**Lab Exercise 1**

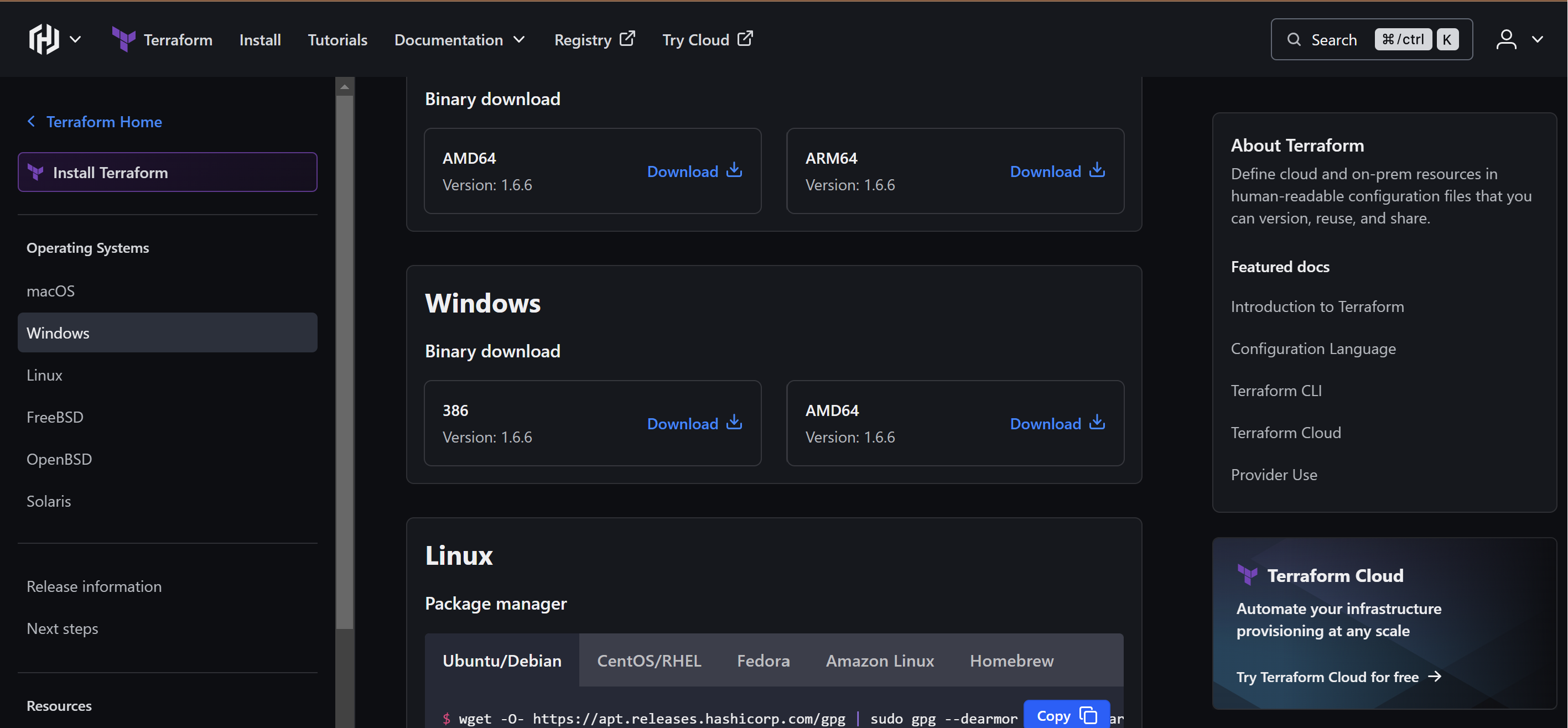
**Install Terraform on Windows**

Installing Terraform on Windows requires you to download the correct Terraform package, unpack, and execute it via the [CLI.](https://phoenixnap.com/glossary/common-language-infrastructure) 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](https://developer.hashicorp.com/terraform/downloads) page.
2. Select the Windows tab under the **Operating System** heading. The latest version is preselected.

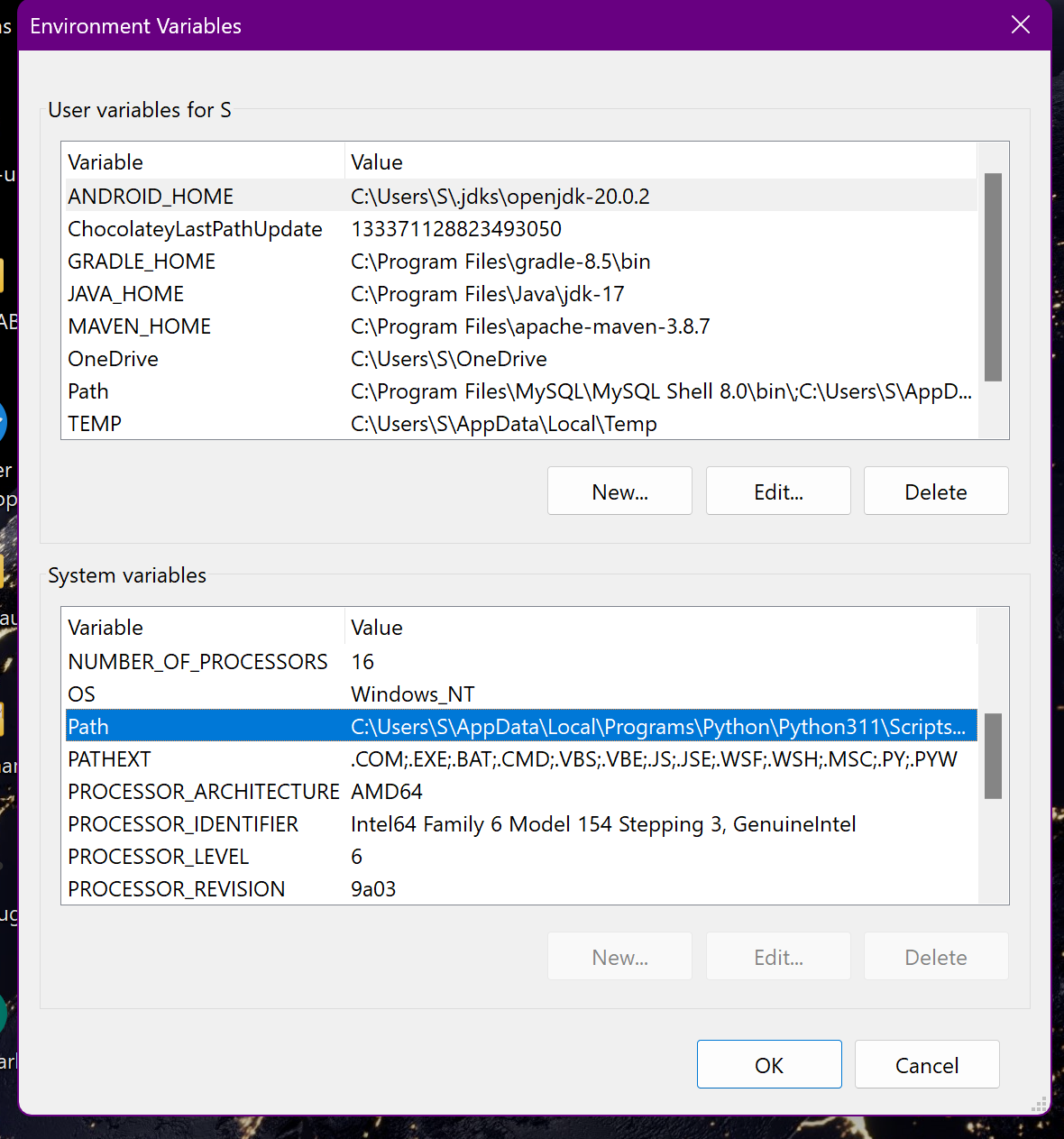


1. Choose the binary to download. Select 386 for 32-bit systems or [AMD64](https://phoenixnap.com/glossary/amd64) for 64-bit systems. Choose the download location for the zip file if the download does not start automatically.
2. Unzip the downloaded file. For example, use the *C:\terraform* path. Remember this location so you can add the path to the environment variables.

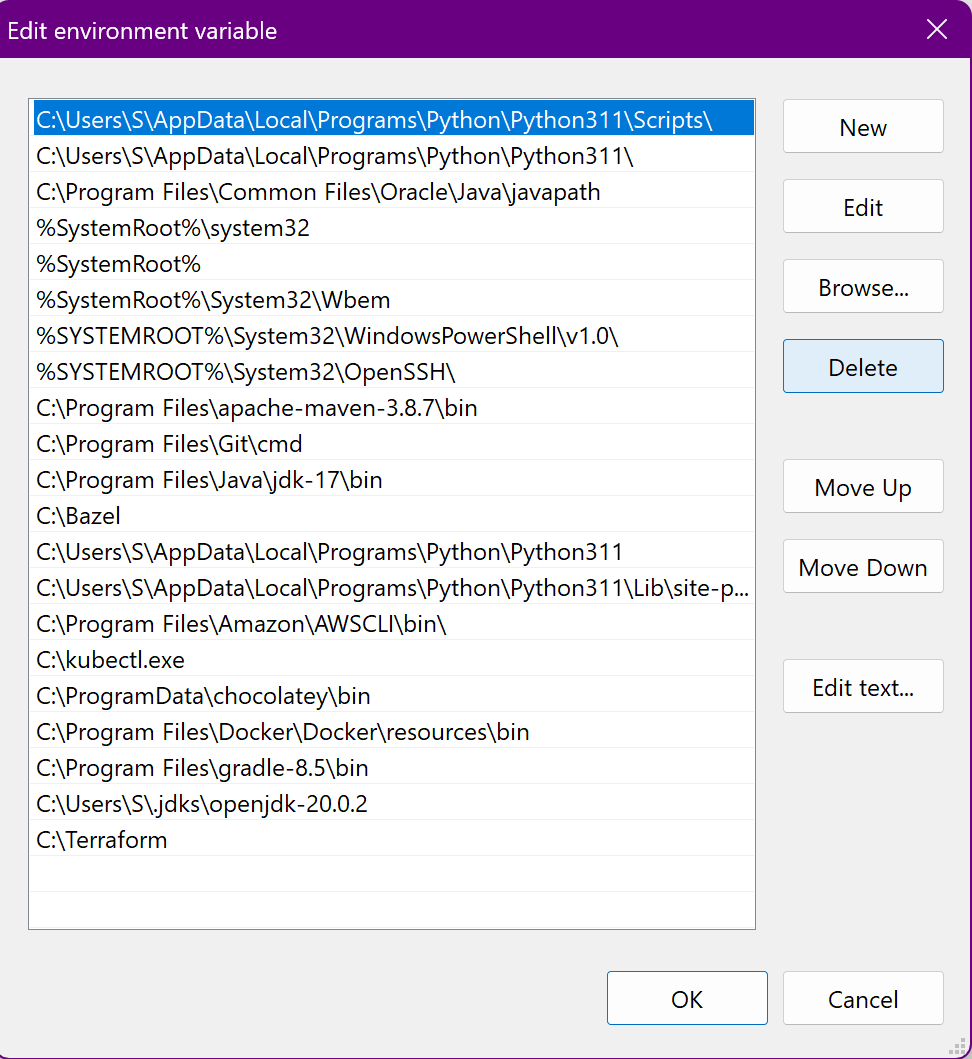
# **Add Terraform Path to System Environment Variables**

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.



1. 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**.**
2. Click **New** in the edit window and enter the location of the Terraform folder.



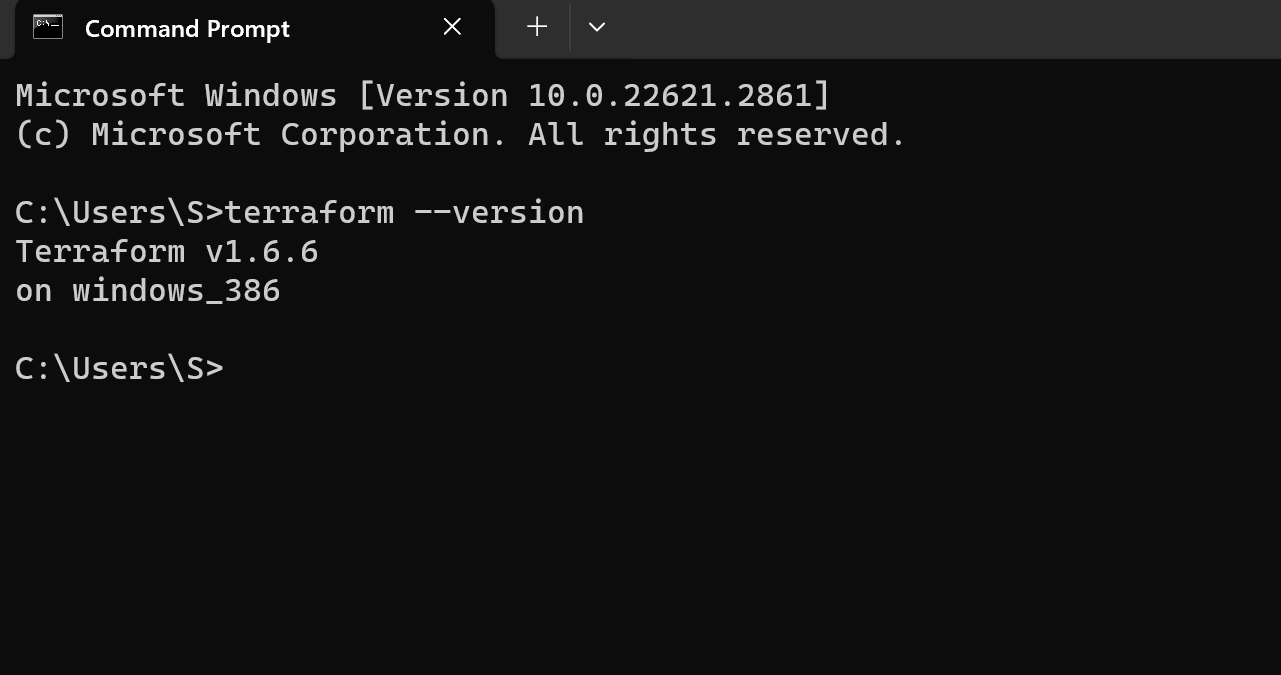
1. Click **OK** on all windows to apply the changes.

