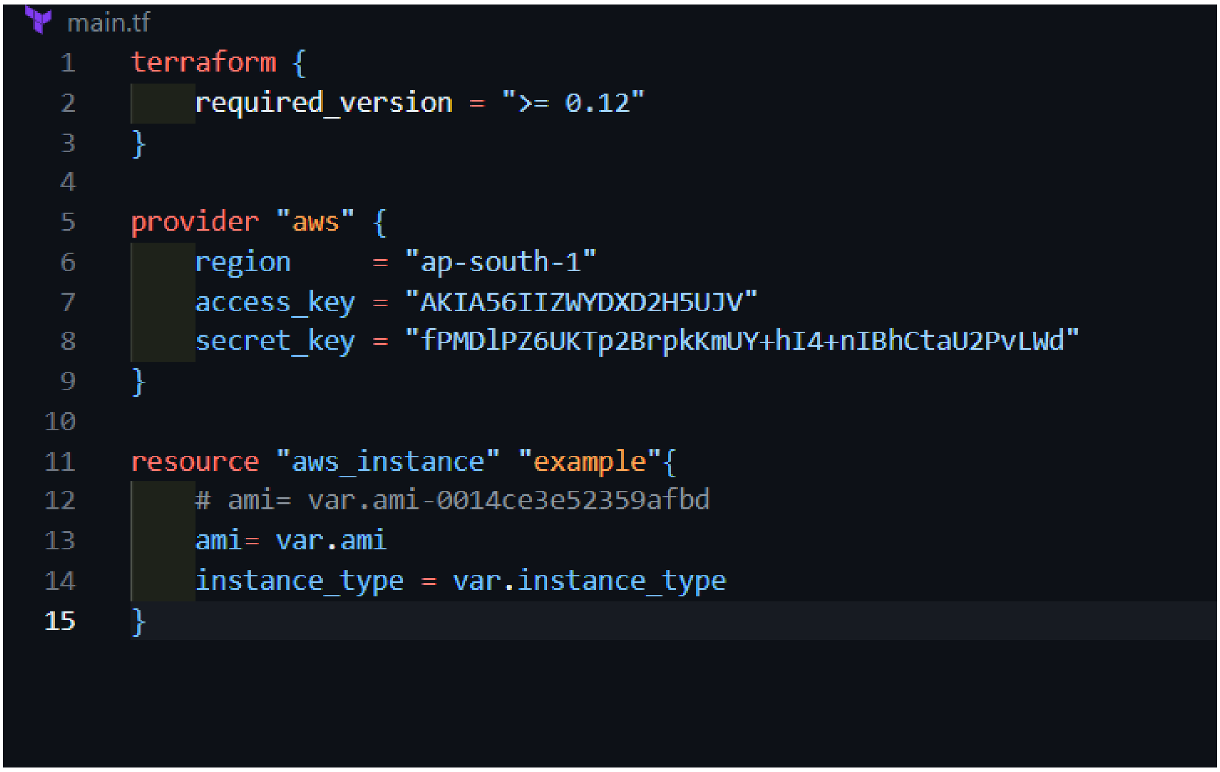
Steps:

1. Create a Terraform Directory:

```
lab-6                    26-02-2024 12:56 AM        File folder
```

- Create Terraform Configuration Files:
- **Create a file named main.tf:**

# main.tf

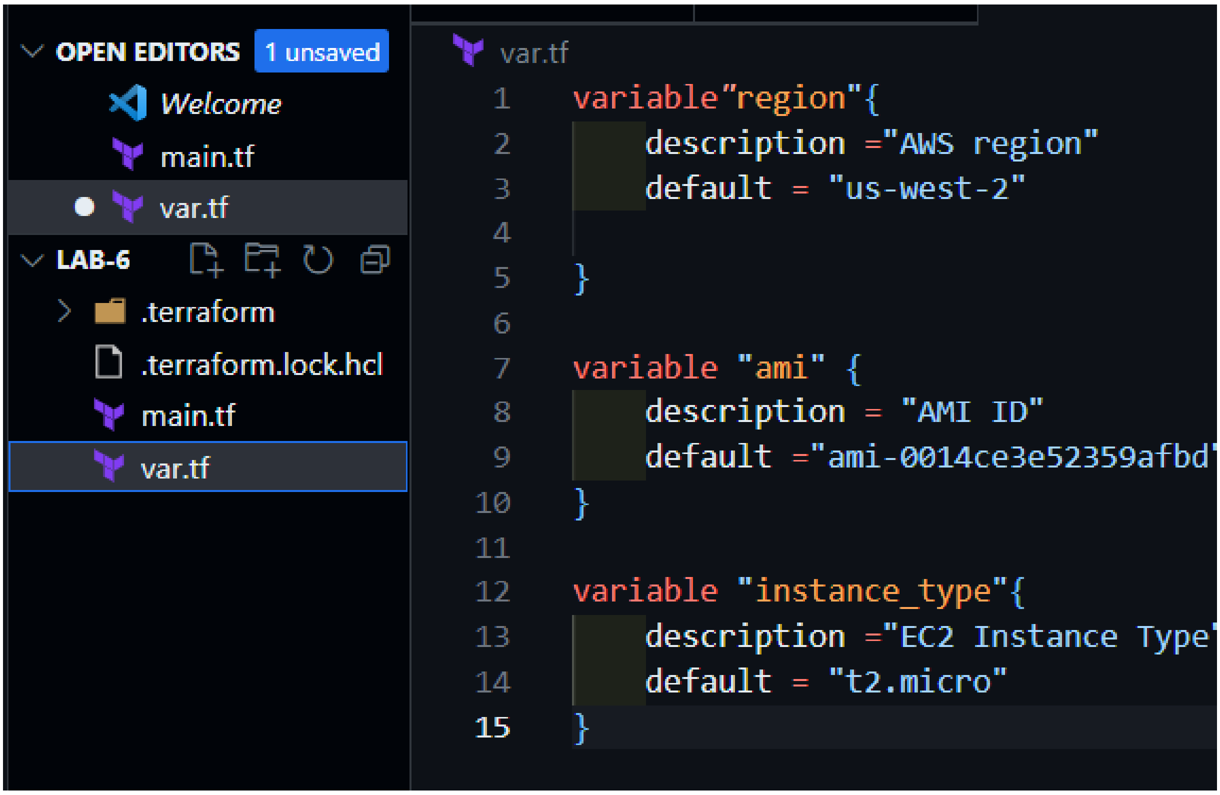


main.tf

```
terraform {
    required_version = ">= 0.12"
}

provider "aws" {
    region     = "ap-south-1"
    access_key = "AKIA56IIZWYDXD2H5UJV"
    secret_key = "fPMDlPZ6UKTp2BrpkKmUY+hI4+nIBhCtaU2PvLWd"
}

resource "aws_instance" "example"{
    # ami= var.ami-0014ce3e52359afbd
    ami= var.ami
    instance_type = var.instance_type
}
```

• **Create a file named variables.tf**:

# variables.tf

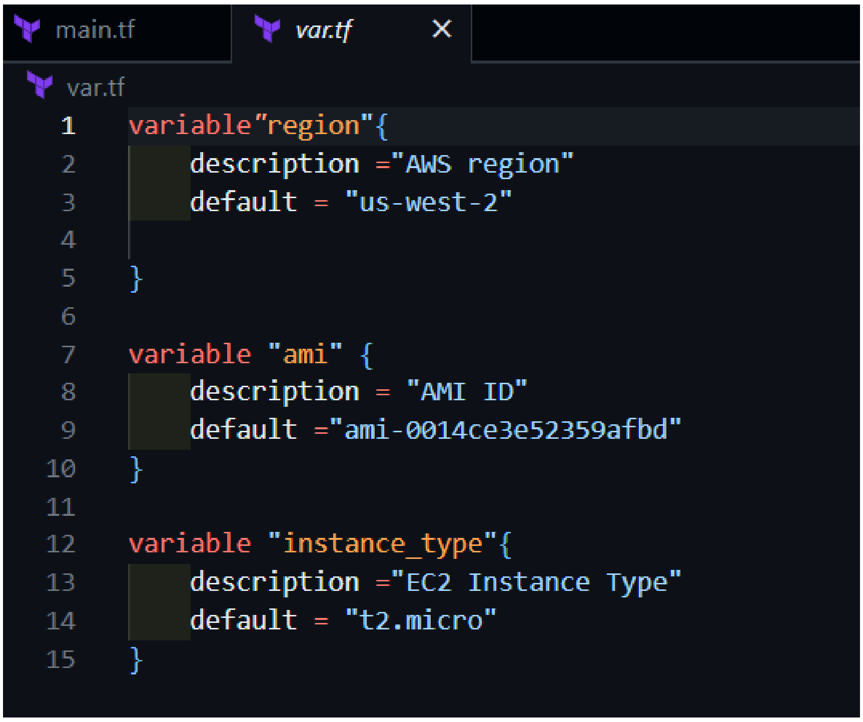


```
variable”region"{
    description ="AWS region"
    default = "us-west-2"

}

variable "ami" {
    description = "AMI ID"
    default ="ami-0014ce3e52359afbd"
}

variable "instance_type"{
    description ="EC2 Instance Type"
    default = "t2.micro"
}
```

2. Create Multiple tfvars Files:

•Create a file named dev.tfvars:

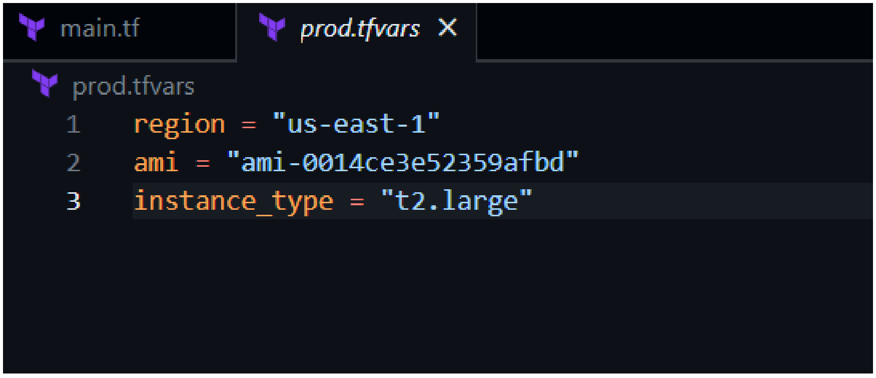
# dev.tfvars



```
variable”region"{
    description ="AWS region"
    default = "us-west-2"

}

variable "ami" {
    description = "AMI ID"
    default ="ami-0014ce3e52359afbd"
}

variable "instance_type"{
    description ="EC2 Instance Type"
    default = "t2.micro"
}
```

•Create a file named prod.tfvars:

# prod.tfvars

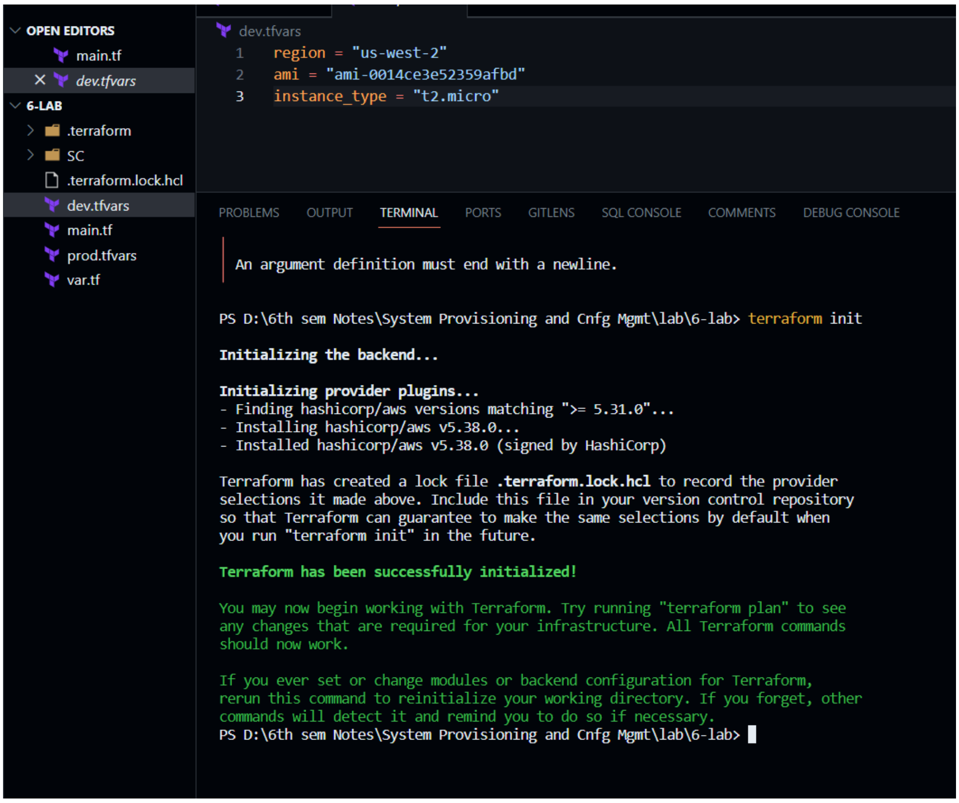


```
prod.tfvars
region = "us-east-1"
ami = "ami-0014ce3e52359afbd"
instance_type = "t2.large"
```

•In these files, provide values for the variables based on the environments.

3. Initialize and Apply for Dev Environment:

•Run the following Terraform commands to initialize and apply the configuration
for the dev environment:



```
dev.tfvars
region = "us-west-2"
ami = "ami-0014ce3e52359afbd"
instance_type = "t2.micro"
```

```
An argument definition must end with a newline.

PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\6-lab> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching ">= 5.31.0"...
- Installing hashicorp/aws v5.38.0...
- Installed hashicorp/aws v5.38.0 (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!

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

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\6-lab> ▌
```

4. Initialize and Apply for Prod Environment:

•Run the following Terraform commands to initialize and apply the configuration
for the prod environment:

5. Test and Verify:

- Observe how different tfvars files are used to set variable values for different environments during the apply process.
- Access the AWS Management Console or use the AWS CLI to verify the creation of resources in the specified regions and instance types.

6. Clean Up:

- After testing, you can clean up resources:

```
terraform destroy -var-file=dev.tfvars
terraform destroy -var-file=prod.tfvars
```

- Confirm the destruction by typing yes.

Lab Exercise 7– Creating Multiple IAM Users inTerraform

Objective:

Learn how to use Terraform to create multiple IAM users with unique settings.

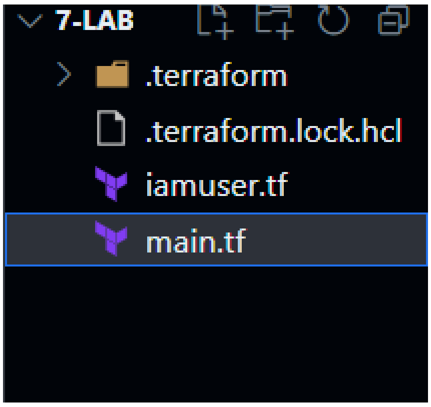
Prerequisites:

• Terraform installed on your machine.

• AWS CLI configured with the necessary credentials.

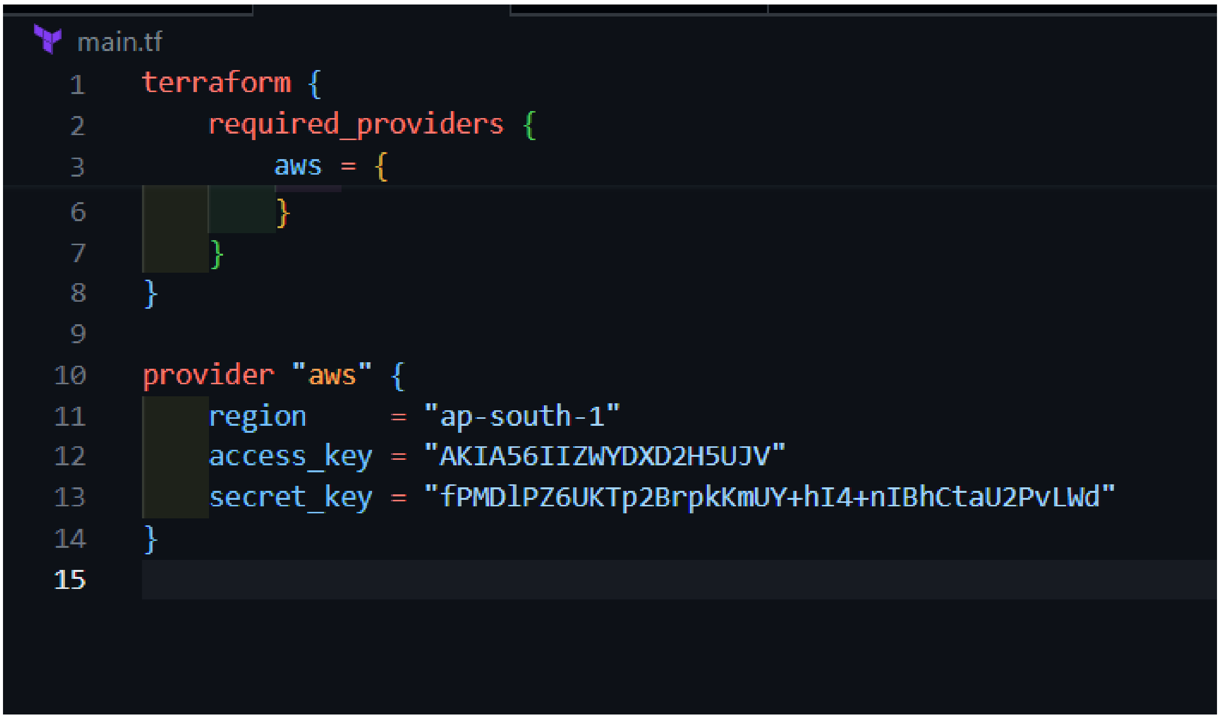
Steps:

1. Create a Terraform Directory:



```
7-LAB
  .terraform
  .terraform.lock.hcl
  iamuser.tf
  main.tf
```

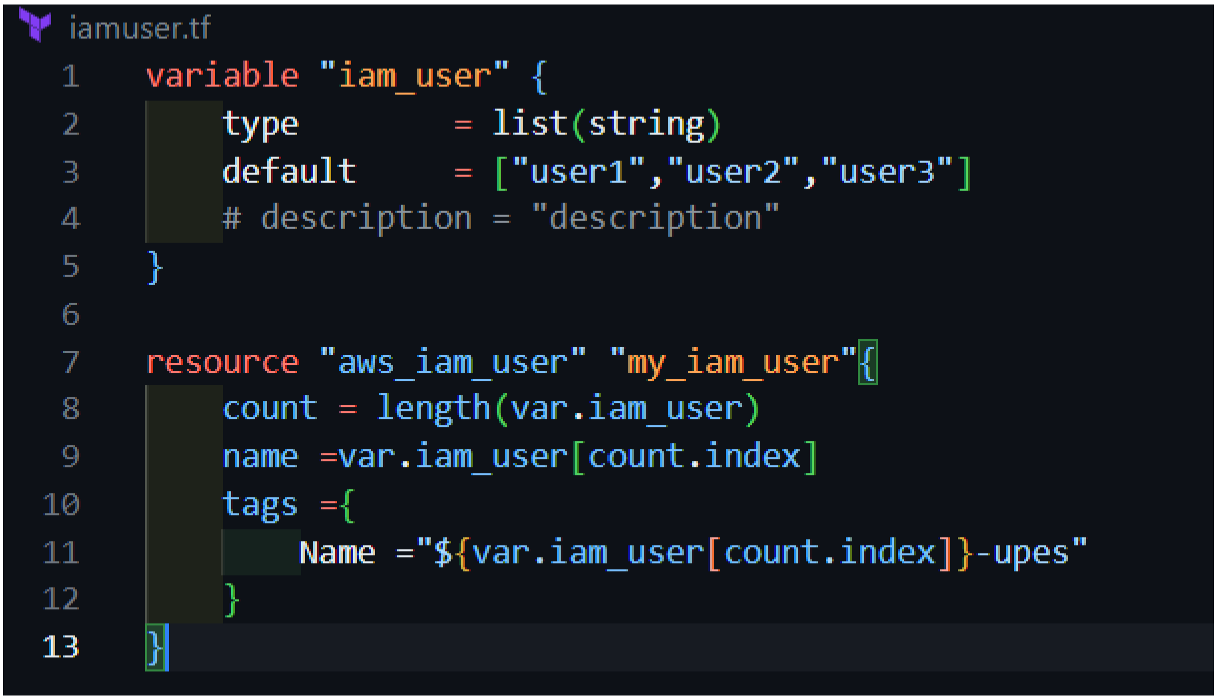
# main.tf



main.tf

```
terraform {
    required_providers {
        aws = {
        }
    }
}

provider "aws" {
    region     = "ap-south-1"
    access_key = "AKIA56IIZWYDXD2H5UJV"
    secret_key = "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"
}
```

#iamuser.tf

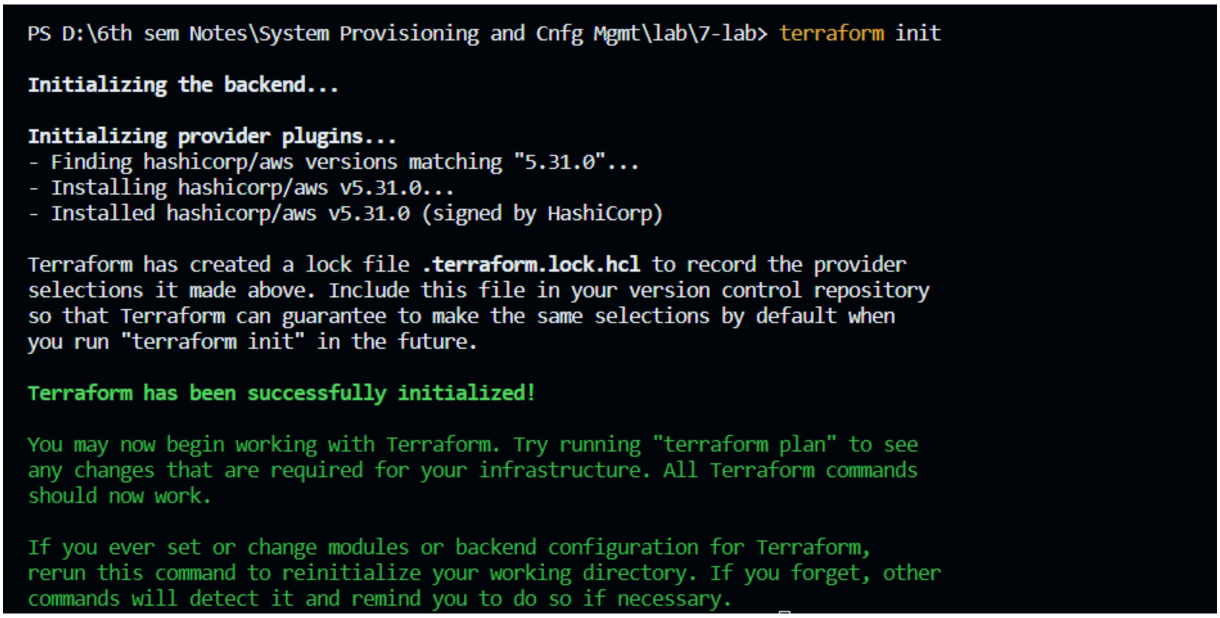


```
variable "iam_user" {
    type        = list(string)
    default     = ["user1","user2","user3"]
    # description = "description"
}

resource "aws_iam_user" "my_iam_user"{
    count = length(var.iam_user)
    name =var.iam_user[count.index]
    tags ={
        Name ="${var.iam_user[count.index]}-upes"
    }
}
```

2. Initialize and Apply:

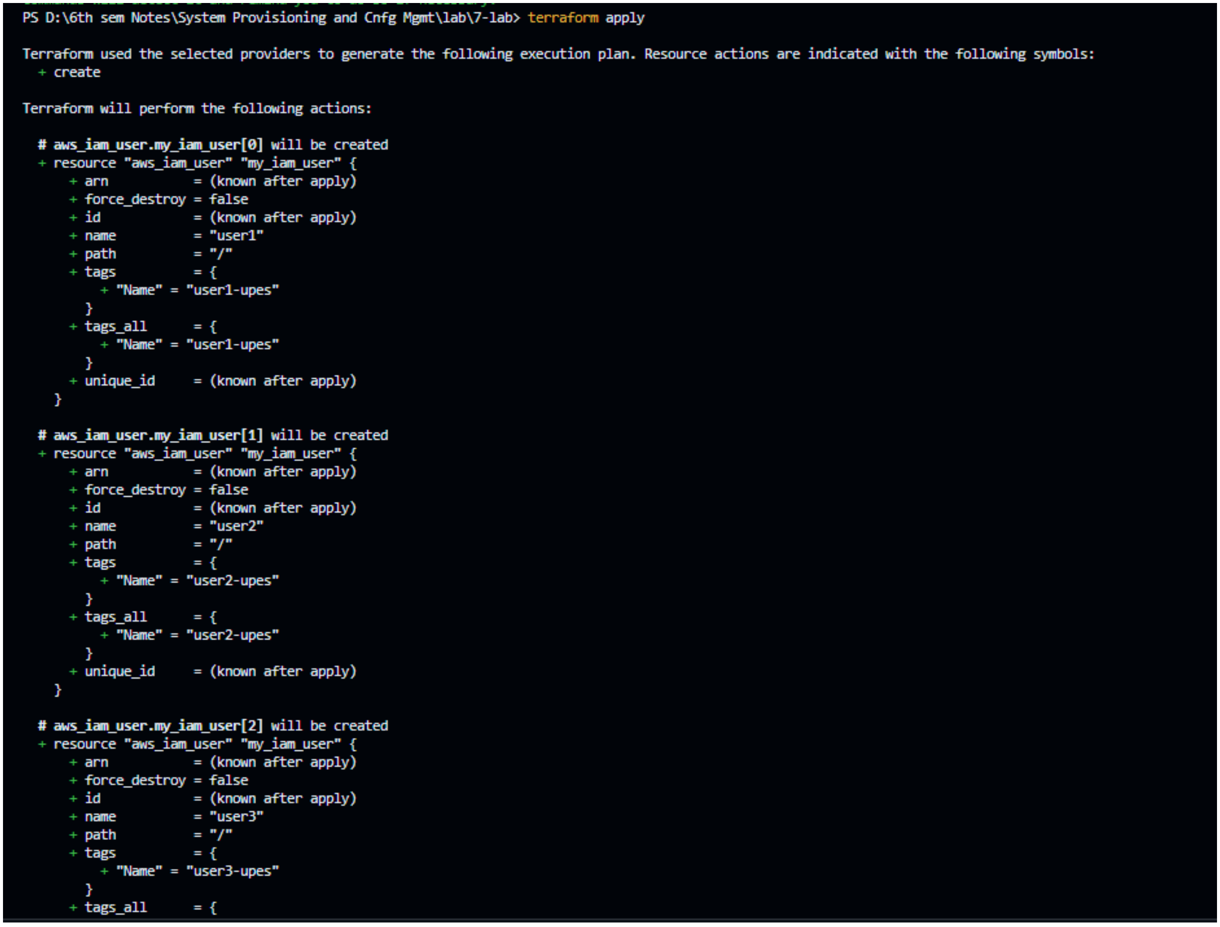
Run the following Terraform commands to initialize and apply the configuration:

terraform init

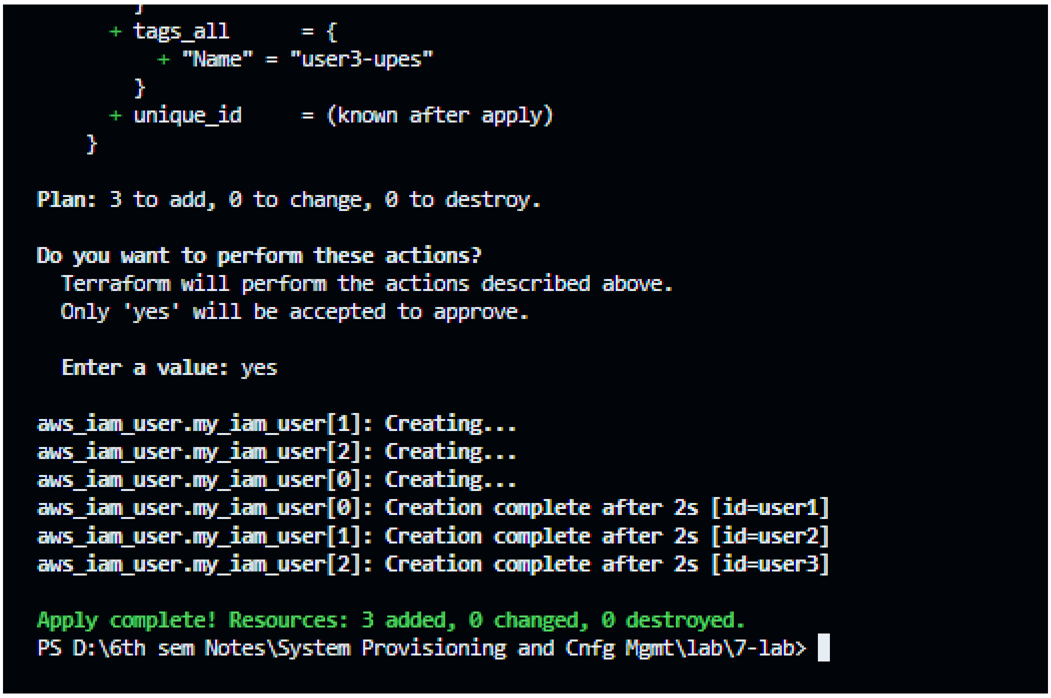


```
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\7-lab> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.31.0"...
- Installing hashicorp/aws v5.31.0...
- Installed hashicorp/aws v5.31.0 (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!

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

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

terraform apply



