

Lesson 09 Demo 02

Running Multiple Cloud Provider Configurations

Objective: To demonstrate the use of multiple AWS provider configurations in Terraform for managing resources across different regions, showcasing the ability to define, configure, and deploy resources in various regions within a single Terraform configuration

Tools required: VS Code and Linux terminal

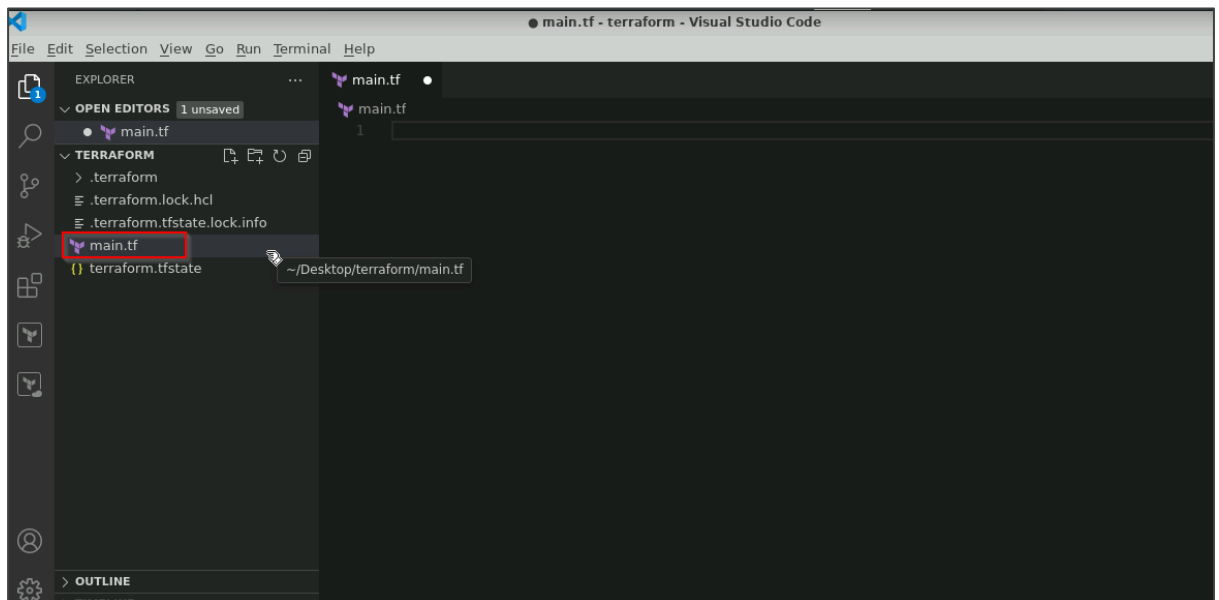
Prerequisites: None

Steps to be followed:

1. Define providers with aliases in the **main.tf** file
2. Specify provider configurations for each resource in the **main.tf** file
3. Initialize and apply the configuration

Step 1: Define providers with aliases in the main.tf file

1.1 Open the **main.tf** file in the **Terraform** folder



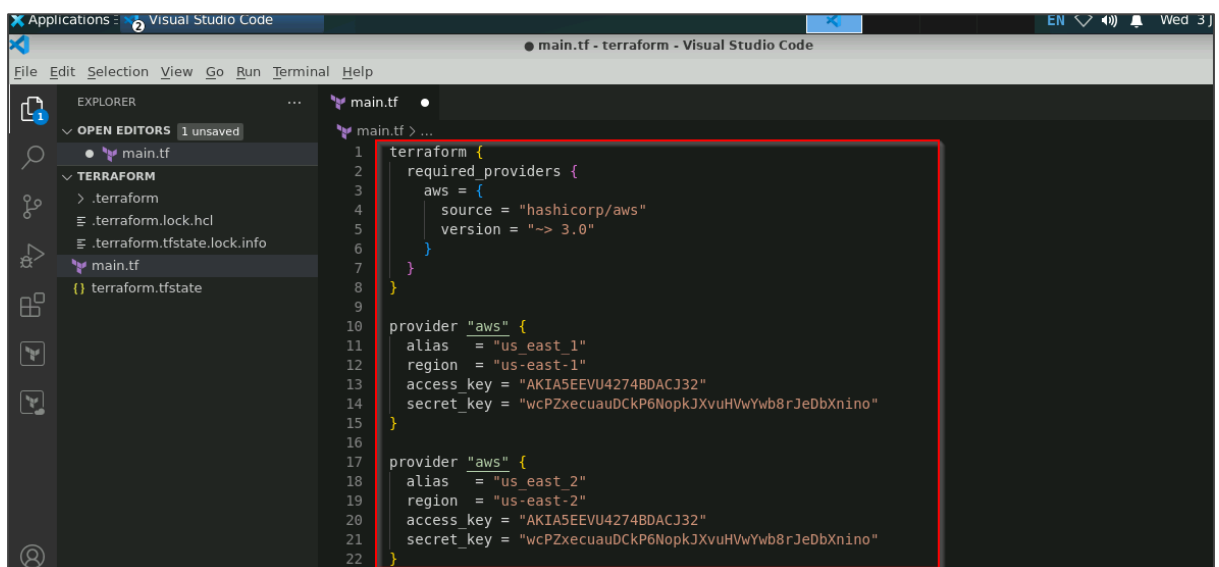
1.2 Enter the code given below in the **main.tf** file to define multiple AWS providers with different aliases and save the file:

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

provider "aws" {
  alias   = "us_east_1"
  region = "us-east-1"
  access_key = "AKIA5EEVU4274BDACJ32"
  secret_key = "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"
}

provider "aws" {
  alias   = "us_east_2"
  region = "us-east-2"
  access_key = "AKIA5EEVU4274BDACJ32"
  secret_key = "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"
}

provider "aws" {
  alias   = "eu_central_1"
  region = "eu-central-1"
  access_key = "AKIA5EEVU4274BDACJ32"
  secret_key = "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"
}
```



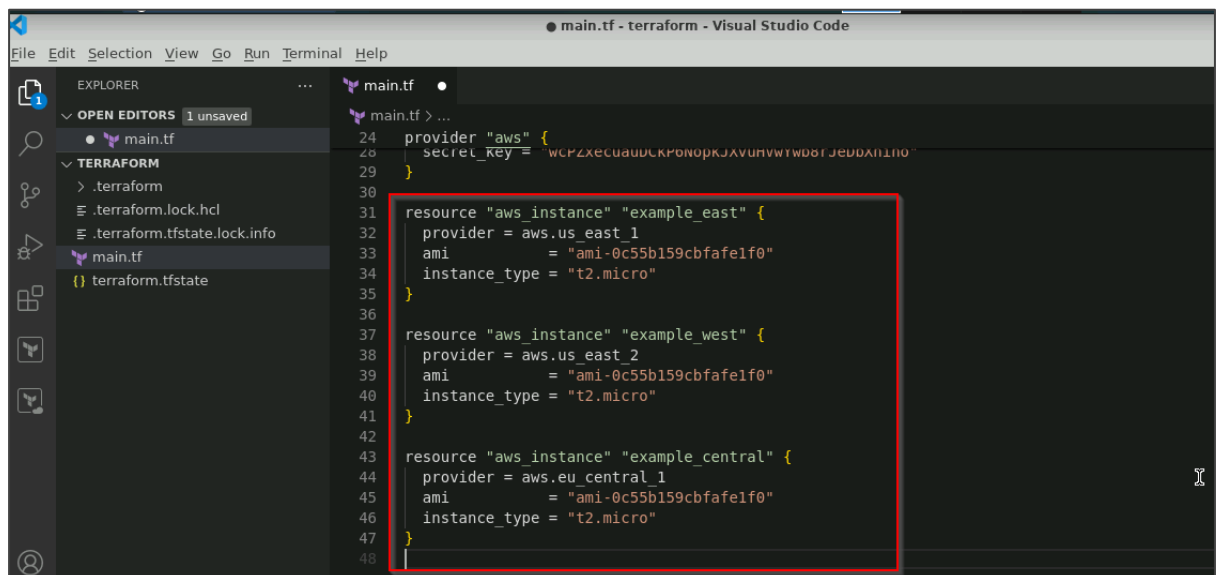
Step 2: Specify provider configurations for each resource in the main.tf file

2.1 Update the **main.tf** file with the provider configurations for each resource as given below and save the file:

```
resource "aws_instance" "example_east" {  
  provider = aws.us_east_1  
  ami      = "ami-0c55b159cbfafa1f0"  
  instance_type = "t2.micro"  
}
```

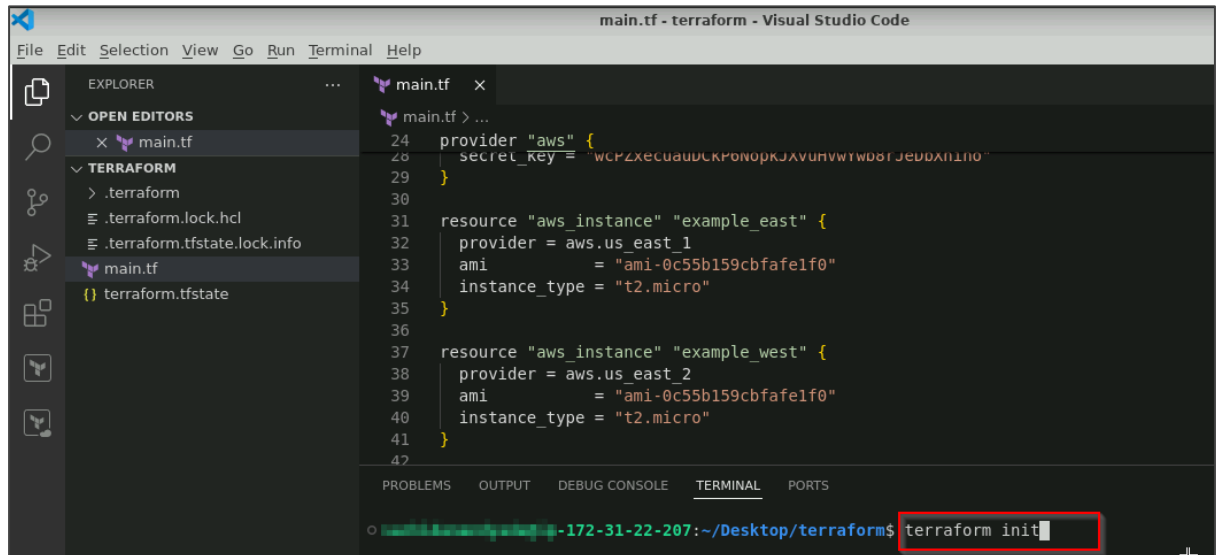
```
resource "aws_instance" "example_west" {  
  provider = aws.us_east_2  
  ami      = "ami-0c55b159cbfafa1f0"  
  instance_type = "t2.micro"  
}
```

```
resource "aws_instance" "example_central" {  
  provider = aws.eu_central_1  
  ami      = "ami-0c55b159cbfafa1f0"  
  instance_type = "t2.micro"  
}
```



Step 3: Initialize and apply the configuration

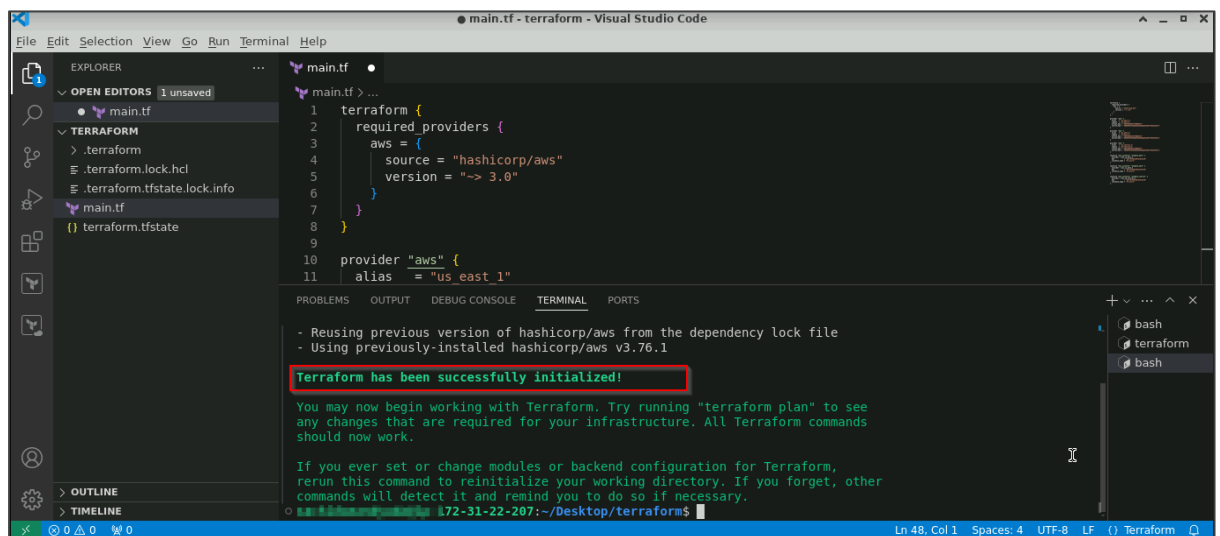
3.1 Open the terminal and run the command given below to initialize the configurations:
terraform init



The screenshot shows the Visual Studio Code interface with the `main.tf` file open. The file contains Terraform configuration for two AWS instances. The terminal at the bottom shows the command `terraform init` being entered.