# **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

terraform -version



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 configured AWS credentials (Access Key ID and Secret Access Key). 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**

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

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

**terraform {**

**required\_providers {**

**aws = {**

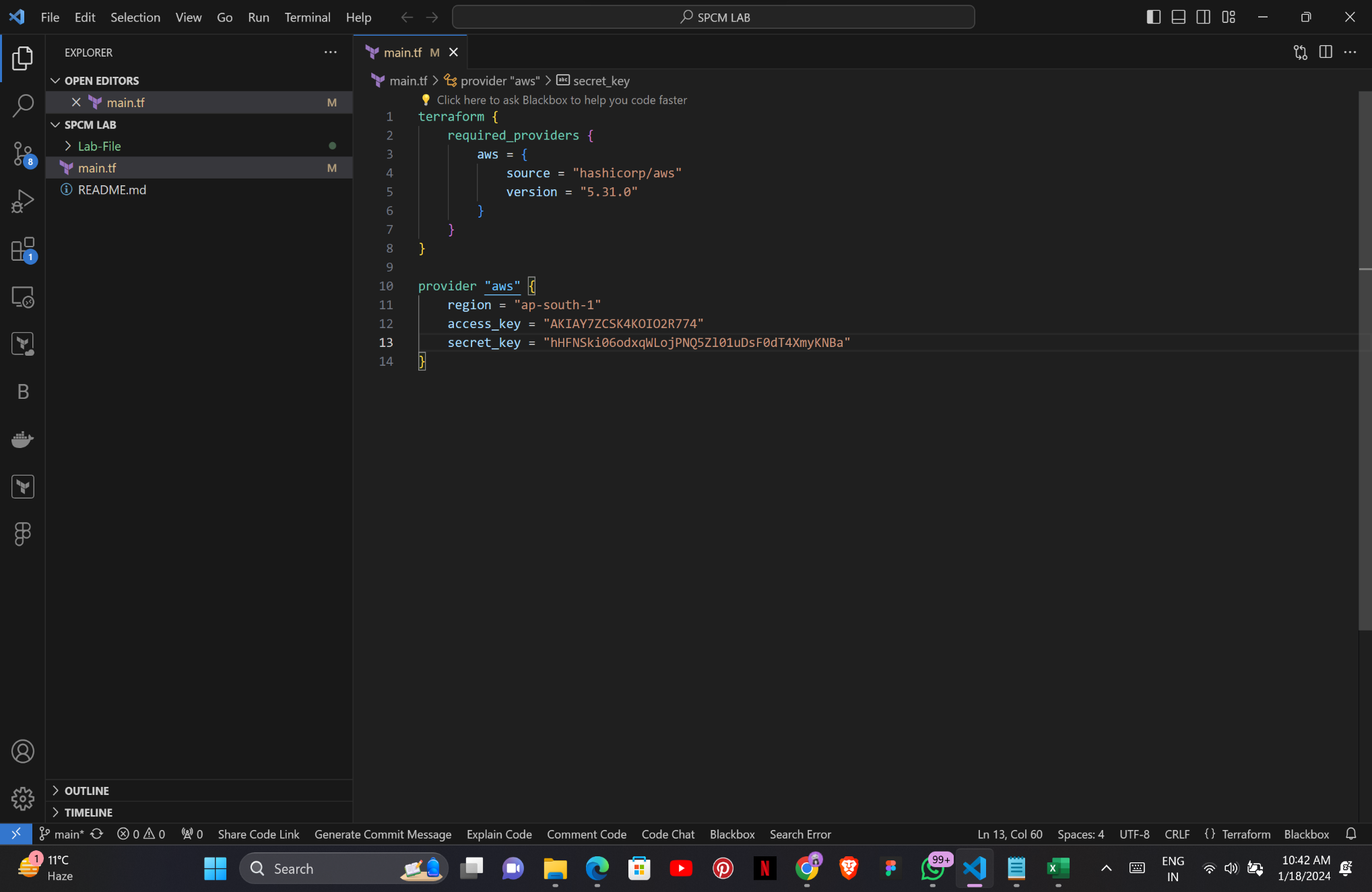
**source = "hashicorp/aws"**

**version = "5.31.0"**

**}**

**}**

**}**



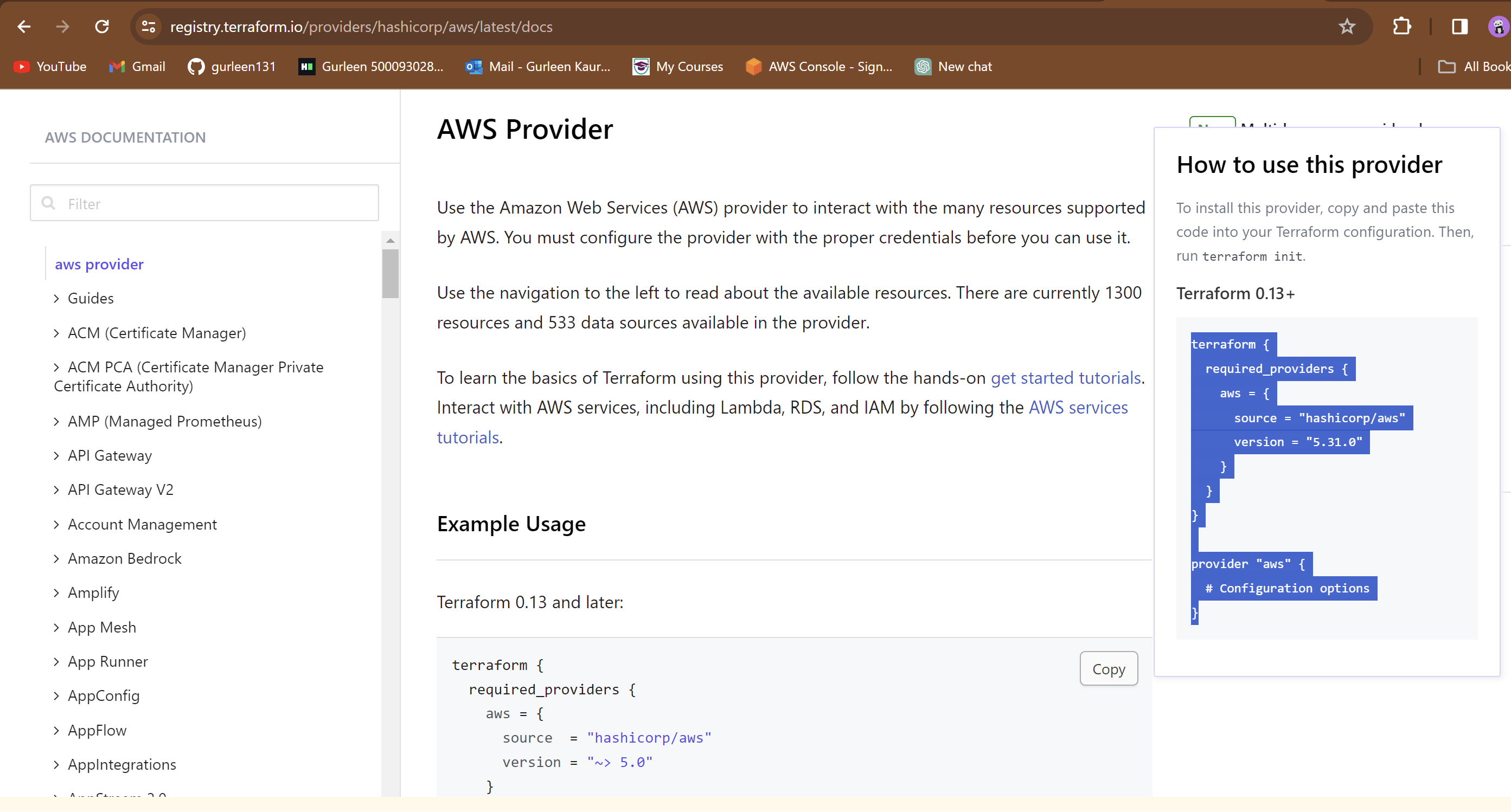
**provider "aws" {**

**region = "ap-south-1"**

**access\_key = "your IAM access key"**

**secret\_key = "your secret access key"**

**}**



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**

****

**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 configured AWS credentials (Access Key ID and Secret Access Key). You can set them up using the AWS CLI or setting environment variables.

**Exercise Steps:**

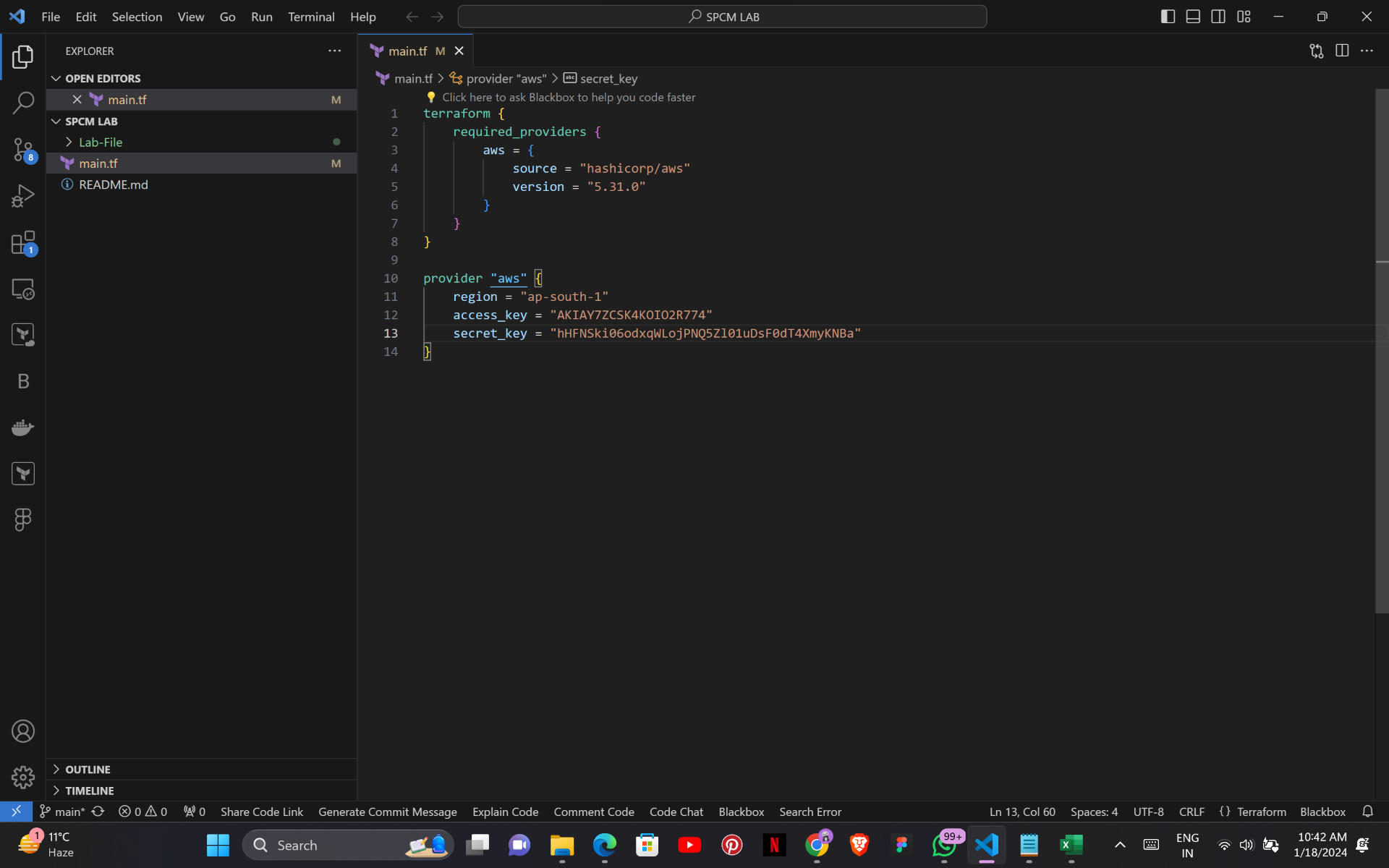
**Step 1: Create a New Directory:**

Create a new directory for your Terraform configuration:

**mkdir aws-terraform-demo**

**cd aws-terraform-demo**

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

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

**terraform {**

**required\_providers {**

**aws = {**

**source = "hashicorp/aws"**

**version = "5.31.0"**

**}**

**}**

**}**

**provider "aws" {**

**region = "ap-south-1"**

**access\_key = "your IAM access key"**

**secret\_key = "your secret access key"**

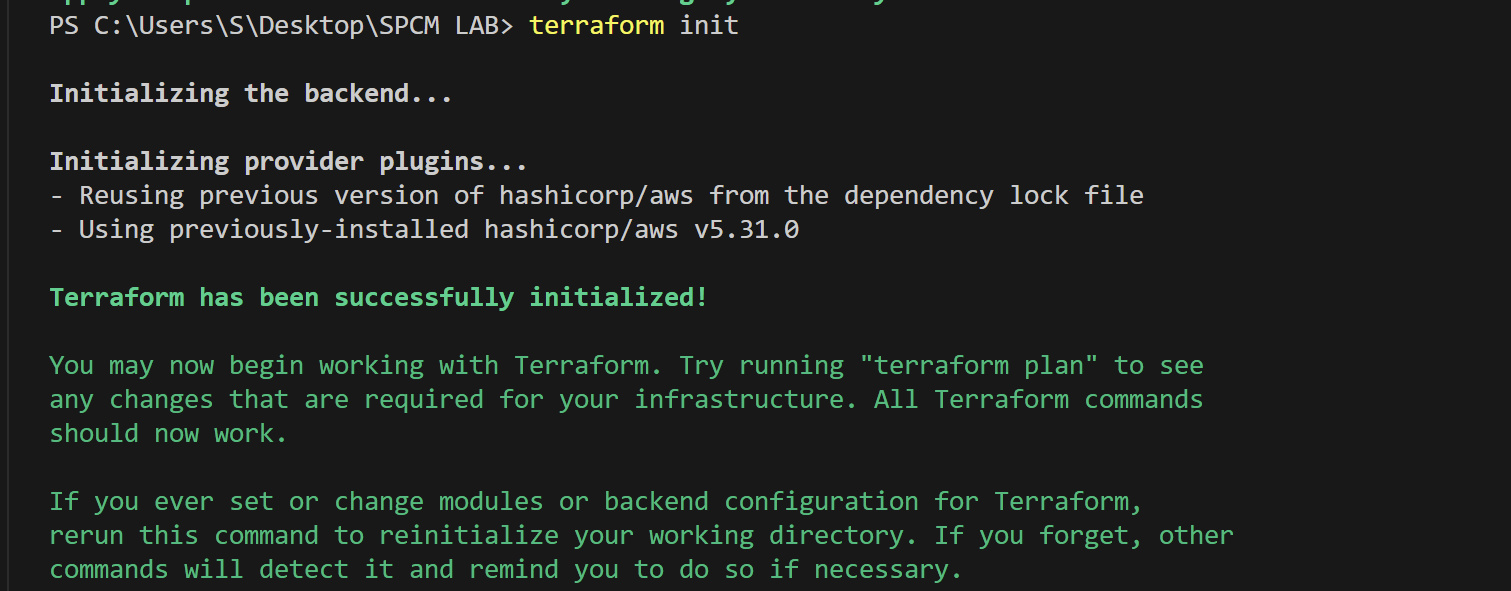
**}**

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**



**Step 4: Create Terraform Configuration File for EC2 instance**

**(instance.tf):**

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

**resource "aws\_instance" "My-instance" {**

**instance\_type = "t2.micro"**

**ami = "ami-03f4878755434977f"**

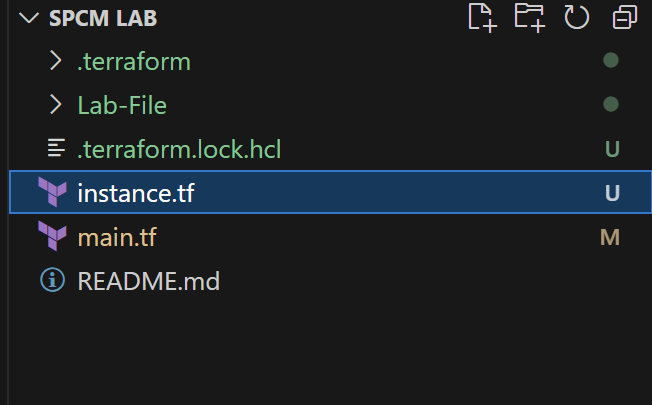
**count = 1**

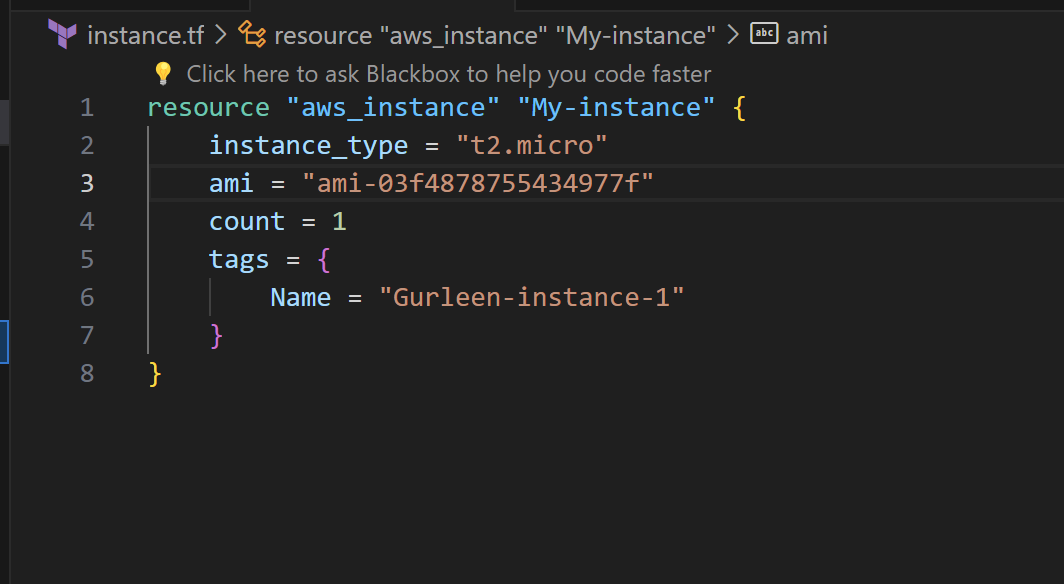
**tags = {**

**Name = "UPES-EC2-Instnace"**

**}**

**}**





**Step 5: Review Plan:**

Run the following command to see what Terraform will do:

**terraform plan**

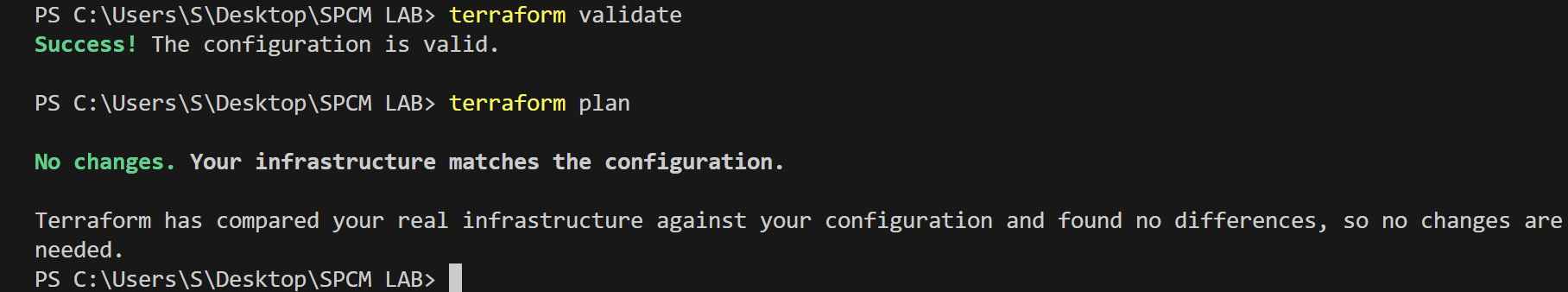
Review the plan to ensure it aligns with your expectations.

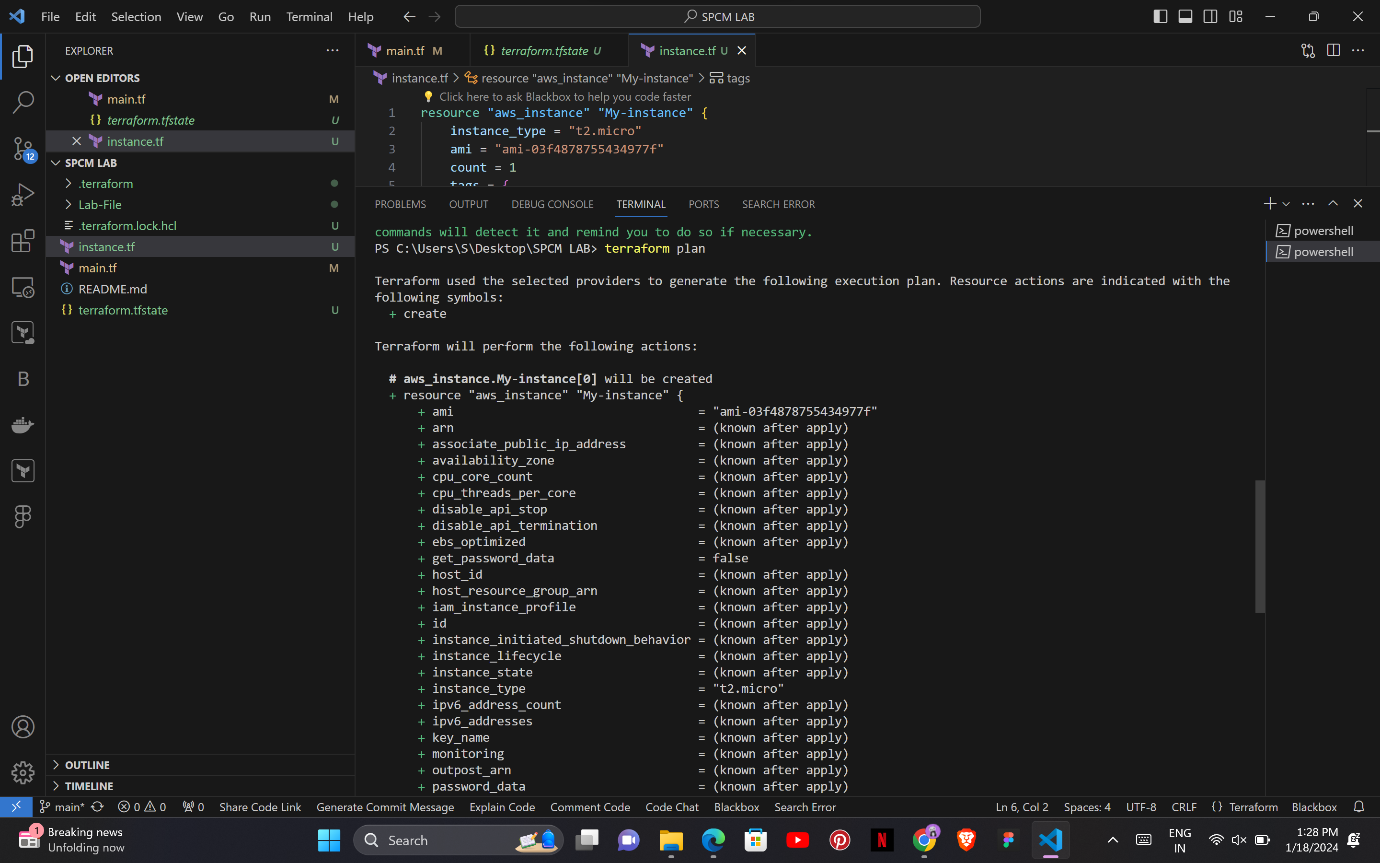
**Step 6: Apply Changes:**

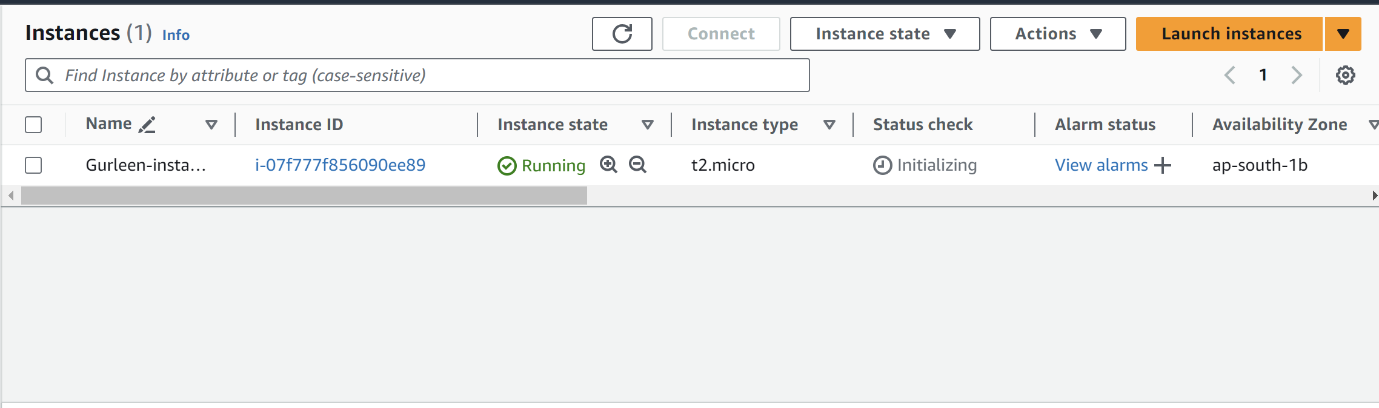
Apply the changes to create the AWS resources:

**terraform apply**

Type yes when prompted.







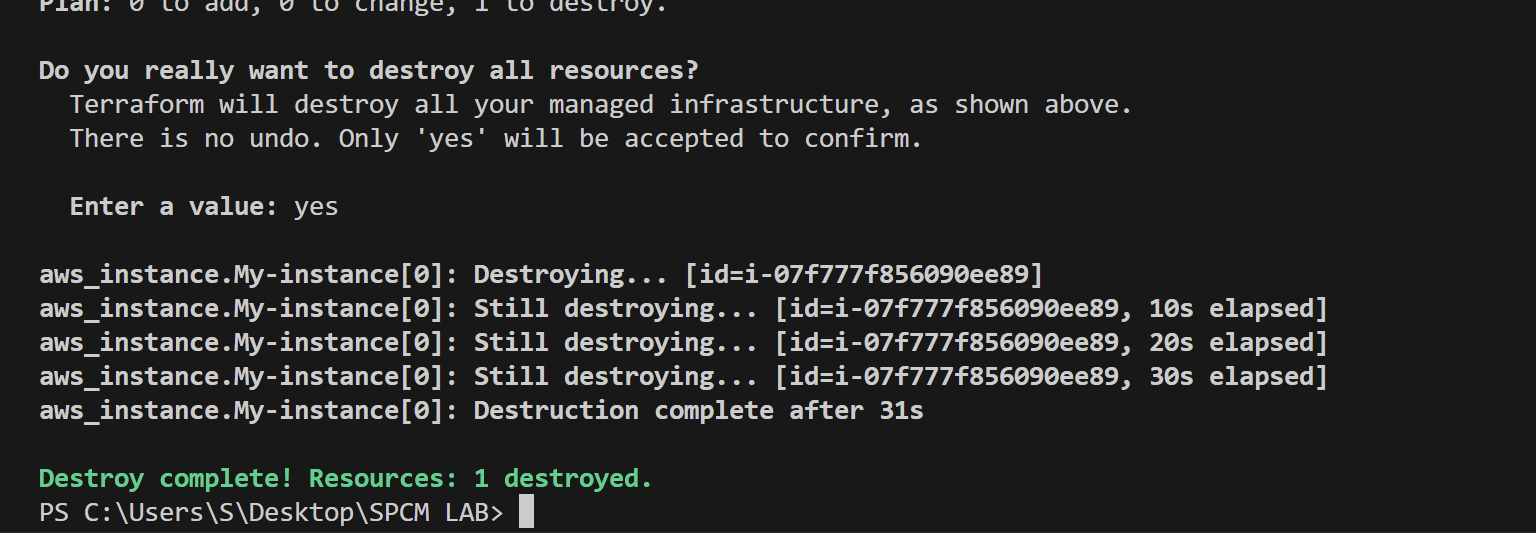
**Step 7: Verify Resources:**

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 resources:

**terraform destroy**



**Lab Exercise 4– Terraform Variables**

**Objective:**

Learn how to define and use variables in Terraform configuration.