```
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\7-lab> terraform apply

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_iam_user.my_iam_user[0] will be created
  + resource "aws_iam_user" "my_iam_user" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "user1"
      + path          = "/"
      + tags          = {
          + "Name" = "user1-upes"
        }
      + tags_all      = {
          + "Name" = "user1-upes"
        }
      + unique_id     = (known after apply)
    }

  # aws_iam_user.my_iam_user[1] will be created
  + resource "aws_iam_user" "my_iam_user" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "user2"
      + path          = "/"
      + tags          = {
          + "Name" = "user2-upes"
        }
      + tags_all      = {
          + "Name" = "user2-upes"
        }
      + unique_id     = (known after apply)
    }

  # aws_iam_user.my_iam_user[2] will be created
  + resource "aws_iam_user" "my_iam_user" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "user3"
      + path          = "/"
      + tags          = {
          + "Name" = "user3-upes"
        }
      + tags_all      = {
```

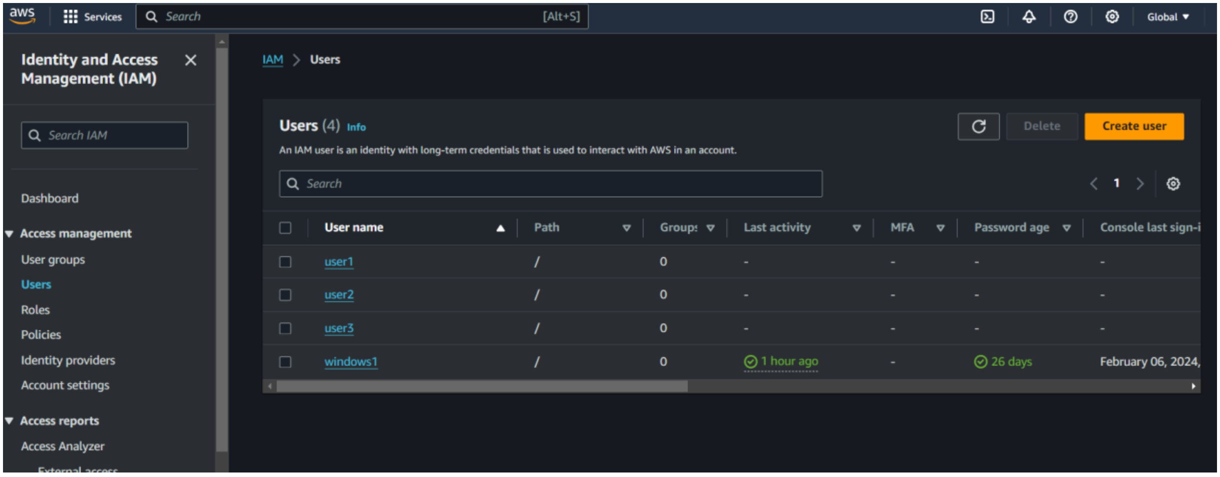


```
      + tags_all      = {
          + "Name" = "user3-upes"
        }
      + unique_id     = (known after apply)
    }

Plan: 3 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_iam_user.my_iam_user[1]: Creating...
aws_iam_user.my_iam_user[2]: Creating...
aws_iam_user.my_iam_user[0]: Creating...
aws_iam_user.my_iam_user[0]: Creation complete after 2s [id=user1]
aws_iam_user.my_iam_user[1]: Creation complete after 2s [id=user2]
aws_iam_user.my_iam_user[2]: Creation complete after 2s [id=user3]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\7-lab>
```

Terraform will prompt you to confirm the creation of IAM users. Type yes and press
Enter.

3. Verify Users in AWS Console:

- Log in to the AWS Management Console and navigate to the IAM service.
- Verify that the IAM users with the specified names and tags have been created.



Users (4) Info

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

User name	Path	Groups	Last activity	MFA	Password age	Console last sign-i
user1	/	0	-	-	-	-
user2	/	0	-	-	-	-
user3	/	0	-	-	-	-
windows1	/	0	✓ 1 hour ago	-	✓ 26 days	February 06, 2024,

4. Update IAM Users:

- If you want to add or remove IAM users, modify the iam_users list in the main.tf file.
- Rerun the terraform apply command to apply the changes:

```
terraform apply
```

5. Clean Up:

- After testing, you can clean up the IAM users:

```
terraform destroy
```

- Confirm the destruction by typing yes.

6. Conclusion:

This lab exercise demonstrates how to create multiple IAM users in AWS using Terraform. The use of variables and loops allows you to easily manage and scale the creation of IAM users. Experiment with different user names and settings in the main.tf file to understand how Terraform provisions resources based on your configuration

Lab Exercise 8– Creating a VPC in Terraform

Objective:

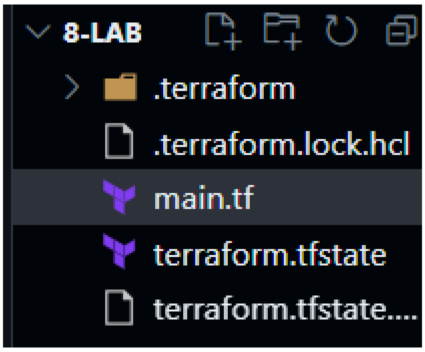
Learn how to use Terraform to create a basic Virtual Private Cloud (VPC) in AWS.

Prerequisites:

- **Terraform installed on your machine.**
- **AWS CLI configured with the necessary credentials.**

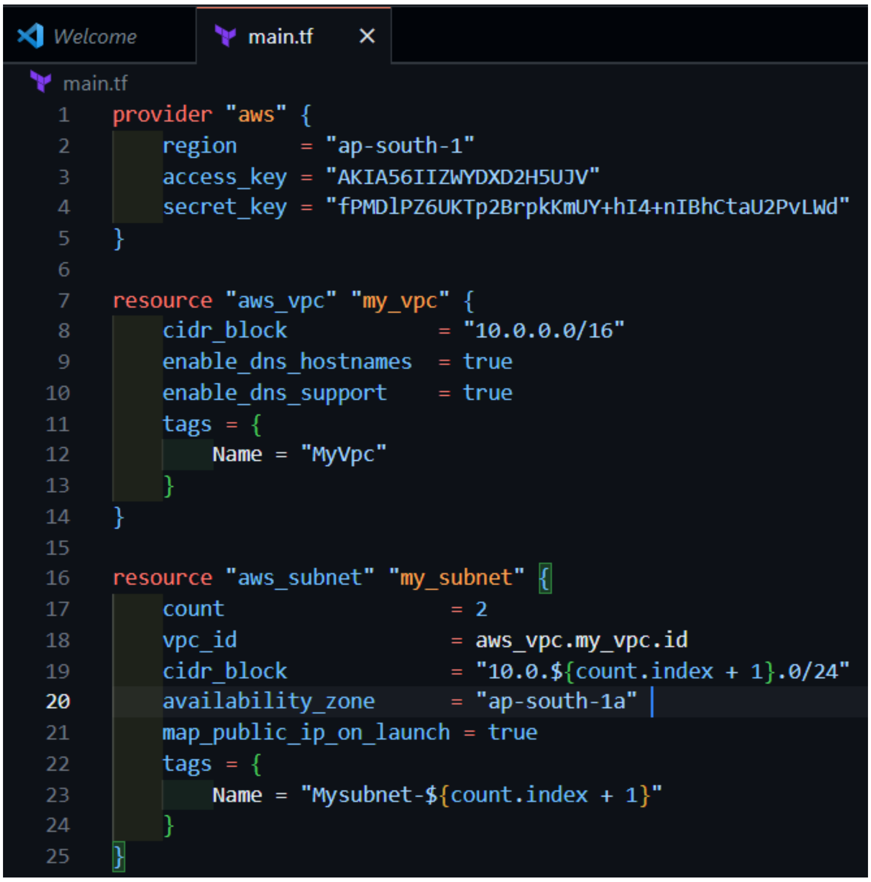
Steps:

1. Create a Terraform Directory:



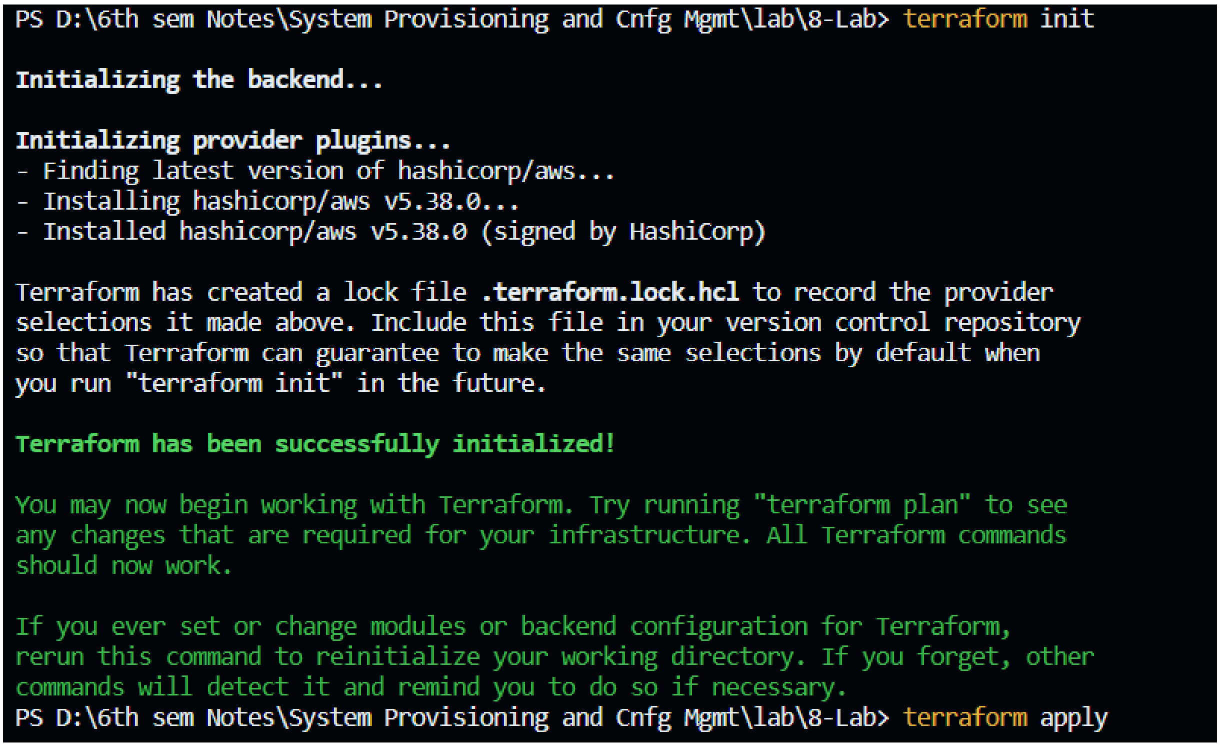
```
8-LAB
  .terraform
  .terraform.lock.hcl
  main.tf
  terraform.tfstate
  terraform.tfstate....
```

# main.tf



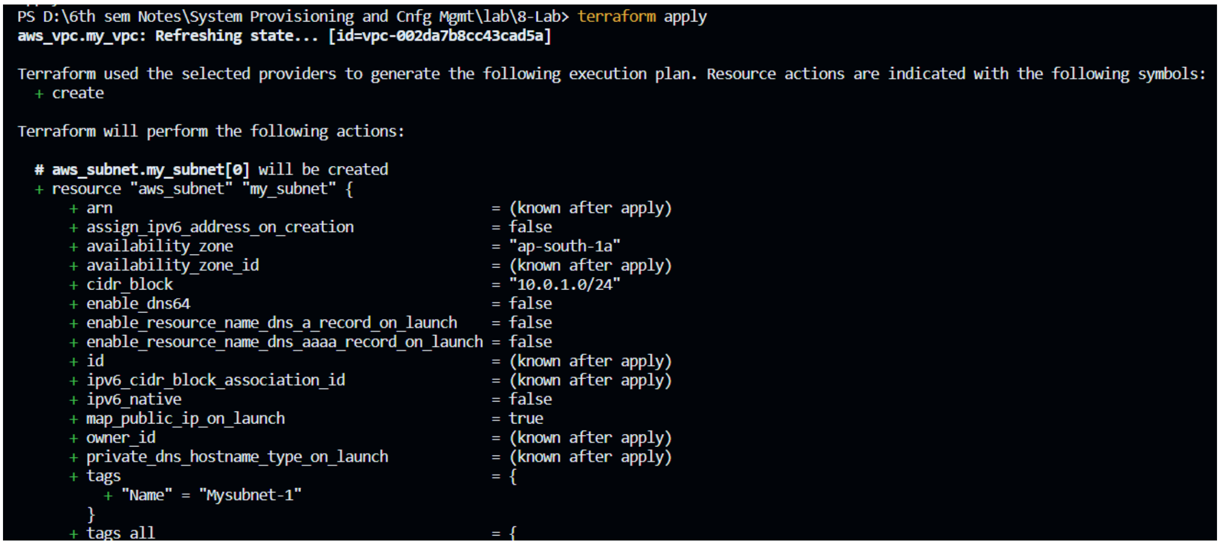
```
provider "aws" {
    region     = "ap-south-1"
    access_key = "AKIA56IIZWYDXD2H5UJV"
    secret_key = "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"
}

resource "aws_vpc" "my_vpc" {
    cidr_block           = "10.0.0.0/16"
    enable_dns_hostnames = true
    enable_dns_support   = true
    tags = {
        Name = "MyVpc"
    }
}

resource "aws_subnet" "my_subnet" {
    count                   = 2
    vpc_id                  = aws_vpc.my_vpc.id
    cidr_block              = "10.0.${count.index + 1}.0/24"
    availability_zone       = "ap-south-1a"
    map_public_ip_on_launch = true
    tags = {
        Name = "Mysubnet-${count.index + 1}"
    }
}
```

In this configuration, we define an AWS provider, a VPC with a specified CIDR block,
and two subnets within the VPC.
2. Initialize and Apply:
• Run the following Terraform commands to initialize and apply the configuration:
terraform init

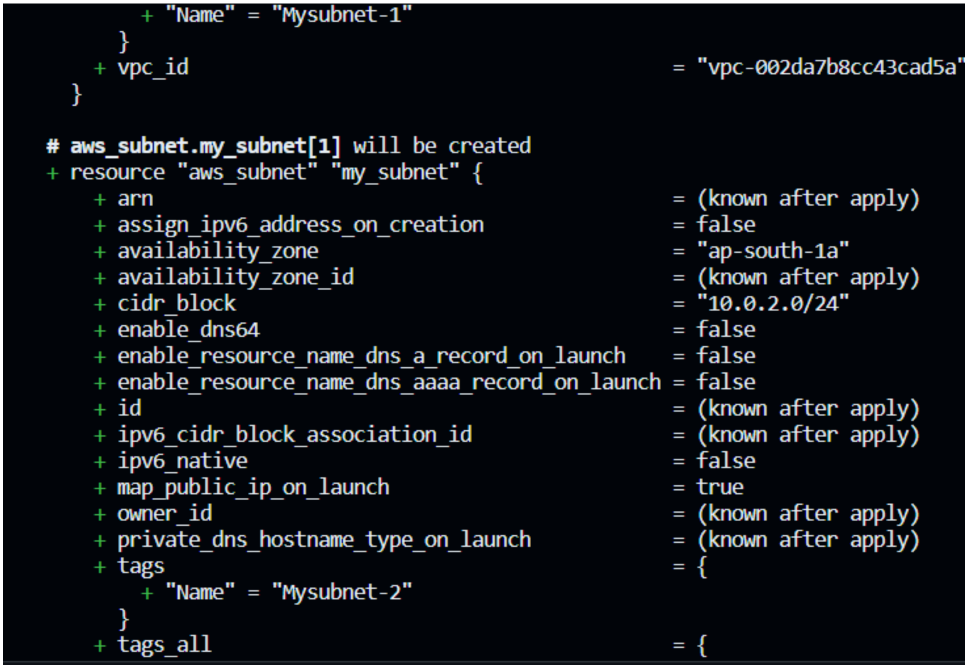


```
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\8-Lab> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.38.0...
- Installed hashicorp/aws v5.38.0 (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!

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

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\8-Lab> terraform apply
```

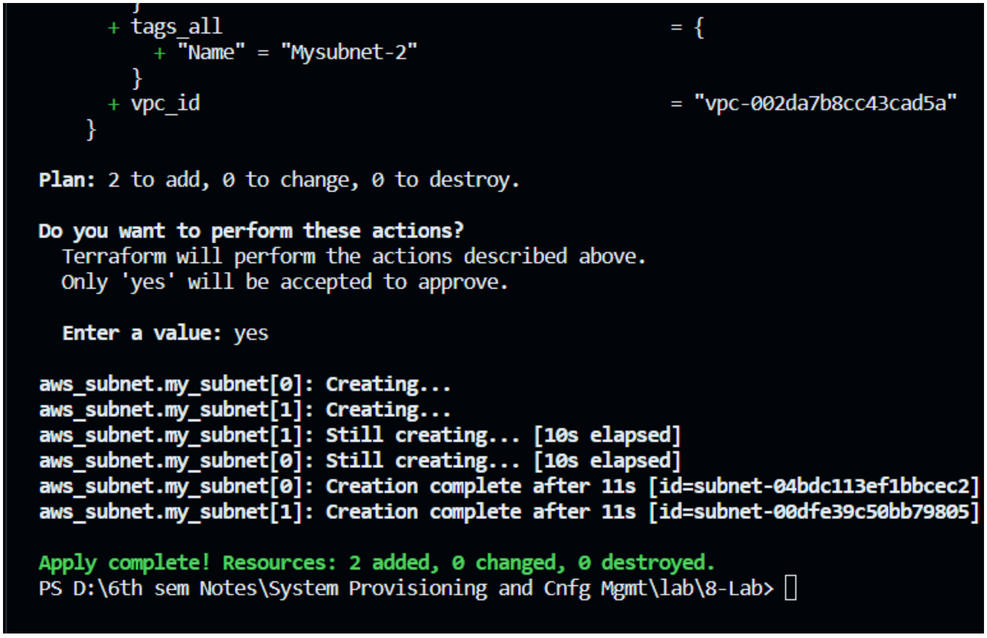
terraform apply
• Terraform will prompt you to confirm the creation of the VPC and subnets. Type yes and press Enter.:



```
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\8-Lab> terraform apply
aws_vpc.my_vpc: Refreshing state... [id=vpc-002da7b8cc43cad5a]

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_subnet.my_subnet[0] will be created
  + resource "aws_subnet" "my_subnet" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "ap-south-1a"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "10.0.1.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = true
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + tags                                           = {
          + "Name" = "Mysubnet-1"
        }
      + tags_all                                       = {
```



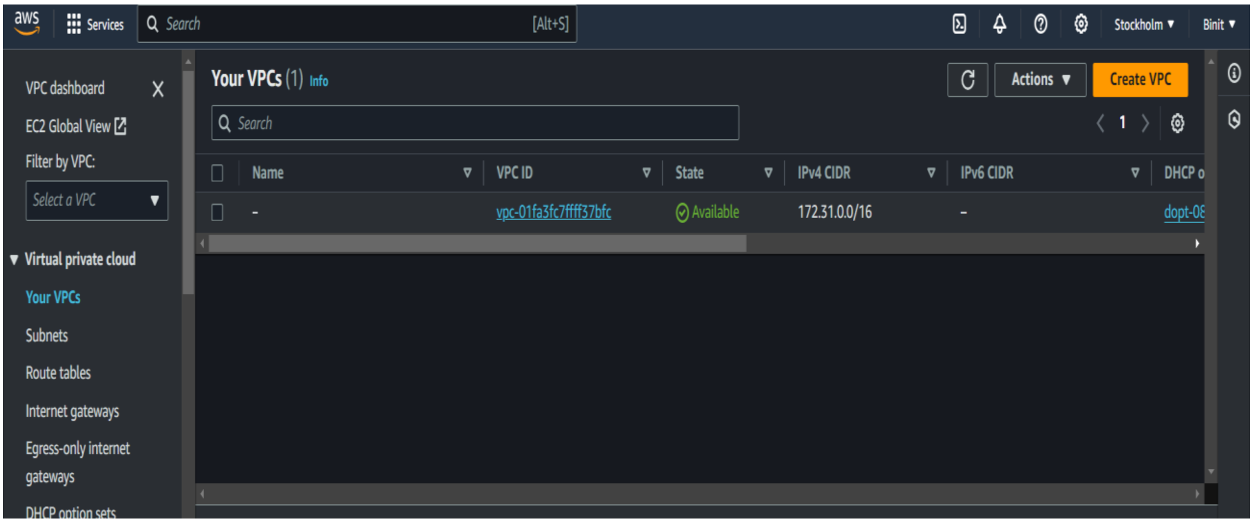
```
          + "Name" = "Mysubnet-1"
        }
      + vpc_id                                         = "vpc-002da7b8cc43cad5a"
    }

  # aws_subnet.my_subnet[1] will be created
  + resource "aws_subnet" "my_subnet" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "ap-south-1a"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "10.0.2.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = true
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + tags                                           = {
          + "Name" = "Mysubnet-2"
        }
      + tags_all                                       = {
```



```
      + tags_all                                       = {
          + "Name" = "Mysubnet-2"
        }
      + vpc_id                                         = "vpc-002da7b8cc43cad5a"
    }

Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_subnet.my_subnet[0]: Creating...
aws_subnet.my_subnet[1]: Creating...
aws_subnet.my_subnet[1]: Still creating... [10s elapsed]
aws_subnet.my_subnet[0]: Still creating... [10s elapsed]
aws_subnet.my_subnet[0]: Creation complete after 11s [id=subnet-04bdc113ef1bbcec2]
aws_subnet.my_subnet[1]: Creation complete after 11s [id=subnet-00dfe39c50bb79805]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\8-Lab> 
```

3. Verify Resources in AWS Console:

- Log in to the AWS Management Console and navigate to the VPC service.
- Verify that the VPC and subnets with the specified names and settings have been



Your VPCs (1) Info

□	Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR	DHCP o
□	-	vpc-01fa3fc7ffff37bfc	Available	172.31.0.0/16	-	dopt-08

4. Update VPC Configuration:

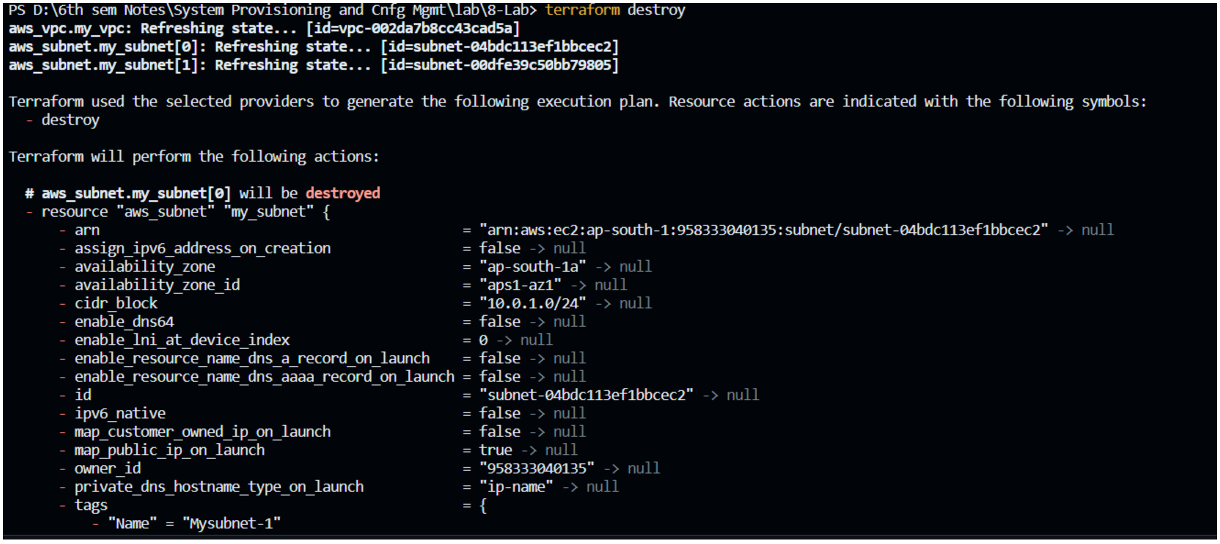
- If you want to modify the VPC configuration, update the main.tf file with the desired changes.
- Rerun the terraform apply command to apply the changes:

terraform apply

5. Clean Up:

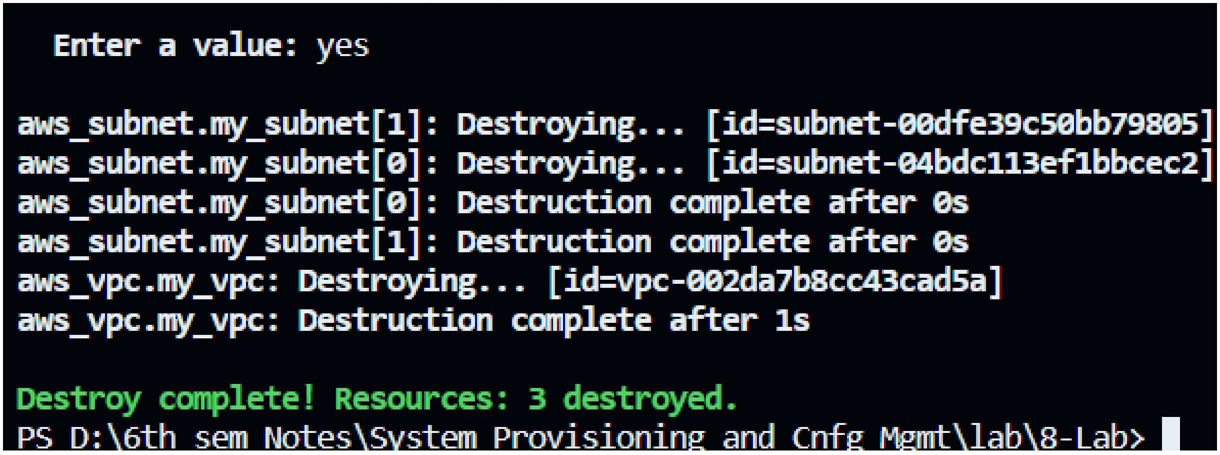
After testing, you can clean up the VPC and subnets:

terraform destroy



```
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\8-Lab> terraform destroy
aws_vpc.my_vpc: Refreshing state... [id=vpc-002da7b8cc43cad5a]
aws_subnet.my_subnet[0]: Refreshing state... [id=subnet-04bdc113ef1bbcec2]
aws_subnet.my_subnet[1]: Refreshing state... [id=subnet-00dfe39c50bb79805]

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

Terraform will perform the following actions:

  # aws_subnet.my_subnet[0] will be destroyed
  - resource "aws_subnet" "my_subnet" {
      - arn                                            = "arn:aws:ec2:ap-south-1:958333040135:subnet/subnet-04bdc113ef1bbcec2" -> null
      - assign_ipv6_address_on_creation                = false -> null
      - availability_zone                              = "ap-south-1a" -> null
      - availability_zone_id                           = "aps1-az1" -> null
      - cidr_block                                     = "10.0.1.0/24" -> null
      - enable_dns64                                   = false -> null
      - enable_lni_at_device_index                     = 0 -> null
      - enable_resource_name_dns_a_record_on_launch    = false -> null
      - enable_resource_name_dns_aaaa_record_on_launch = false -> null
      - id                                             = "subnet-04bdc113ef1bbcec2" -> null
      - ipv6_native                                    = false -> null
      - map_customer_owned_ip_on_launch                = false -> null
      - map_public_ip_on_launch                        = true -> null
      - owner_id                                       = "958333040135" -> null
      - private_dns_hostname_type_on_launch            = "ip-name" -> null
      - tags                                           = {
          - "Name" = "Mysubnet-1"
```

Confirm the destruction by typing yes.



