EXPERIMENT NO. 06 DATE –

Aim -

To build, change, and destroy AWS/GCP/Microsoft Azure/DigitalOcean Infrastructure Using Terraform.

Theory -

Terraform Block

The terraform {} block contains Terraform settings, including the required providers Terraform will use to provision your infrastructure. For each provider, the source attribute defines an optional hostname, a namespace, and the provider type. Terraform installs providers from the Terraform Registry by default. In this example configuration, the aws provider's source is defined as hashicorp/aws, which is shorthand for registry.terraform.io/hashicorp/aws.

Providers

The provider block configures the specified provider, in this case aws. A provider is a plugin that Terraform uses to create and manage your resources. The profile attribute in the aws provider block refers Terraform to the AWS credentials stored in your AWS configuration file, which you created when you configured the AWS CLI. Never hard-code credentials or other secrets in your Terraform configuration files. Like other types of code, you may share and manage your Terraform configuration files using source control, so hard-coding secret values can expose them to attackers.

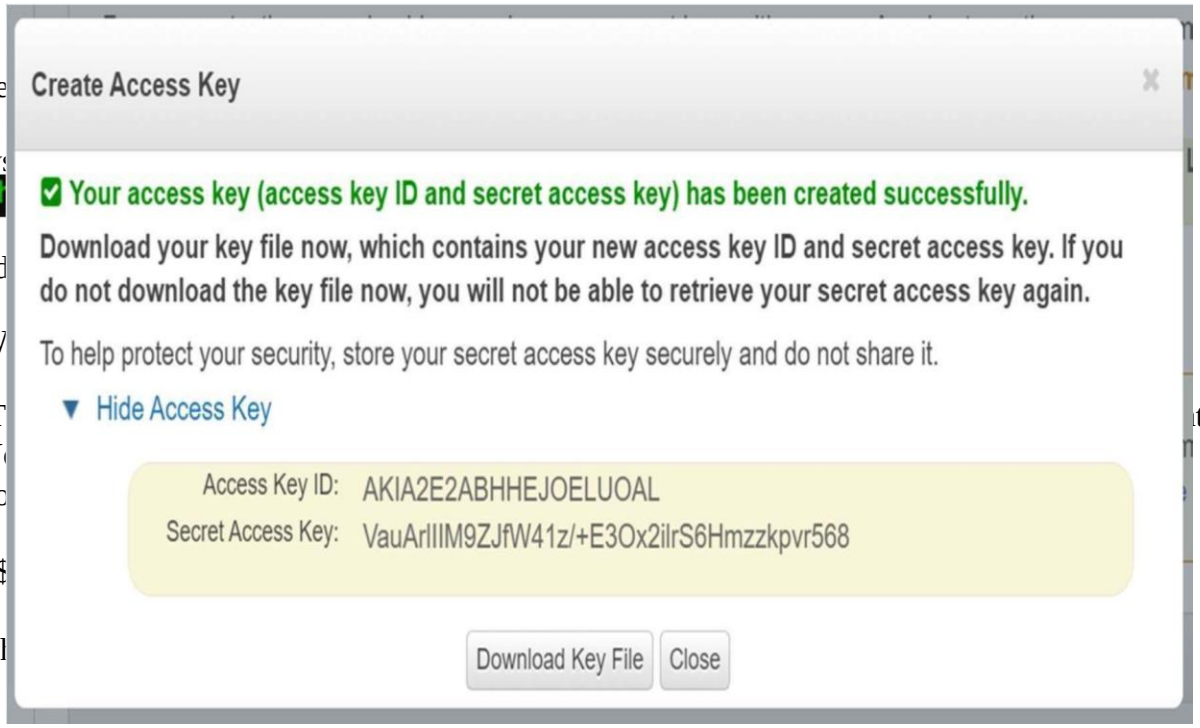
Resources

Use resource blocks to define components of your infrastructure. A resource might be a physical or virtual component such as an EC2 instance, or it can be a logical resource such as a Heroku application. Resource blocks have two strings before the block: the resource type and the resource name. In this example, the resource type is aws_instance and the name is app_server. The prefix of the type maps to the name of the provider. In the example configuration, Terraform manages the aws_instance resource with the aws provider. Together, the resource type and resource name form a unique ID for the resource. For example, the ID for your EC2 instance is aws_instance.app_server.

Resource blocks contain arguments which you use to configure the resource. Arguments can include things like machine sizes, disk image names, or VPC IDs. Our providers reference documents the required and optional arguments for each resource. For your EC2 instance, the example configuration sets the AMI ID to an Ubuntu image, and the instance type to t2.micro, which qualifies for AWS' free tier. It also sets a tag to give the instance a name. Prerequisites

- The Terraform CLI installed.
- The AWS CLI installed.
- Your AWS credentials. You can create a new Access Key on this page.