**Prerequisites:**

Install Terraform on your machine.

**Steps:**

**Step 1: Create a Terraform Directory:**

Create a new directory for your Terraform project.

**Step 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" "example" {**

**ami = "ami-0c55b159cbfafe1f0"**

**instance\_type = "t2.micro”**

**}**

**Step 3: Define Variables:**

Open a new file named variables.tf. Define variables for region, ami, and instance\_type.

**# variables.tf**

**variable "region" {**

**description = "AWS region"**

**default = "us-west-2"**

**}**

**variable "ami" {**

**description = "AMI ID"**

**default = "ami-0c55b159cbfafe1f0"**

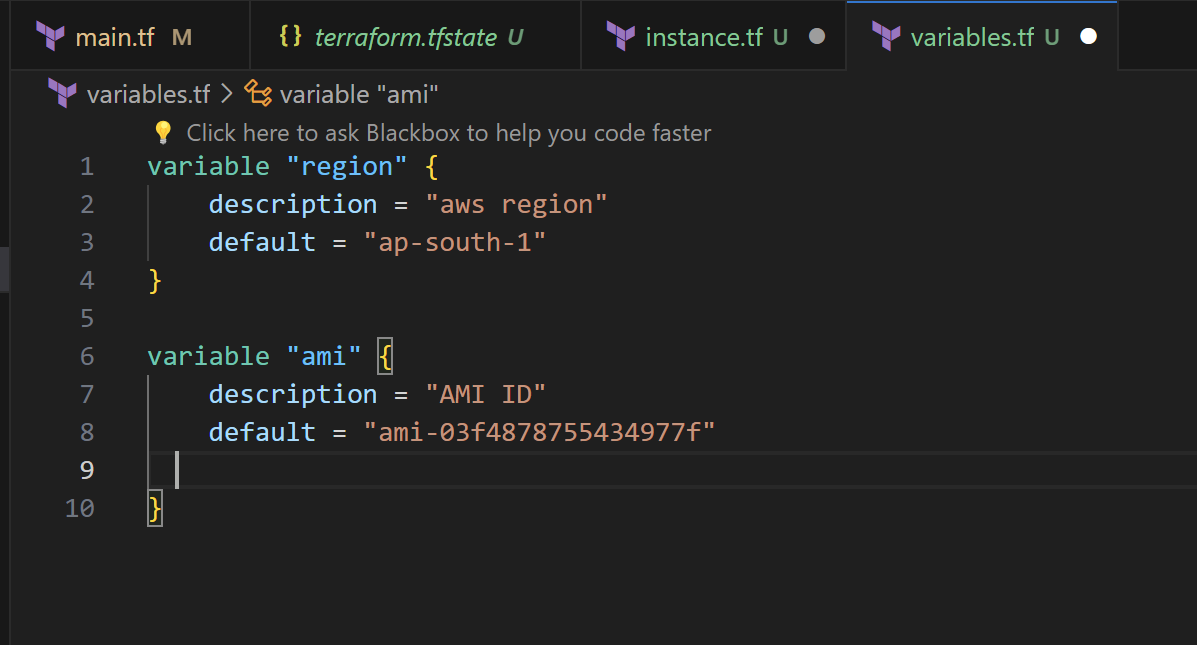
**}**

**variable "instance\_type" {**

**description = "EC2 Instance Type"**

**default = "t2.micro"**

**}**



**Step 4: Use Variables in main.tf:**

Modify main.tf to use the variables.

**# main.tf**

**provider "aws" {**

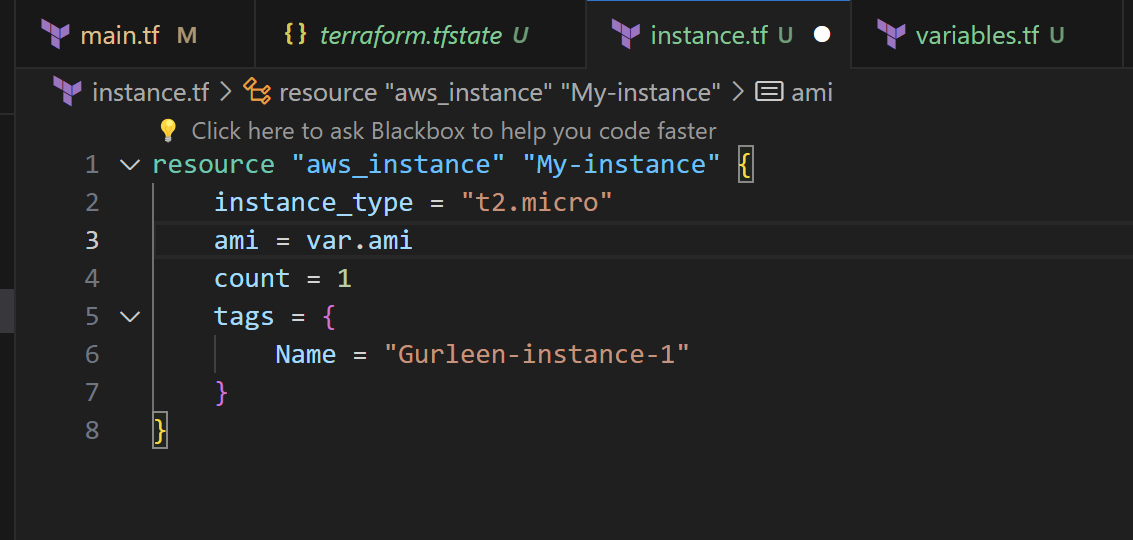
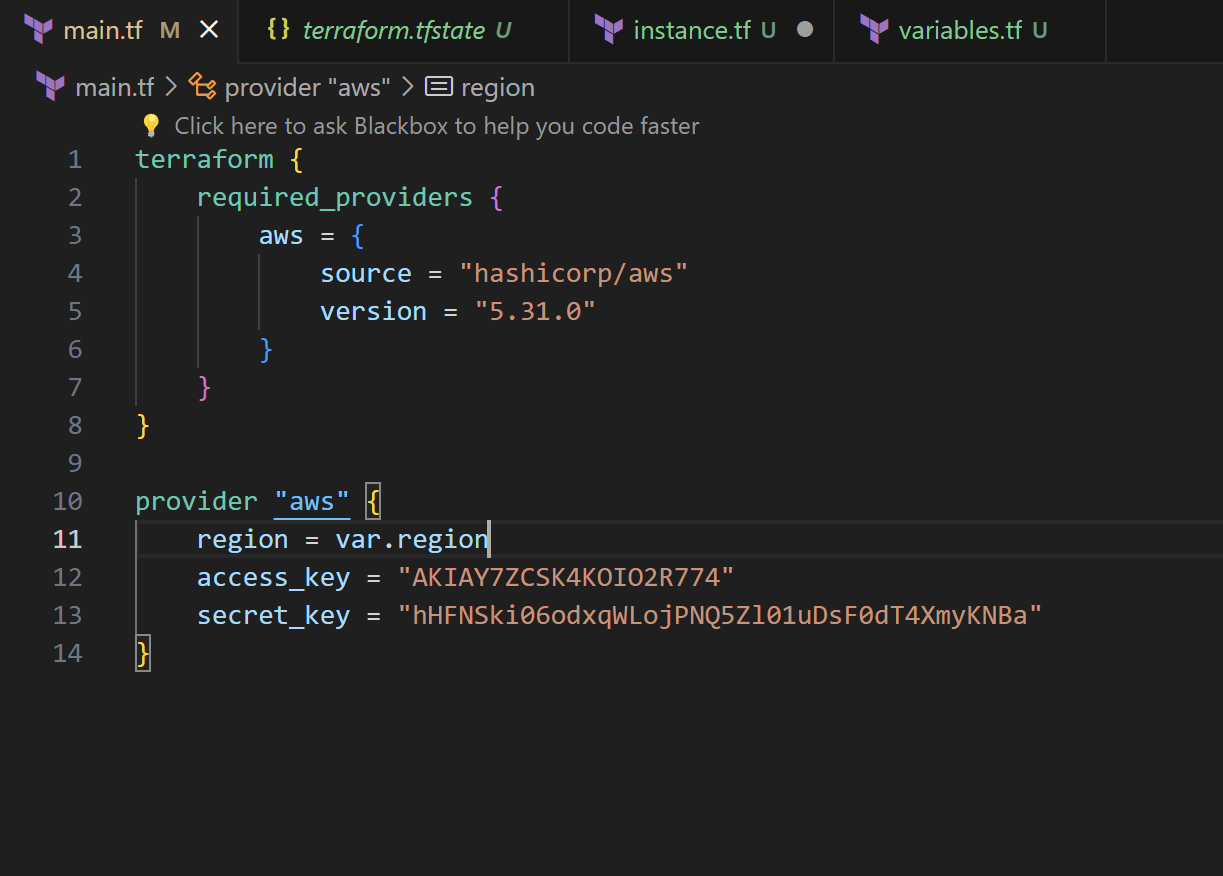
**region = var.region**

**resource "aws\_instance" "example" {**

**ami = var.ami**

**instance\_type = var.instance\_type**

**}**



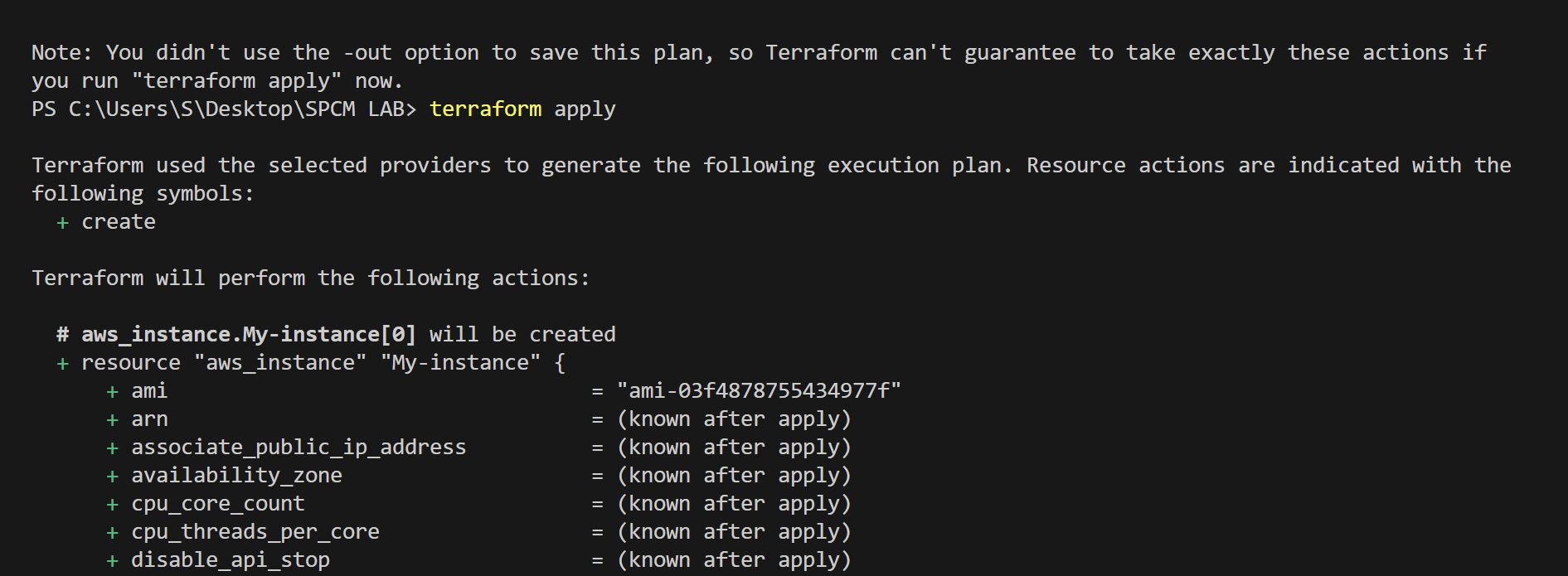
**Step 5: Initialize and Apply:**

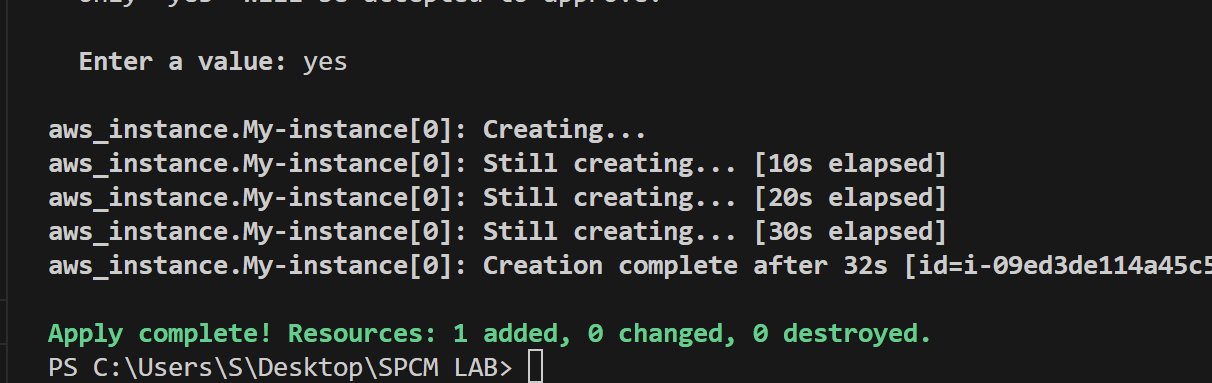
Run the following Terraform commands to initialize and apply the configuration.