```
main.tf > ...
24 provider "aws" {
25   secret_key = "wcpZxecuaudCKPbnOPKJxvuhVvTWd8rJEDDxh1no"
26 }
27
28 resource "aws_instance" "example_east" {
29   provider = aws.us_east_1
30   ami      = "ami-0c55b159cbfaffe1f0"
31   instance_type = "t2.micro"
32 }
33
34 resource "aws_instance" "example_west" {
35   provider = aws.us_east_2
36   ami      = "ami-0c55b159cbfaffe1f0"
37   instance_type = "t2.micro"
38 }
39
40 }
```

Terminal: `terraform init`



The screenshot shows the Visual Studio Code interface after running `terraform init`. The terminal displays the following output:

```
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v3.76.1

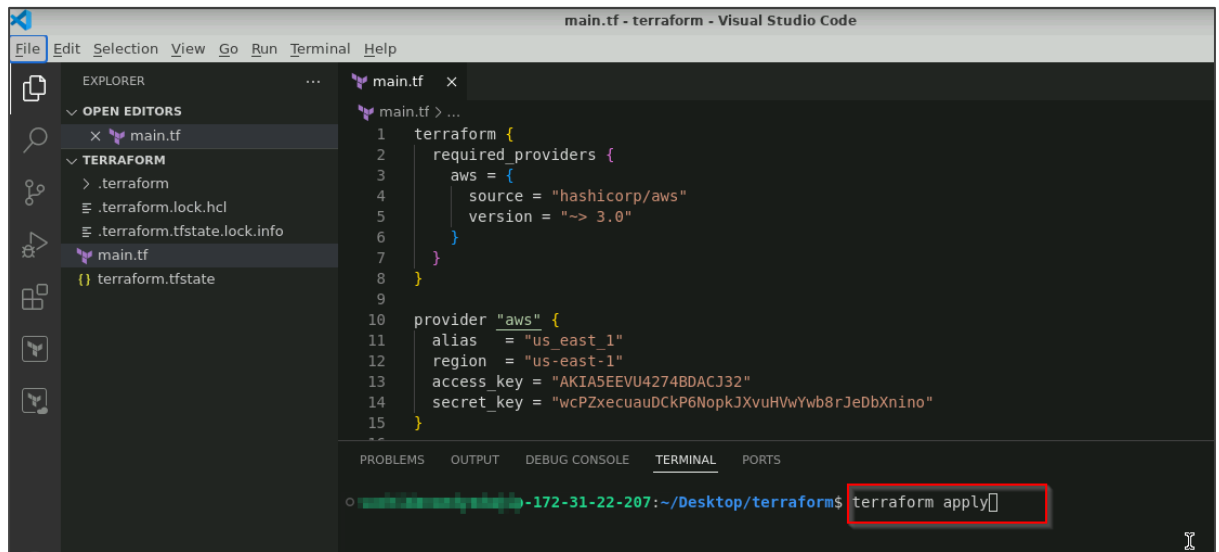
Terraform has been successfully initialized!