```
  Enter a value: yes

aws_subnet.my_subnet[1]: Destroying... [id=subnet-00dfe39c50bb79805]
aws_subnet.my_subnet[0]: Destroying... [id=subnet-04bdc113ef1bbcec2]
aws_subnet.my_subnet[0]: Destruction complete after 0s
aws_subnet.my_subnet[1]: Destruction complete after 0s
aws_vpc.my_vpc: Destroying... [id=vpc-002da7b8cc43cad5a]
aws_vpc.my_vpc: Destruction complete after 1s

Destroy complete! Resources: 3 destroyed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\8-Lab>
```

6. Conclusion:

This lab exercise demonstrates how to create a basic Virtual Private Cloud (VPC) with
subnets in AWS using Terraform. The example includes a simple VPC configuration
with two subnets. Experiment with different CIDR blocks, settings, and additional AWS resources to customize your VPC.

Lab Exercise 10– Creating an AWS RDS Instance in

Terraform

Objective:

Learn how to use Terraform to create an AWS RDS instance.

Prerequisites:

• Terraform installed on your machine.

• AWS CLI configured with the necessary credentials.

Steps:

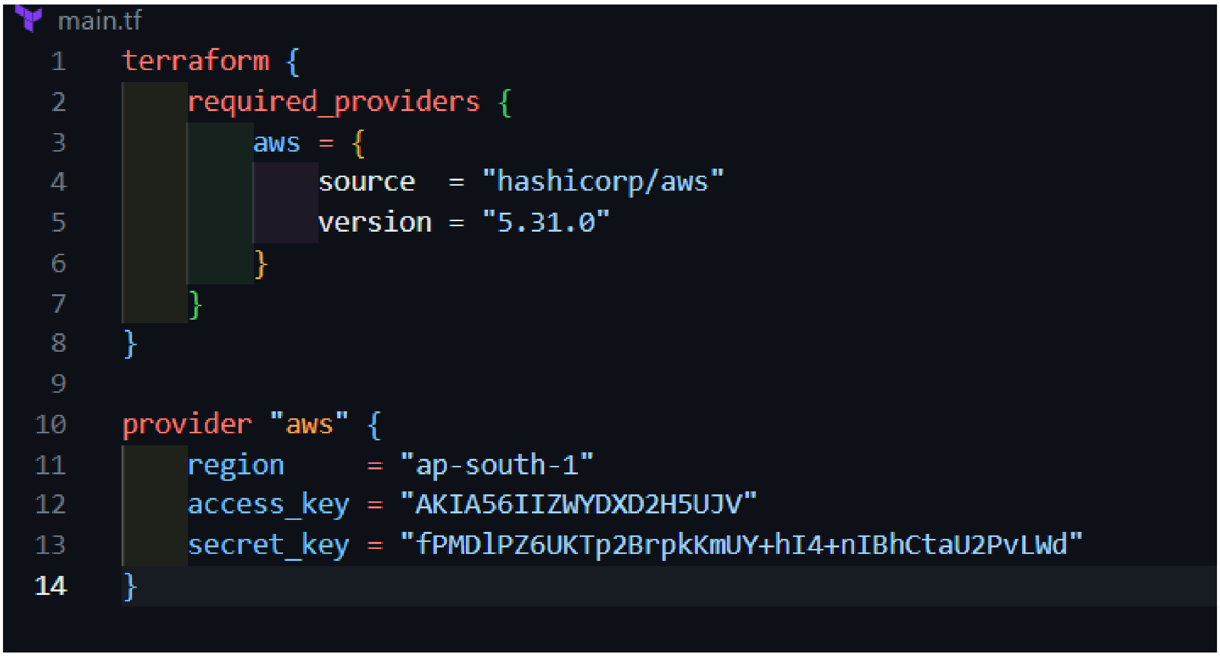
1. Create a Terraform Directory:

main.tf
rds.tf

2. Create Terraform Configuration Files:

Create a file named main.tf:

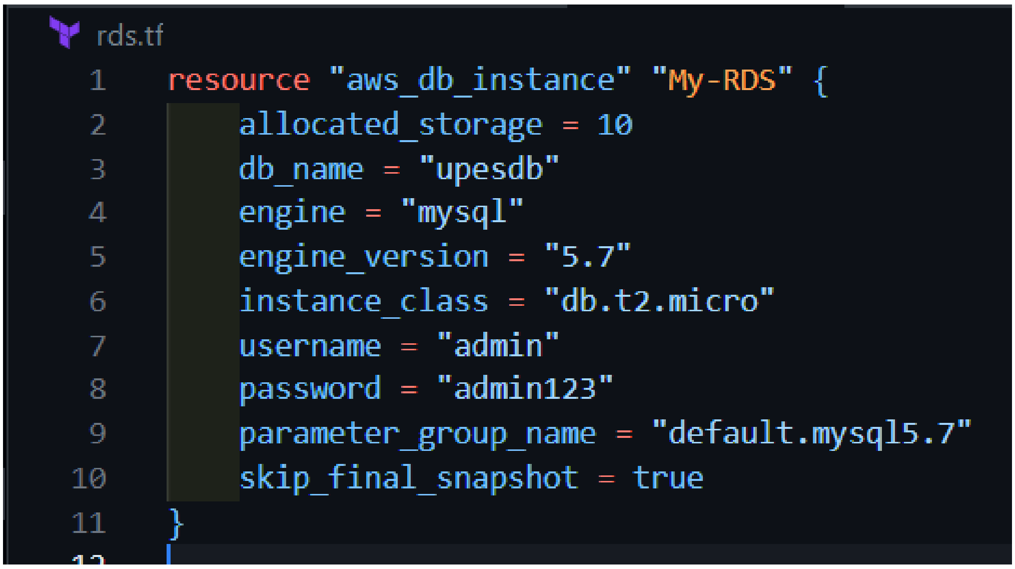
# main.tf



main.tf

```
terraform {
    required_providers {
        aws = {
            source  = "hashicorp/aws"
            version = "5.31.0"
        }
    }
}

provider "aws" {
    region     = "ap-south-1"
    access_key = "AKIA56IIZWYDXD2H5UJV"
    secret_key = "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"
}
```

#rds.tf

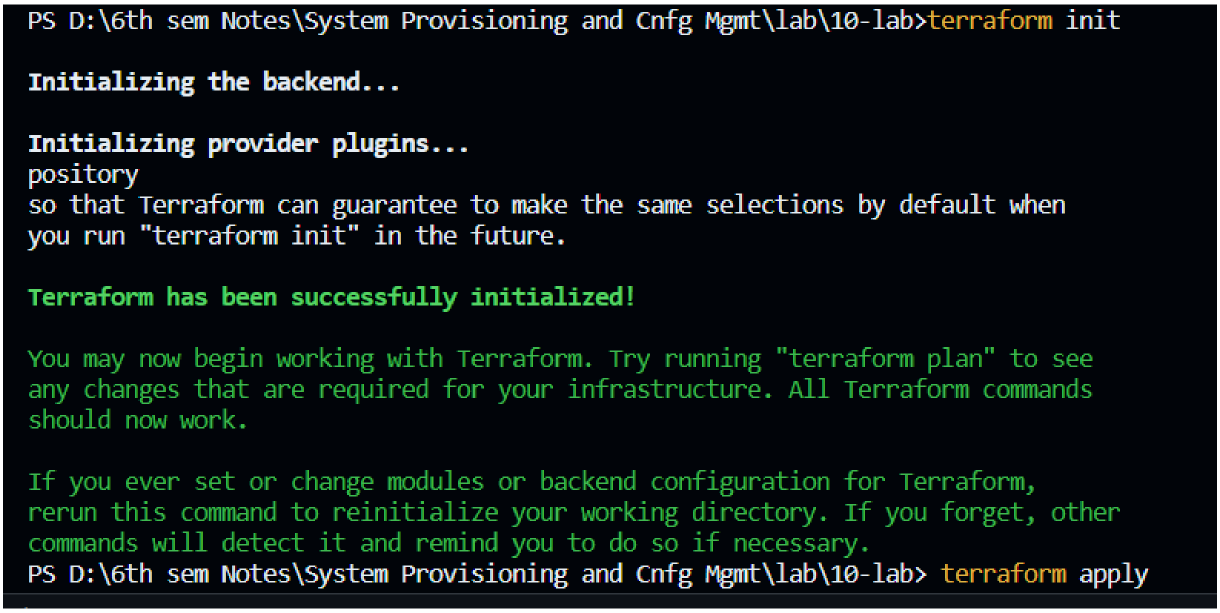