**terraform init**

**terraform apply**

Observe how the region changes based on the variable override.



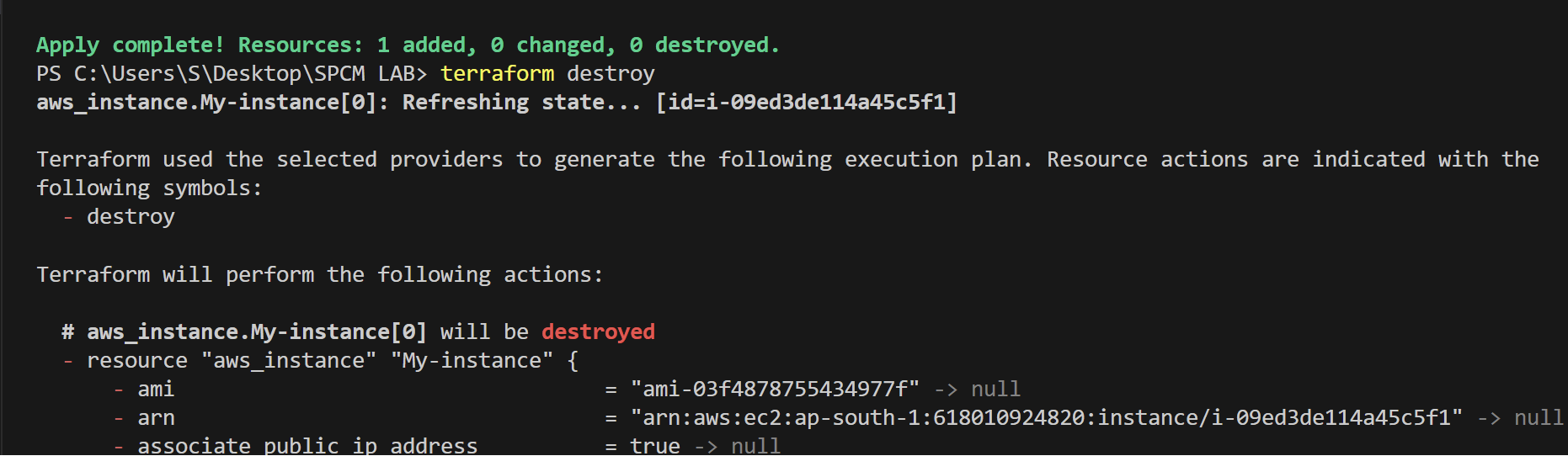


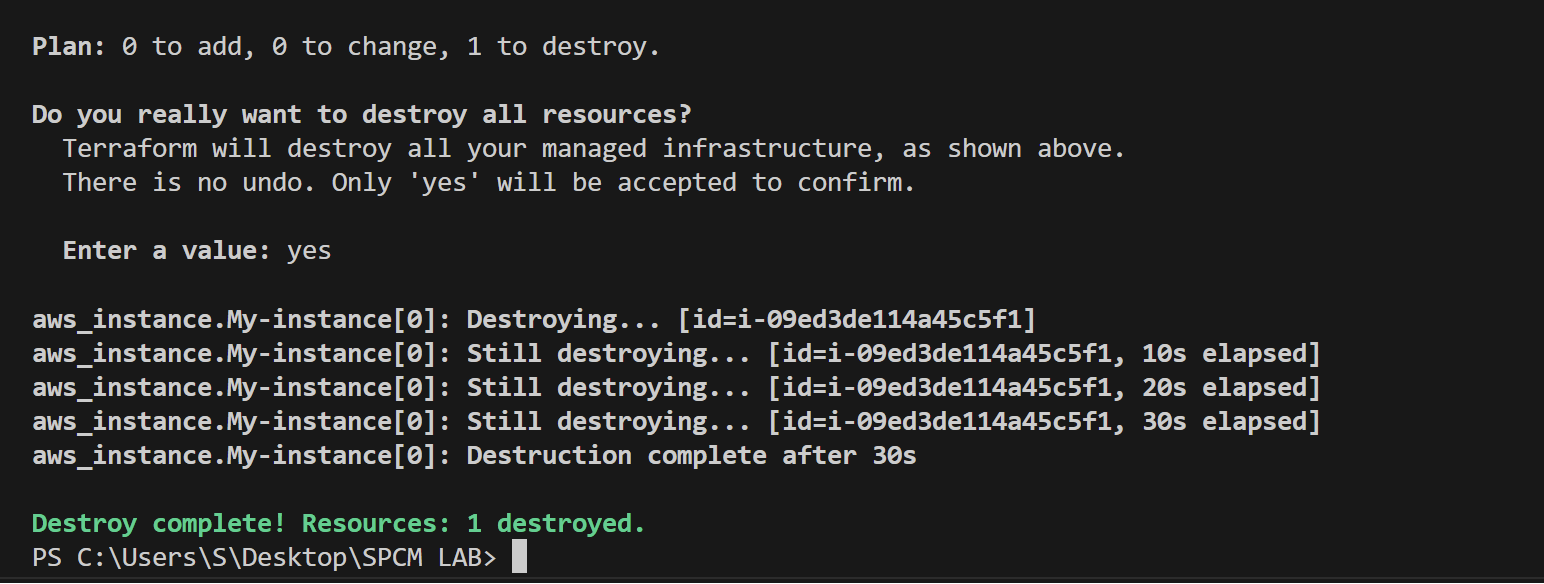
**Step 6: Clean Up:**

After testing, you can clean up resources.

**terraform destroy**

Confirm the destruction by typing yes.





**Step 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.

**Step 1: Create Terraform Configuration Files:**

# variables.tf

**variable "region" {**

**description = "AWS region"**

**default = "us-west-2"**

**}**

**variable "ami" {**

**description = "AMI ID"**

**default = "ami-0c55b159cbfafe1f0"**

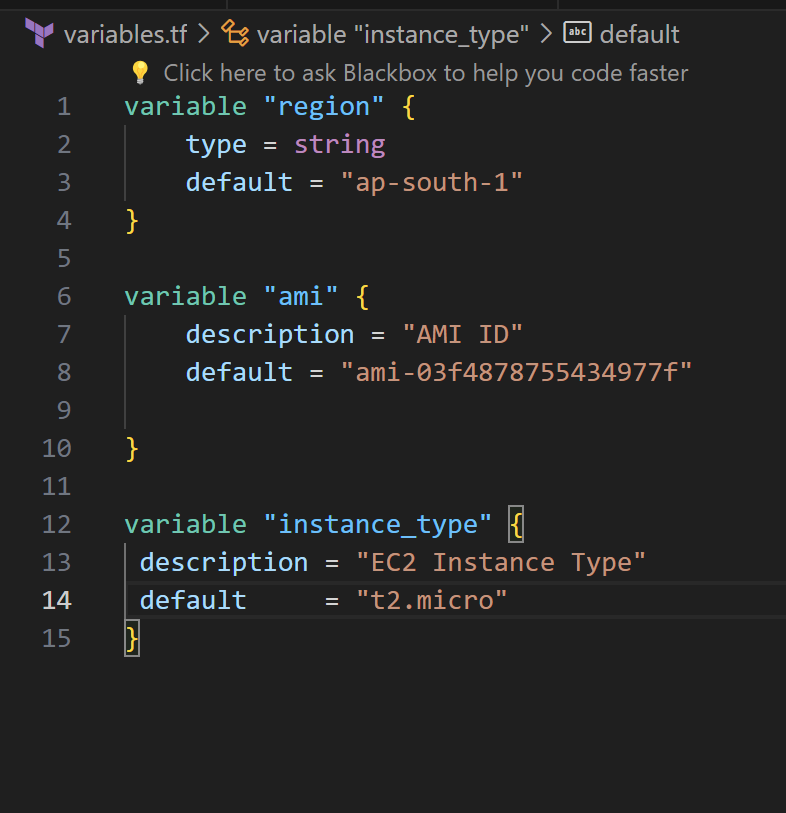
**}**

**variable "instance\_type" {**

**description = "EC2 Instance Type"**

**default = "t2.small"**

**}**

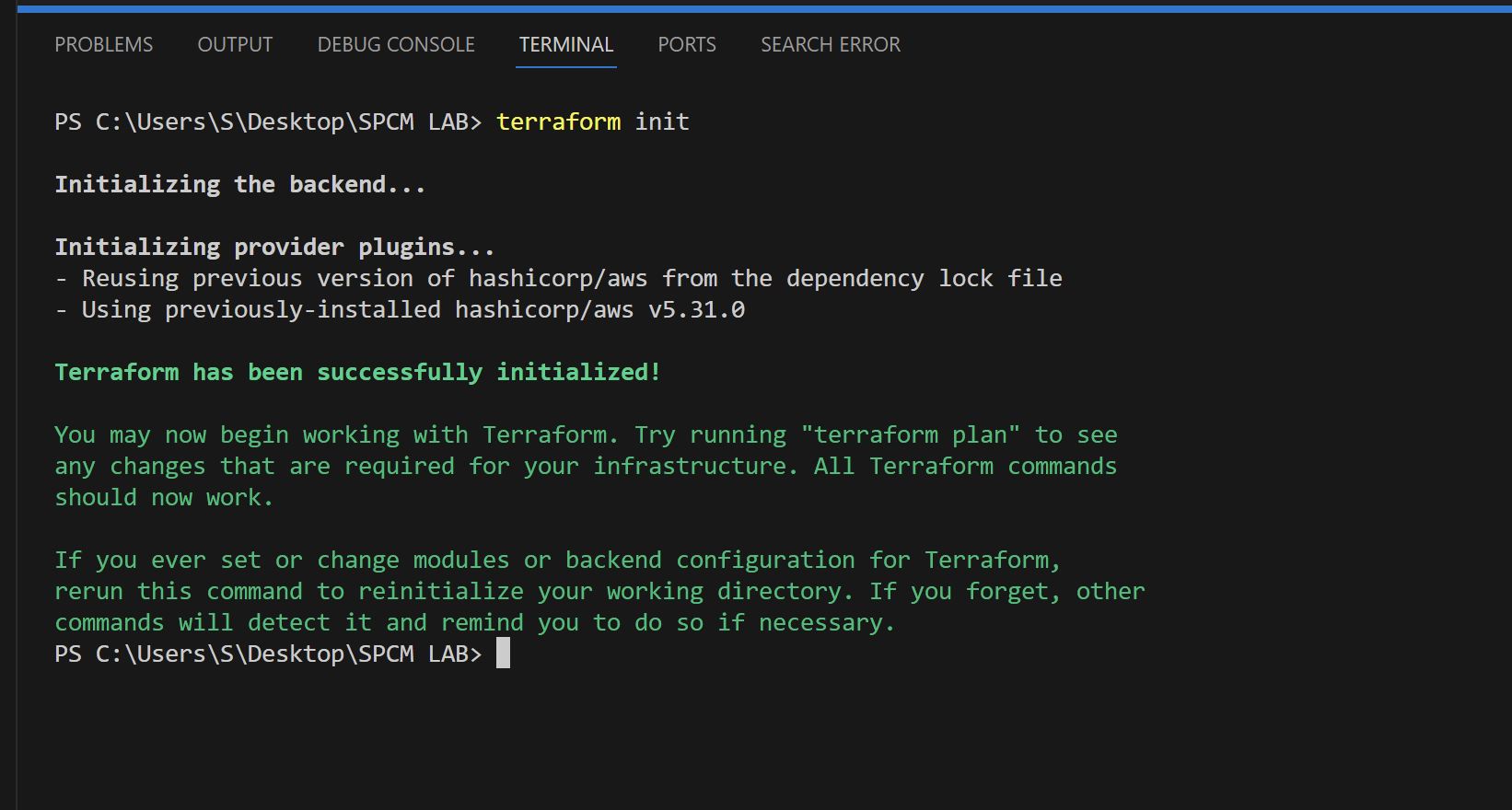


**Step 2: Use Command Line Arguments:**

Open a terminal and navigate to your Terraform project directory.