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.
```

The Terraform configuration is successfully initialized.

3.2 Open the terminal and run the command given below to apply the configurations:
terraform apply

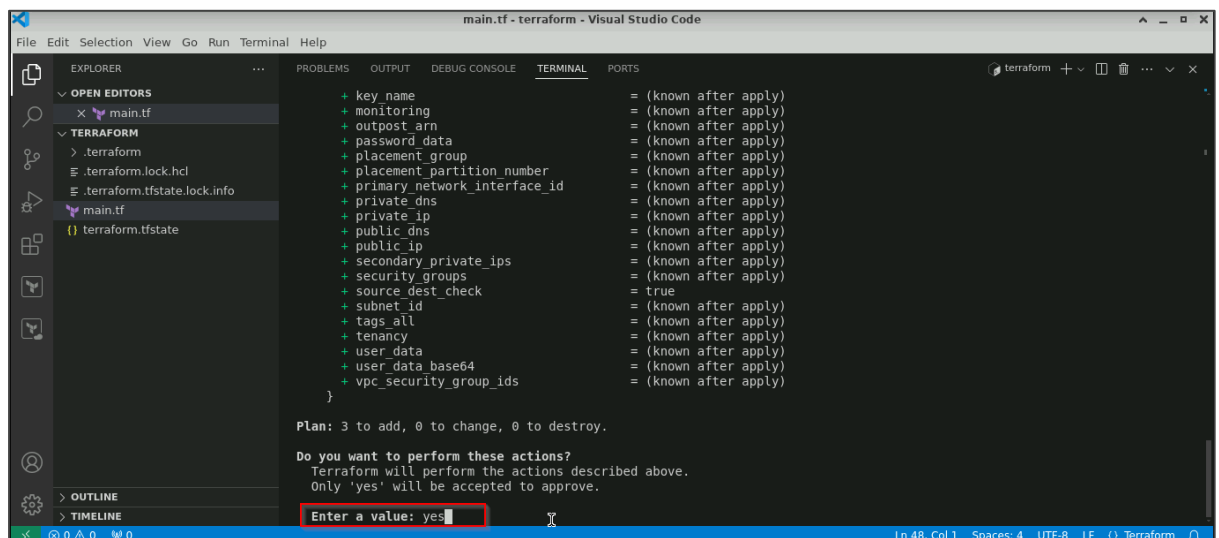


The screenshot shows the Visual Studio Code interface with the file explorer on the left displaying the project structure. The main editor shows the `main.tf` file with the following content:

```
1 terraform {
2   required_providers {
3     aws = {
4       source = "hashicorp/aws"
5       version = "~> 3.0"
6     }
7   }
8 }
9
10 provider "aws" {
11   alias     = "us_east_1"
12   region    = "us-east-1"
13   access_key = "AKIA5EEVU4274BDACJ32"
14   secret_key = "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"
15 }
```

The terminal window at the bottom shows the command `terraform apply` being entered at the prompt `-172-31-22-207:~/Desktop/terraform$`.

3.3 Enter **yes** to apply the configuration



The screenshot shows the Visual Studio Code interface with the terminal window displaying the output of the `terraform apply` command. The output lists the plan for the configuration, showing that 3 resources will be added. The plan is as follows:

```
+ key_name = (known after apply)
+ monitoring = (known after apply)
+ outpost_arn = (known after apply)
+ password_data = (known after apply)
+ placement_group = (known after apply)
+ placement_partition_number = (known after apply)
+ primary_network_interface_id = (known after apply)
+ private_dns = (known after apply)
+ private_ip = (known after apply)
+ public_dns = (known after apply)
+ public_ip = (known after apply)
+ secondary_private_ips = (known after apply)
+ security_groups = (known after apply)
+ source_dest_check = true
+ subnet_id = (known after apply)
+ tags_all = (known after apply)
+ tenancy = (known after apply)
+ user_data = (known after apply)
+ user_data_base64 = (known after apply)
+ vpc_security_group_ids = (known after apply)
```

The plan summary indicates: `Plan: 3 to add, 0 to change, 0 to destroy.`

The terminal then asks: `Do you want to perform these actions?` and `Terraform will perform the actions described above. Only 'yes' will be accepted to approve.`

The user has entered `yes` at the prompt `Enter a value: yes`.

The screenshot shows the Visual Studio Code interface with a Terraform configuration file named `main.tf` open. The configuration defines an AWS provider and several resources. The terminal window shows the output of the `terraform apply` command, indicating that the resources were successfully created.

```
main.tf - terraform - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
  OPEN EDITORS
    main.tf
  TERRAFORM
    .terraform
    .terraform.lock.hcl
    main.tf
    terraform.tfstate
    terraform.tfstate.backup

24 provider "aws" {
+   source_dest_check      = true
+   subnet_id              = (known after apply)
+   tags_all               = (known after apply)
+   tenancy                = (known after apply)
+   user_data              = (known after apply)
+   user_data_base64       = (known after apply)
+   vpc_security_group_ids = (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_instance.example_east: Creating...
aws_instance.example_west: Creating...
aws_instance.example_central: Creating...
aws_instance.example_west: Still creating... [10s elapsed]
aws_instance.example_west: Still creating... [20s elapsed]
aws_instance.example_west: Creation complete after 21s [id=i-014ab531929b844d6]

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

By following these steps, you have successfully demonstrated the use of multiple AWS provider configurations in Terraform for managing resources across different regions, showcasing the ability to define, configure, and deploy resources in various regions within a single Terraform configuration.