```
resource "aws_db_instance" "My-RDS" {
    allocated_storage = 10
    db_name = "upesdb"
    engine = "mysql"
    engine_version = "5.7"
    instance_class = "db.t2.micro"
    username = "admin"
    password = "admin123"
    parameter_group_name = "default.mysql5.7"
    skip_final_snapshot = true
}
```

3. Initialize and Apply:

- Run the following Terraform commands to initialize and apply the configuration:

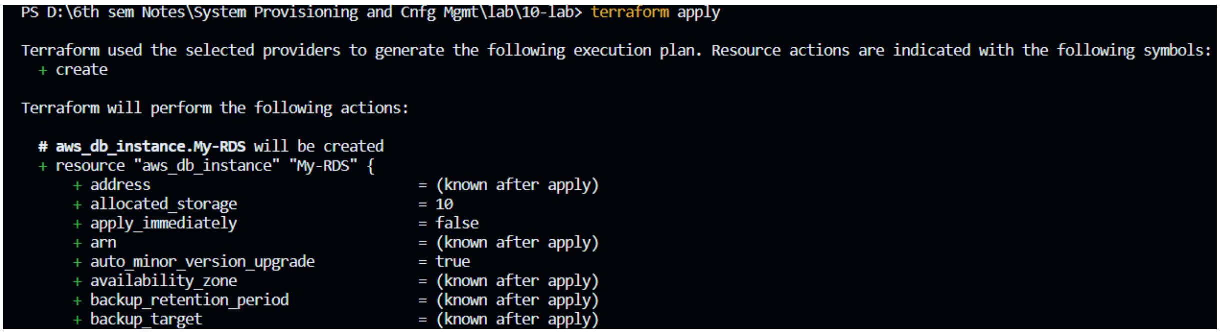
terraform init



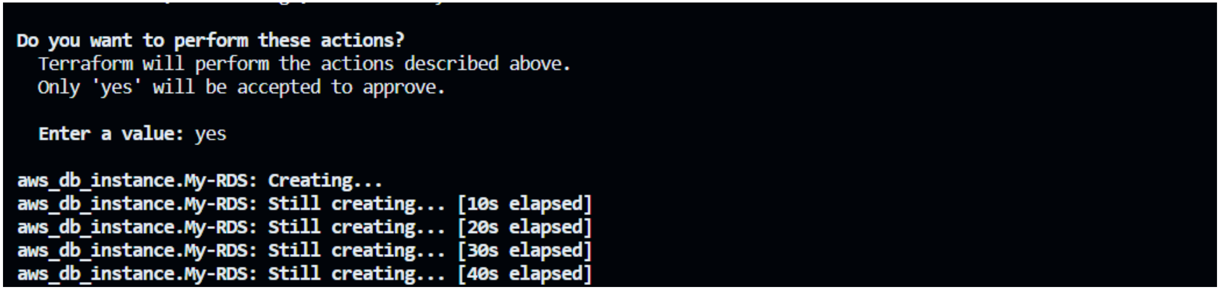
```
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\10-lab>terraform init

Initializing the backend...

Initializing provider plugins...
pository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

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

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 D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\10-lab> terraform apply
```

terraform apply

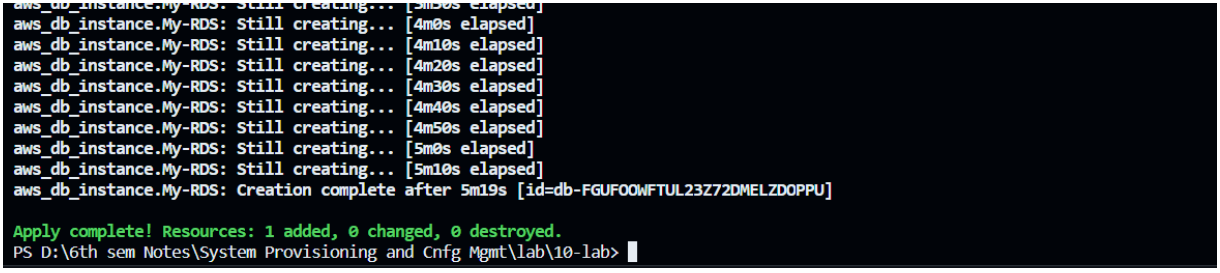
- Terraform will prompt you to confirm the creation of the RDS instance. Type yes and press Enter.



```
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\10-lab> terraform apply

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_db_instance.My-RDS will be created
  + resource "aws_db_instance" "My-RDS" {
      + address                               = (known after apply)
      + allocated_storage                     = 10
      + apply_immediately                     = false
      + arn                                   = (known after apply)
      + auto_minor_version_upgrade            = true
      + availability_zone                     = (known after apply)
      + backup_retention_period               = (known after apply)
      + backup_target                         = (known after apply)
```



```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_db_instance.My-RDS: Creating...
aws_db_instance.My-RDS: Still creating... [10s elapsed]
aws_db_instance.My-RDS: Still creating... [20s elapsed]
aws_db_instance.My-RDS: Still creating... [30s elapsed]
aws_db_instance.My-RDS: Still creating... [40s elapsed]
```

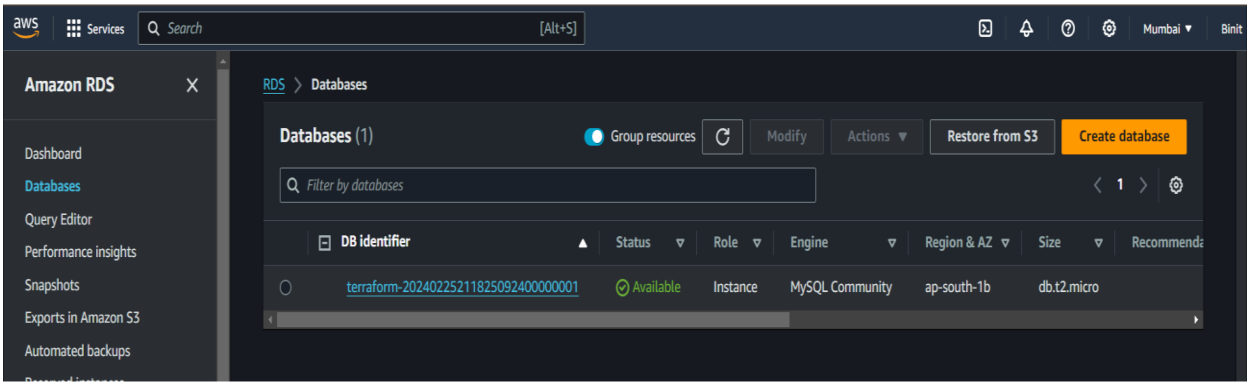


```
aws_db_instance.My-RDS: Still creating... [4m0s elapsed]
aws_db_instance.My-RDS: Still creating... [4m10s elapsed]
aws_db_instance.My-RDS: Still creating... [4m20s elapsed]
aws_db_instance.My-RDS: Still creating... [4m30s elapsed]
aws_db_instance.My-RDS: Still creating... [4m40s elapsed]
aws_db_instance.My-RDS: Still creating... [4m50s elapsed]
aws_db_instance.My-RDS: Still creating... [5m0s elapsed]
aws_db_instance.My-RDS: Still creating... [5m10s elapsed]
aws_db_instance.My-RDS: Creation complete after 5m19s [id=db-FGUFOOWFTUL23Z72DMELZDOPPU]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\10-lab>
```

4. Verify RDS Instance in AWS Console:

• Log in to the AWS Management Console and navigate to the RDS service.

• Verify that the specified RDS instance with the specified settings has been created.



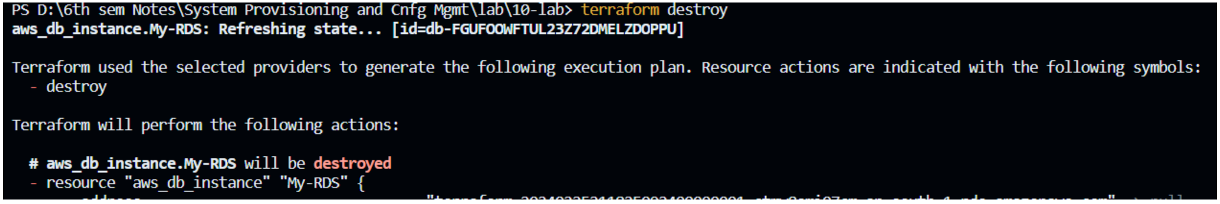
Databases (1)

	DB identifier	Status	Role	Engine	Region & AZ	Size	Recommenda
○	terraform-20240225211825092400000001	Available	Instance	MySQL Community	ap-south-1b	db.t2.micro	

6. Clean Up:

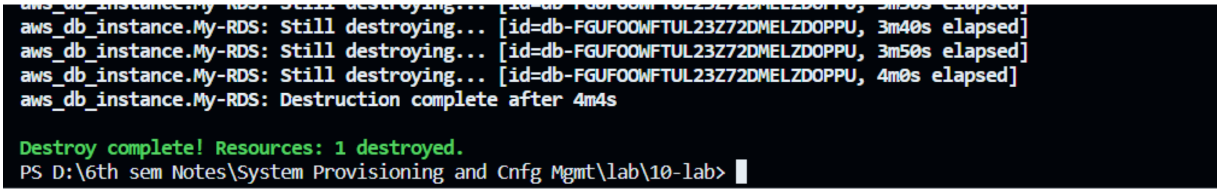
After testing, you can clean up the RDS instance:

terraform destroy



```
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\10-lab> terraform destroy
aws_db_instance.My-RDS: Refreshing state... [id=db-FGUFOOWFTUL23Z72DMELZDOPPU]

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

Terraform will perform the following actions:

  # aws_db_instance.My-RDS will be destroyed
  - resource "aws_db_instance" "My-RDS" {
```

Confirm the destruction by typing yes.



```
aws_db_instance.My-RDS: Still destroying... [id=db-FGUFOOWFTUL23Z72DMELZDOPPU, 3m30s elapsed]
aws_db_instance.My-RDS: Still destroying... [id=db-FGUFOOWFTUL23Z72DMELZDOPPU, 3m40s elapsed]
aws_db_instance.My-RDS: Still destroying... [id=db-FGUFOOWFTUL23Z72DMELZDOPPU, 3m50s elapsed]
aws_db_instance.My-RDS: Still destroying... [id=db-FGUFOOWFTUL23Z72DMELZDOPPU, 4m0s elapsed]
aws_db_instance.My-RDS: Destruction complete after 4m4s

Destroy complete! Resources: 1 destroyed.
PS D:\6th sem Notes\System Provisioning and Cnfg Mgmt\lab\10-lab>
```

7. Conclusion:

This lab exercise demonstrates how to use Terraform to create an AWS RDS instance.

You learned how to define RDS settings, initialize and apply the Terraform configuration, and verify the creation of the RDS instance in the AWS Management
Console. Experiment with different RDS settings in the main.tf file to observe how