Run the terraform init command:

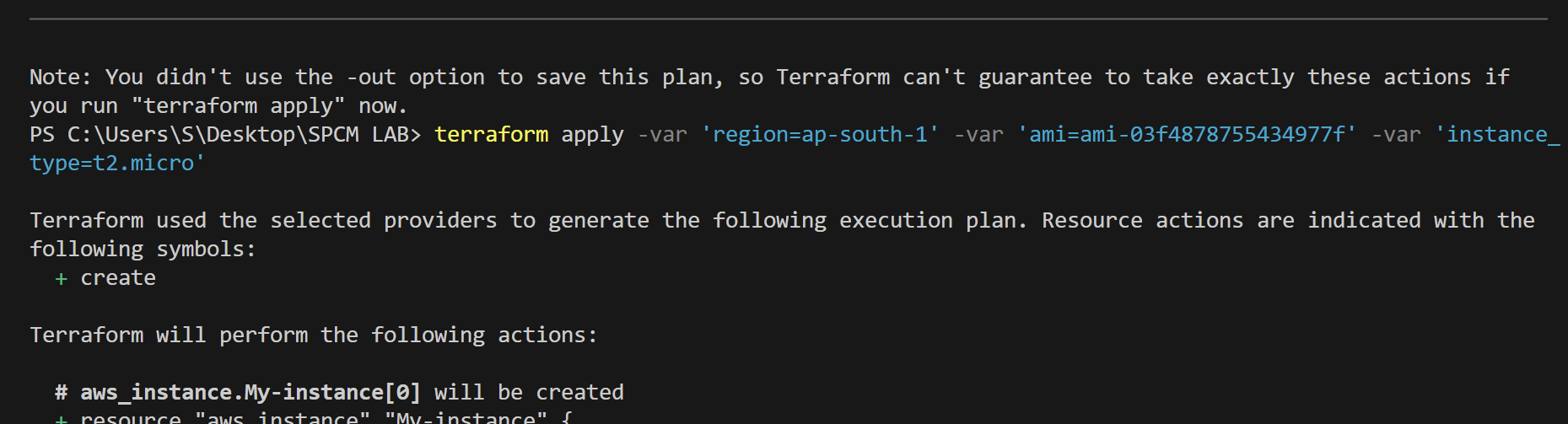
**terraform init**

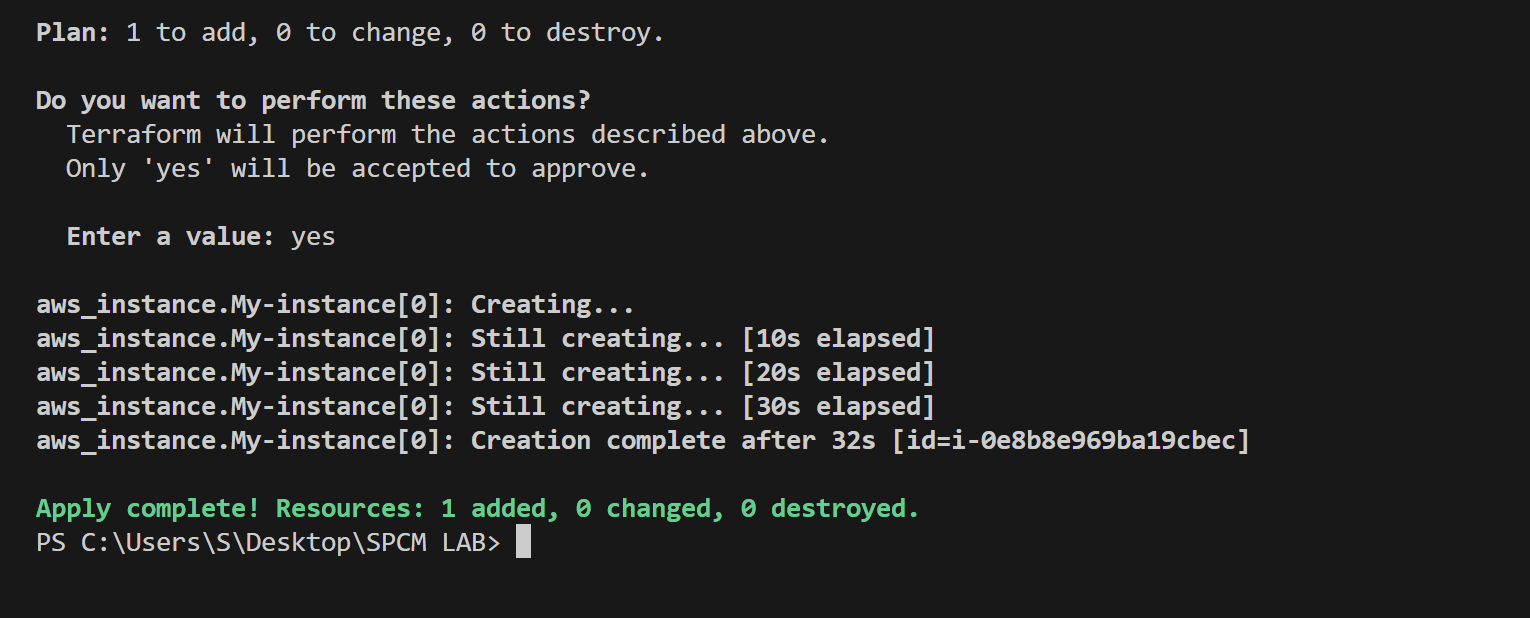


Run the terraform apply command with command line arguments to set

variable values:

**terraform apply -var 'region=us-east-1' -var 'ami=ami-12345678' -var** **' instance\_type=t2.micro**





**Step 3: Test and Verify:**

Observe how the command line arguments dynamically set the variable values during the apply process.

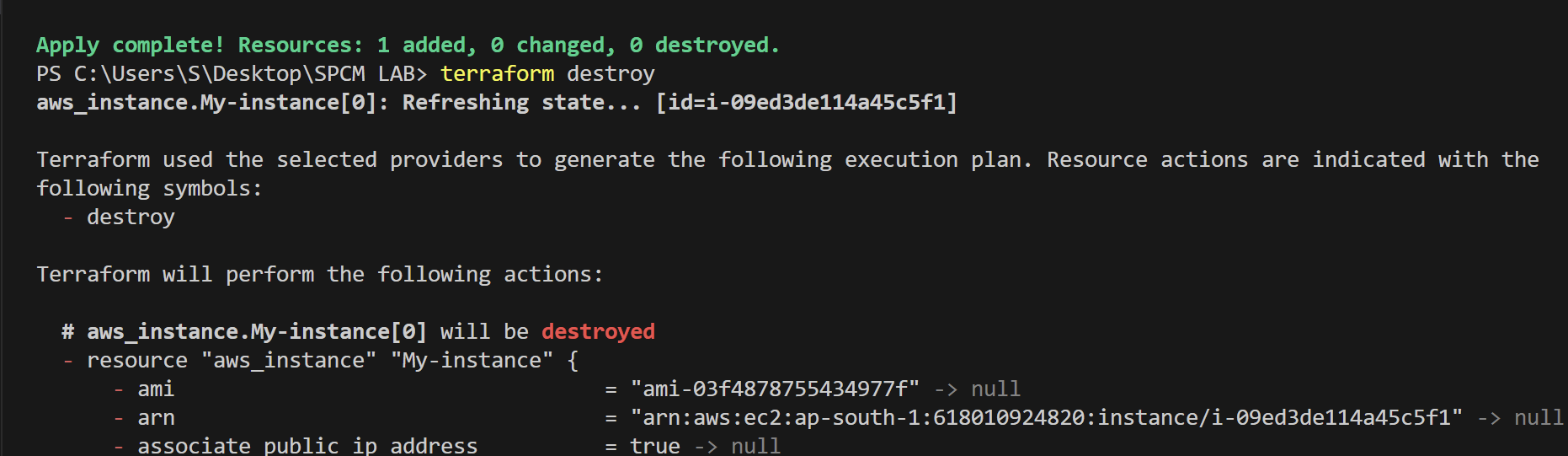
Access the AWS Management Console or use the AWS CLI to verify the creation of resources in the specified region.

**Step 4: Clean Up:**

After testing, you can clean up resources:

**terraform destroy**

Confirm the destruction by typing yes.



**Step 5: Conclusion:**

This lab exercise demonstrates how to use command line arguments to set variable values 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 7 – Creating Multiple IAM Users in Terraform**

**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:**

**mkdir terraform-iam-users cd terraform-iam-users**

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

## # main.tf

**provider "aws" { region = "us-east-1"**

**}**

**variable "iam\_users" { type = list(string)**

**default = ["user1", "user2", "user3"]**

**}**

**resource "aws\_iam\_user" "iam\_users" { count = length(var.iam\_users)**

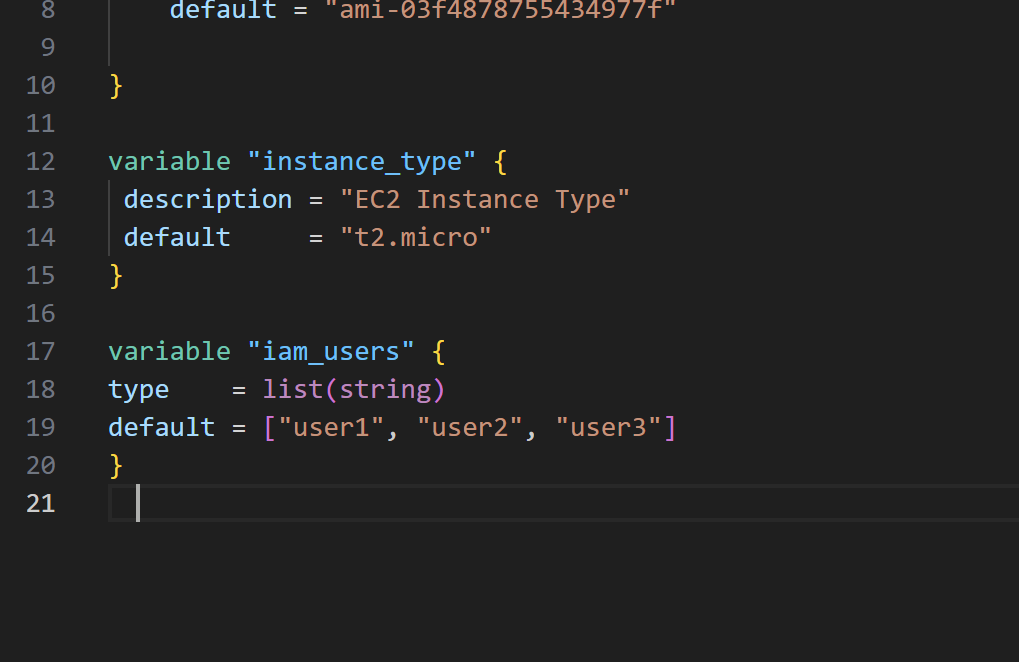
**name = var.iam\_users[count.index]**

**tags = {**

**Name = "${var.iam\_users[count.index]}-user"**

**}**

**}**



In this configuration, we define a list variable iam\_users containing the names of the IAM users we want to create. The aws\_iam\_user resource is then used in a loop to create users based on the values in the list.

# **Initialize and Apply:**

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

**Terraform init provider "aws" { region = "us-east-1"**

**}**

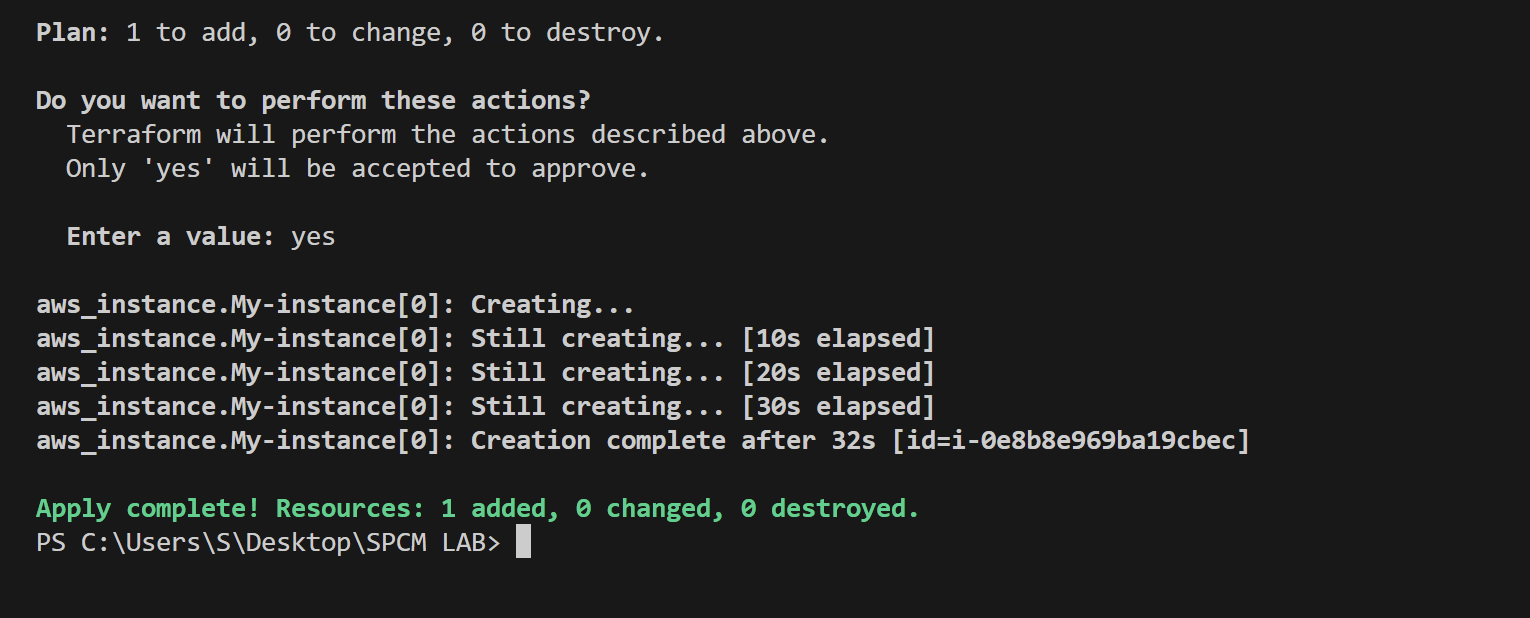
**resource "aws\_vpc" "my\_vpc" { cidr\_block = "10.0.0.0/16" enable\_dns\_support = true enable\_dns\_hostnames = true**

**tags = {**

**Name = "MyVPC"**

**terraform apply**

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



# **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.

# **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**

# **Clean Up:**

* + After testing, you can clean up the IAM users:

**terraform destroy**

* + Confirm the destruction by typing yes.

# **Conclusion:**

This lab exercise demonstrates creating 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:**

**mkdir terraform-vpc cd terraform-vpc**

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

**provider "aws" { region = "us-east-1"**

**}**

**resource "aws\_vpc" "my\_vpc" { cidr\_block = "10.0.0.0/16" enable\_dns\_support = true enable\_dns\_hostnames = 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 = "us-east-1a"**

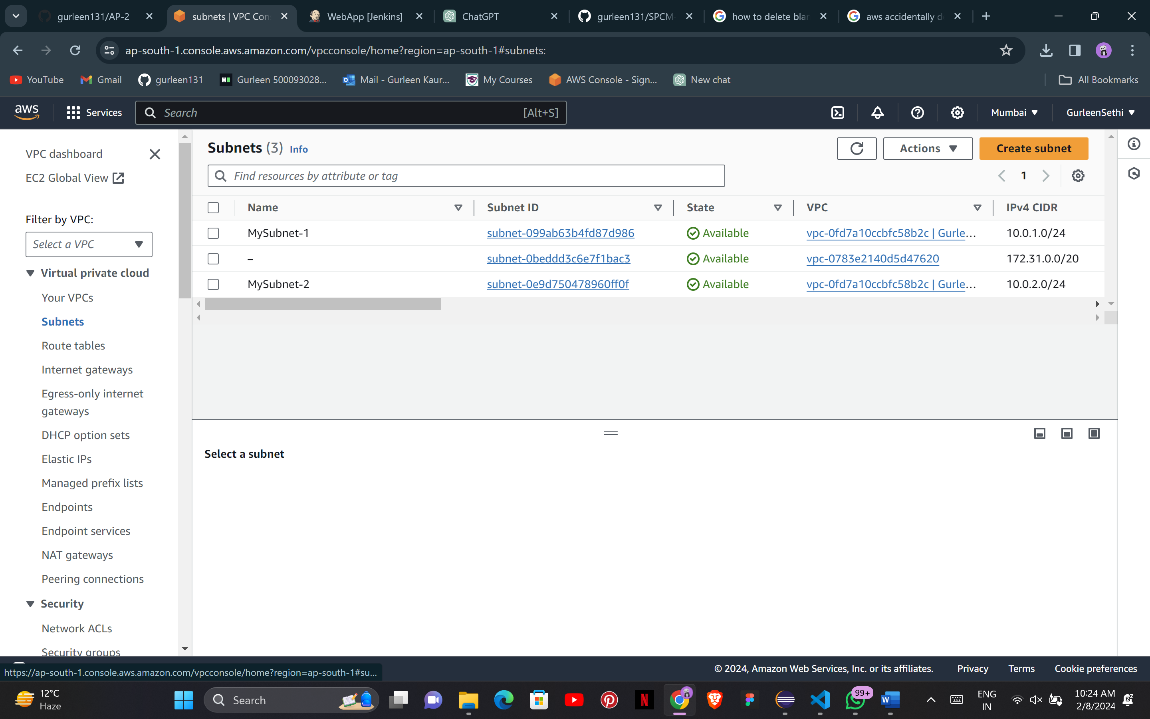
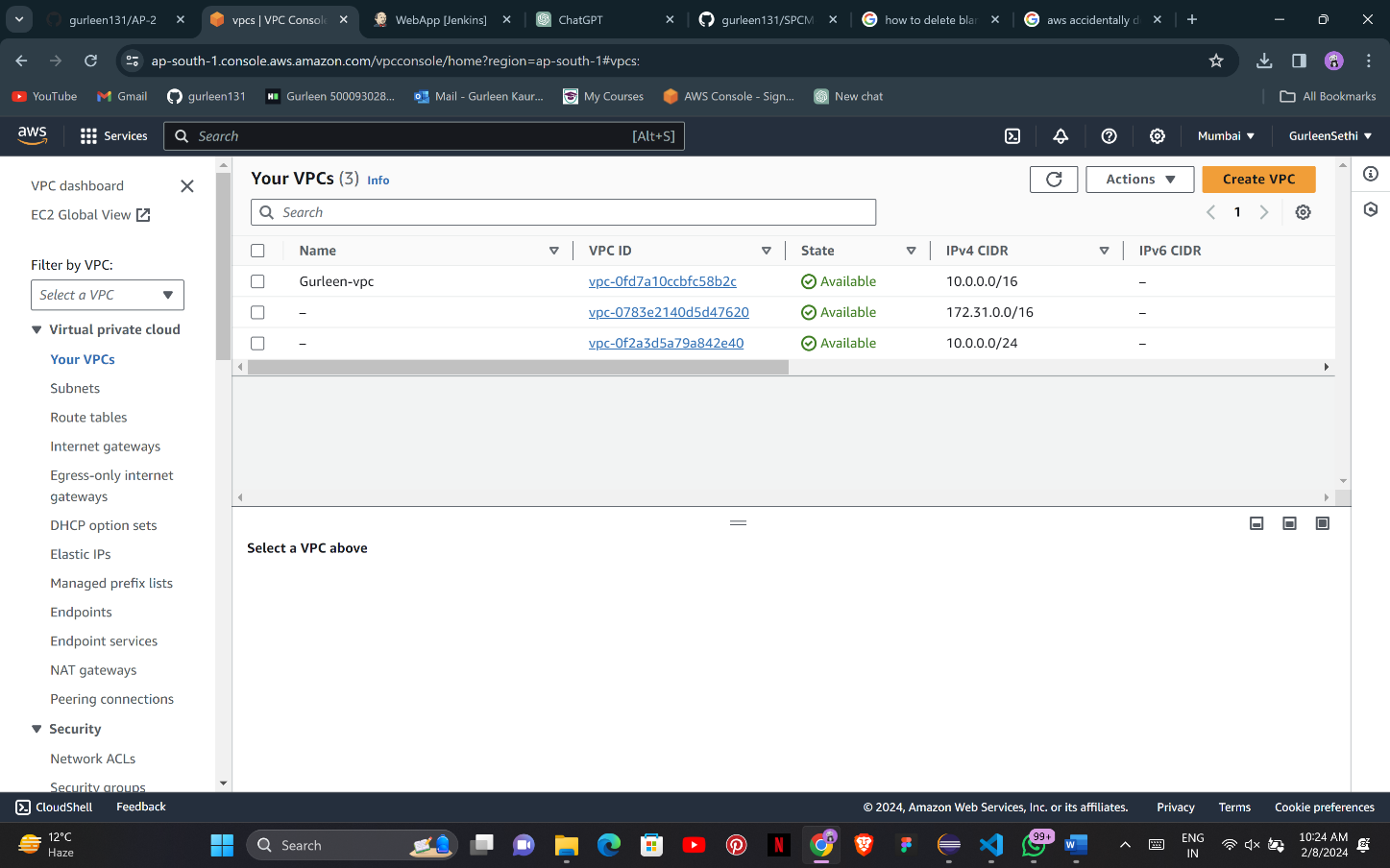
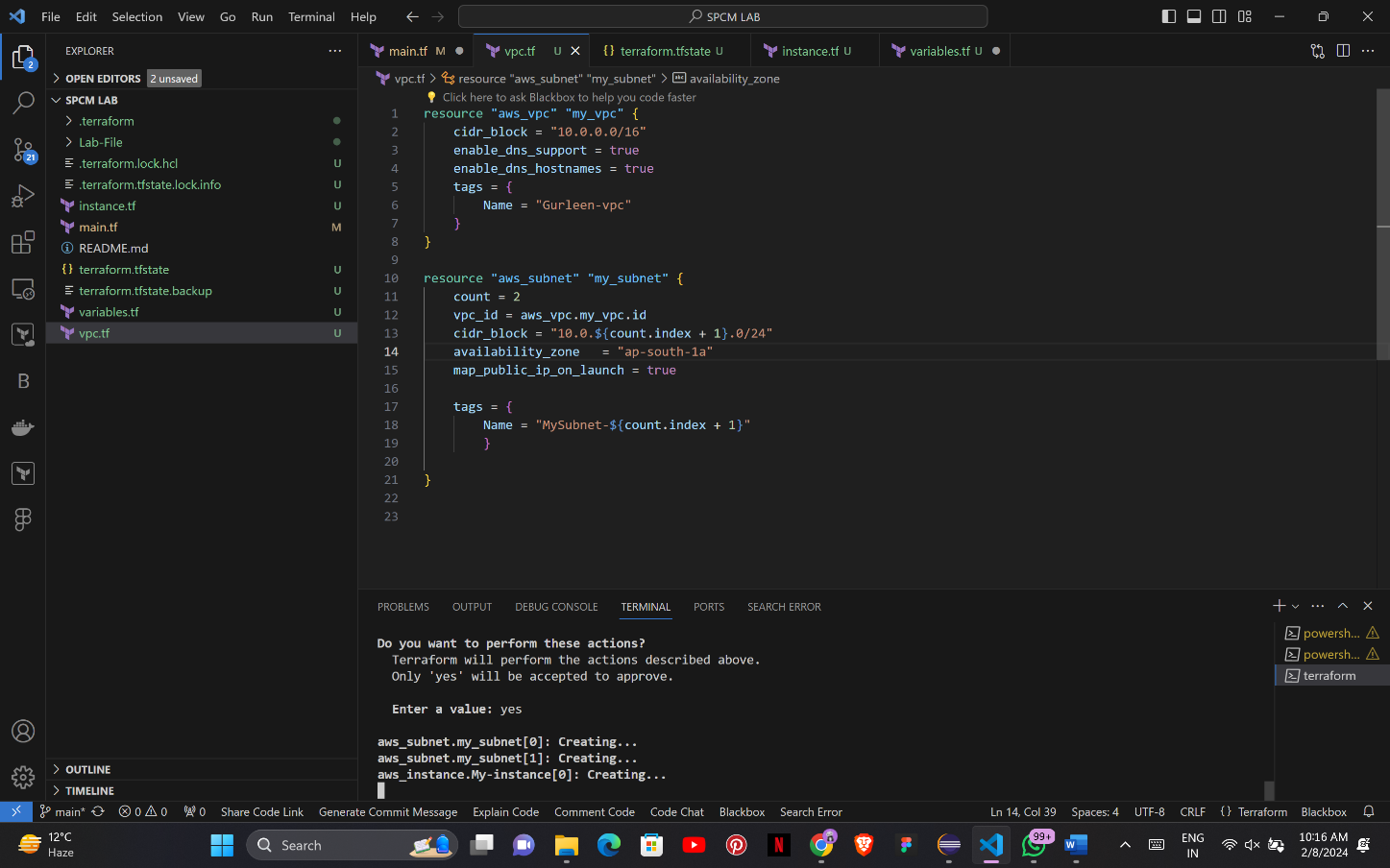
**map\_public\_ip\_on\_launch = true**

**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.

# **Initialize and Apply:**

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

**terraform init terraform apply**

* Terraform will prompt you to confirm the creation of the VPC and subnets. Type yes and press Enter.

# **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 created.

# **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:

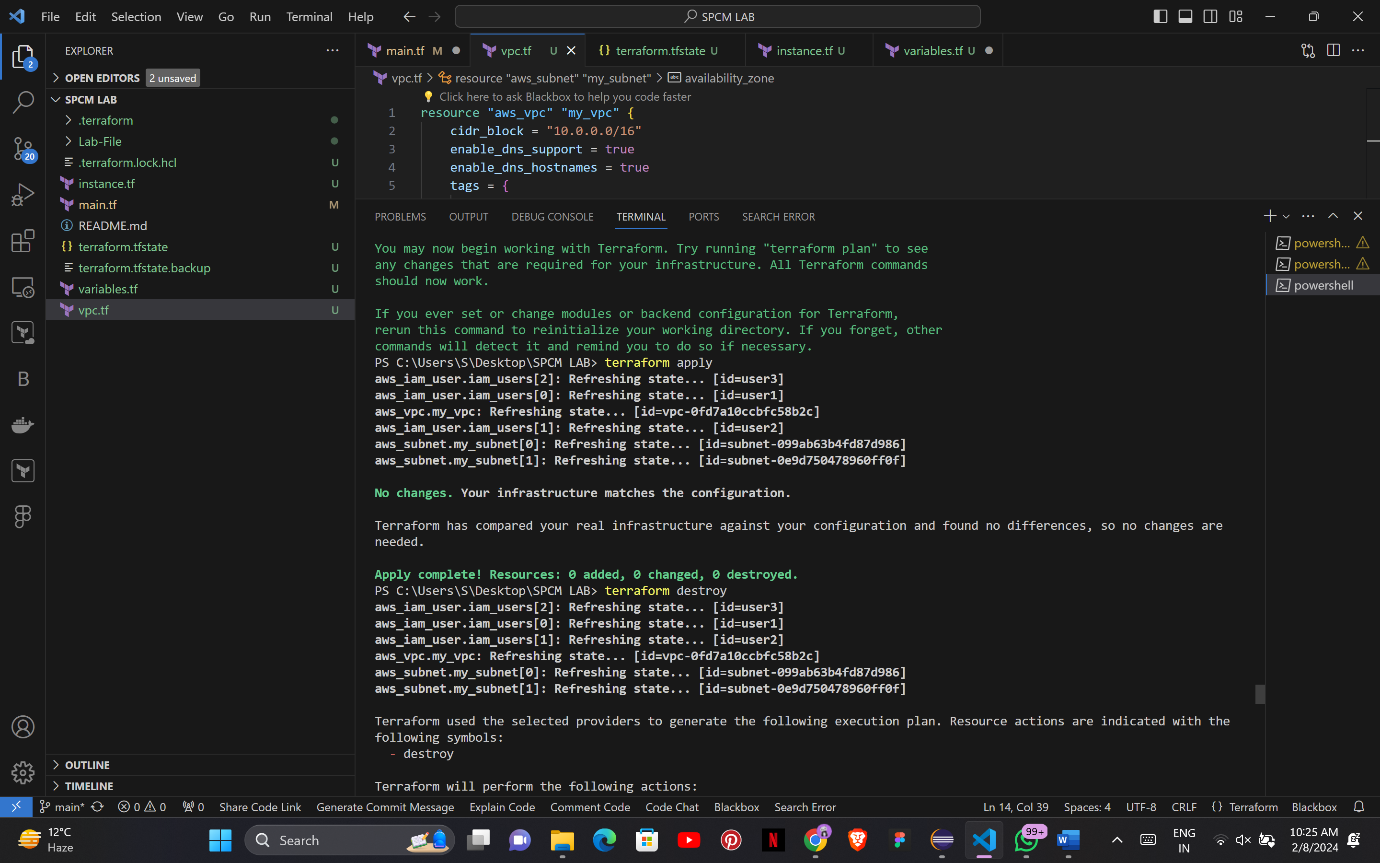
**terraform apply**

**Clean Up:**

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

**terraform destroy**

Confirm the destruction by typing yes.



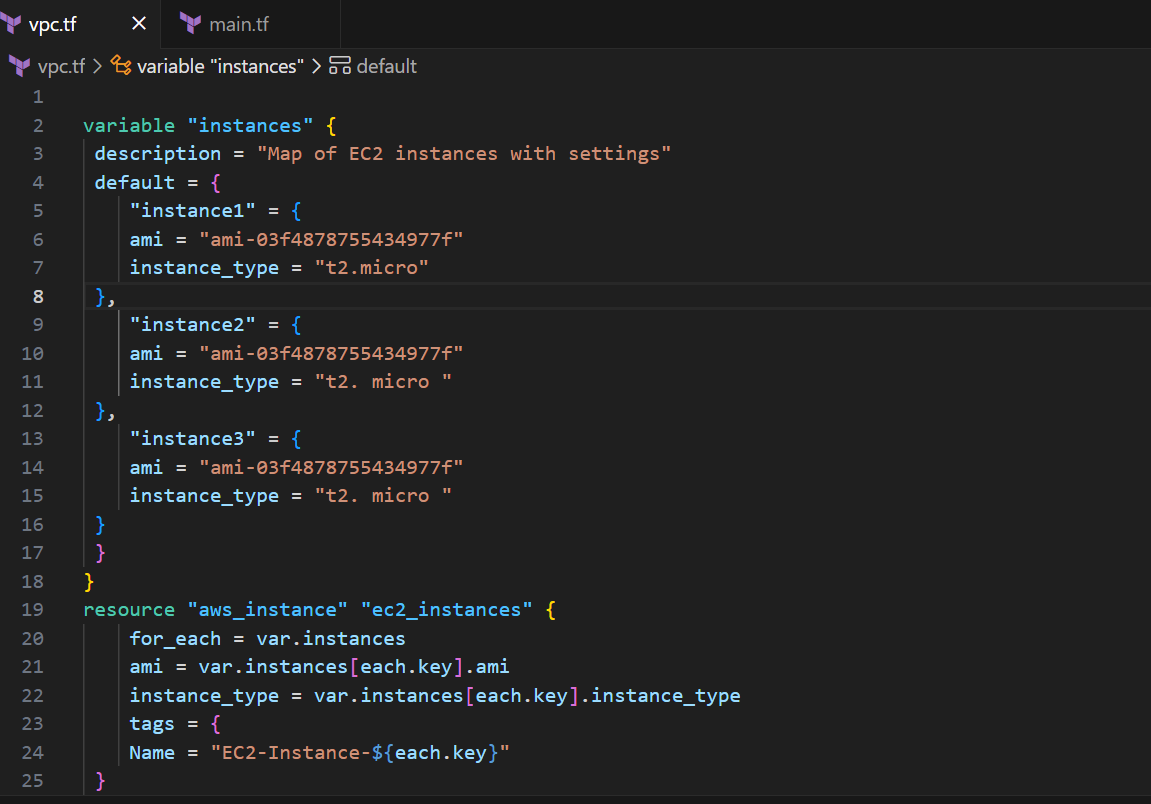
# **Conclusion:**

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.

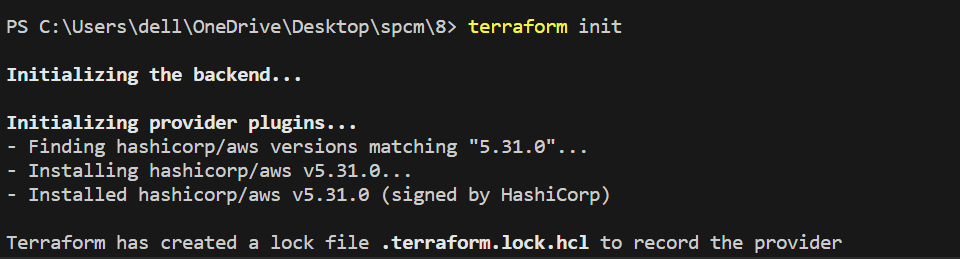
**EXPERIMENT -9**

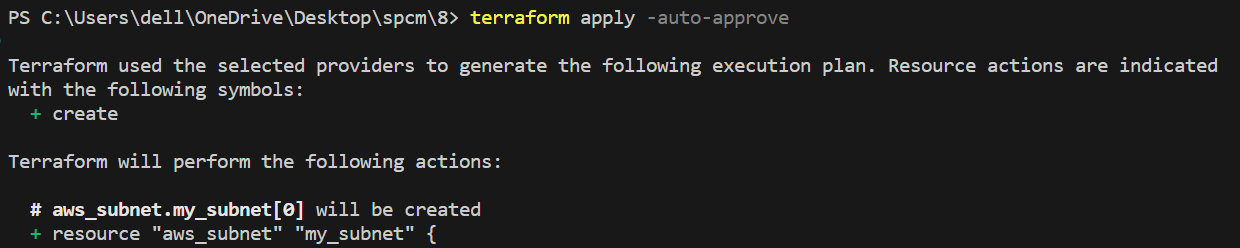
**Creating Multiple EC2 Instances with for each in Terraform**

**Creating the file**

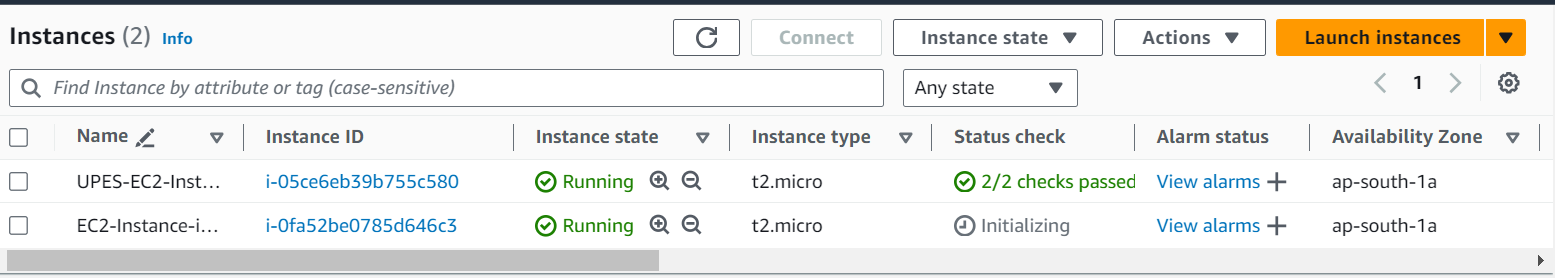
****

**INIT AND APPLY**

****

****

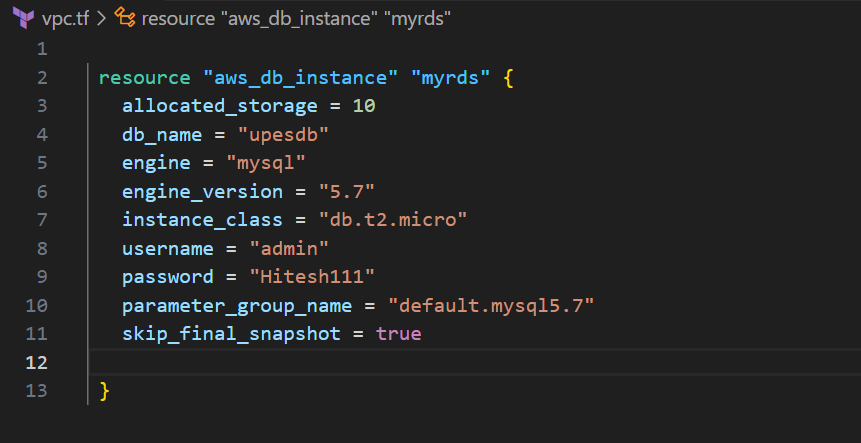
**RESULT**

****

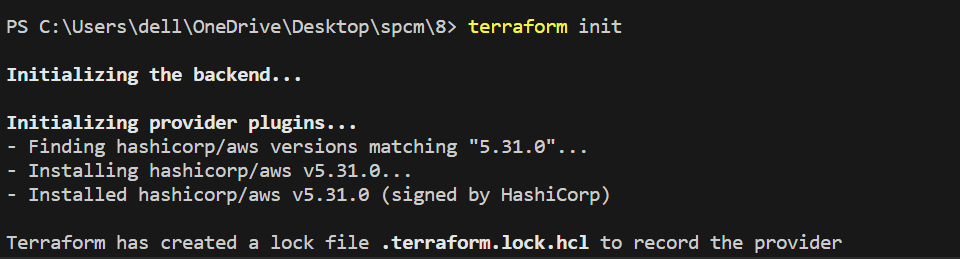
**EXPERIMENT 10**

Creating an AWS RDS Instance in Terraform

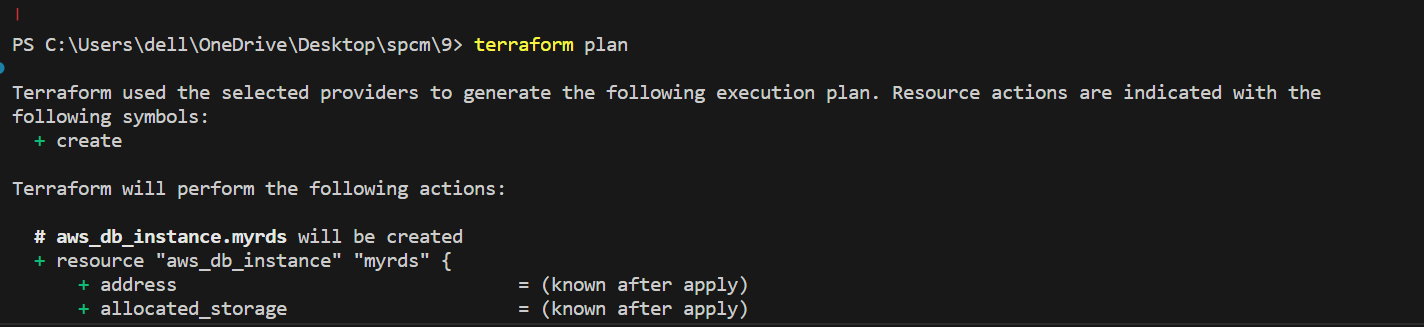
Creating the file

****

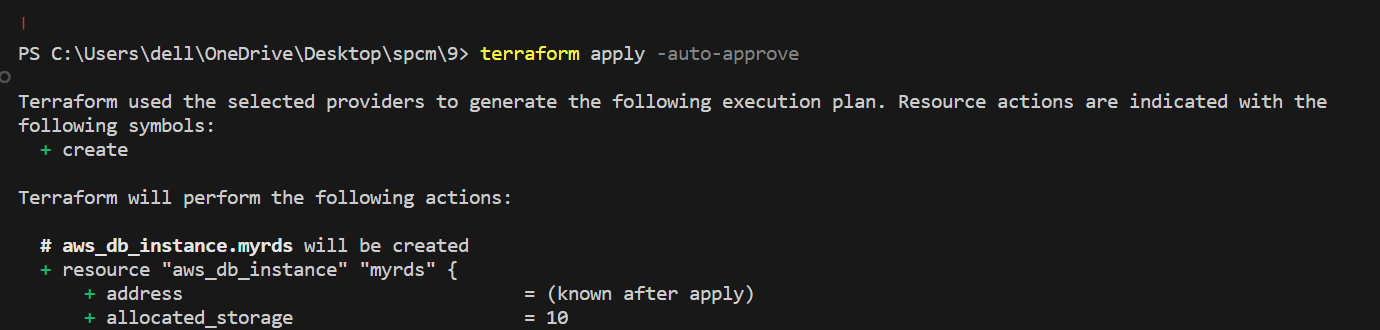
**Terraform init**

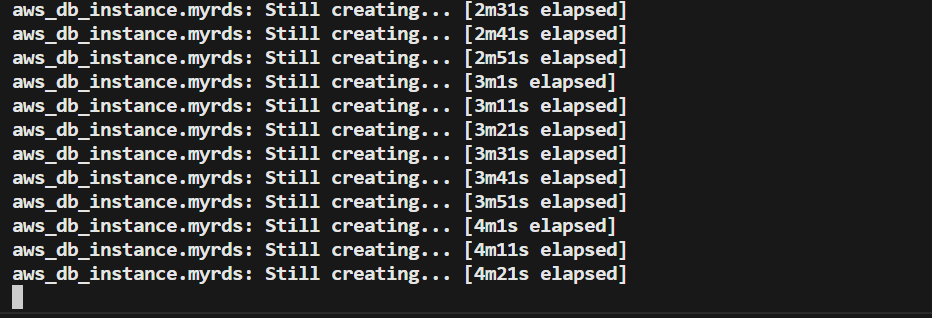
****

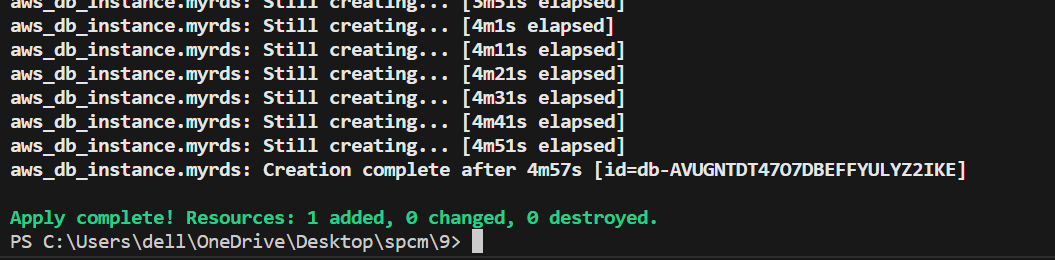
**Terraform plan**

****

**Terraform apply**

****

****

****