Copy the



$ aws

aacl

Build

W

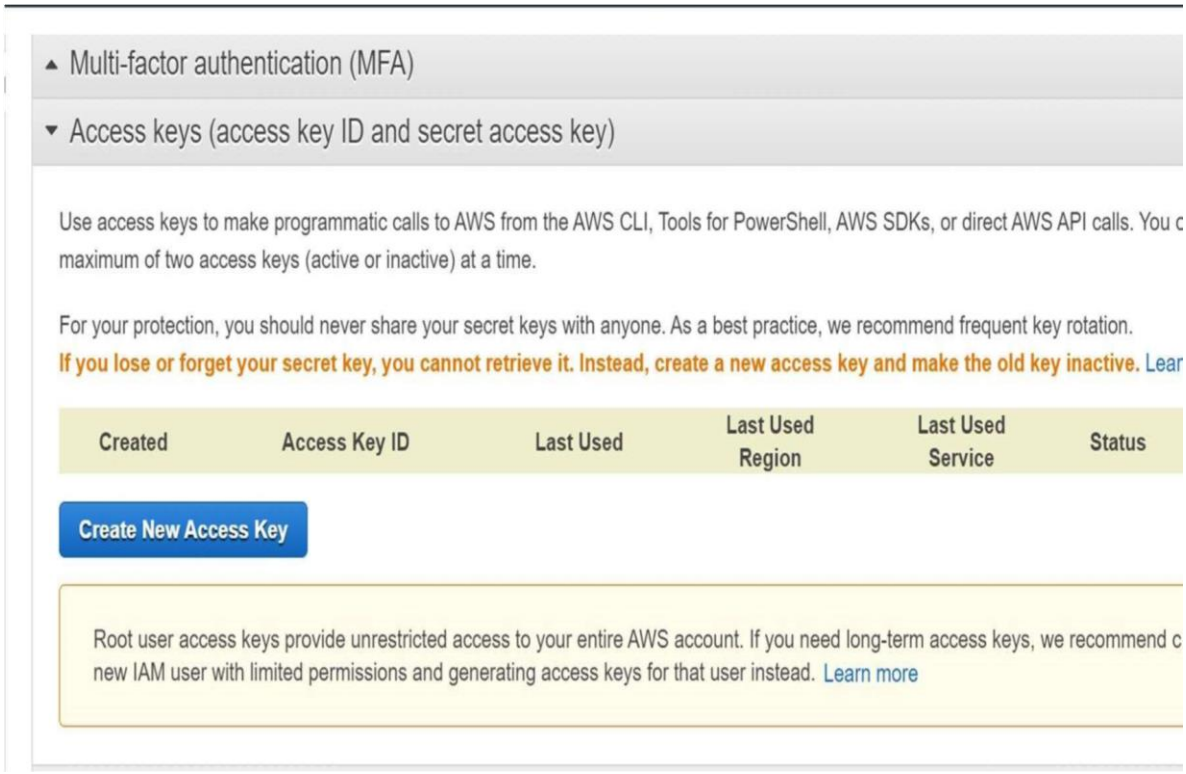
T tion.

Y

co

$

Cl



$ cd learn-terraform-aws-instance

Create a file to define your infrastructure.

$ touch main.tf

```
praj@it14-V530-15ICB:~$ mkdir learn-terraform-aws-instance
praj@it14-V530-15ICB:~$ cd learn-terraform-aws-instance
praj@it14-V530-15ICB:~/learn-terraform-aws-instance$ touch main.tf
praj@it14-V530-15ICB:~/learn-terraform-aws-instance$
```

Open AWS EC2 console, click on launch instance and copy the ami id of the EC2 instance you want to create. Open main.tf in your text editor, paste in the configuration below, and save the file.

main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.27"
    }
  }

  required_version = ">= 0.14.9"
}

provider "aws" {
  profile = "default"
  region  = "ap-south-1"
}

resource "aws_instance" "app_server" {
  ami           = "ami-0c1a7f89451184c8b"
  instance_type = "t2.micro"

  tags = {
    Name = "VaibhavAppServerInstance"
  }
}

```

Initialize the directory

When you create a new configuration or check out an existing configuration from version control you need to initialize the directory with terraform init. Initializing a configuration directory downloads and installs the providers defined in the configuration, which in this case is the aws provider.Initialize the directory.

```
praj@it14-V530-15ICB:~/learn-terraform-aws-instance$ touch main.tf
praj@it14-V530-15ICB:~/learn-terraform-aws-instance$ terraform init
```

```
Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 3.27"...
- Installing hashicorp/aws v3.57.0...
- Installed hashicorp/aws v3.57.0 (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!

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.
```

Create infrastructure

Apply the configuration now with the terraform apply command. Terraform will print output similar

```
praj@it14-V530-15ICB:~/learn-terraform-aws-instance$ terraform apply
```

to what is shown below.

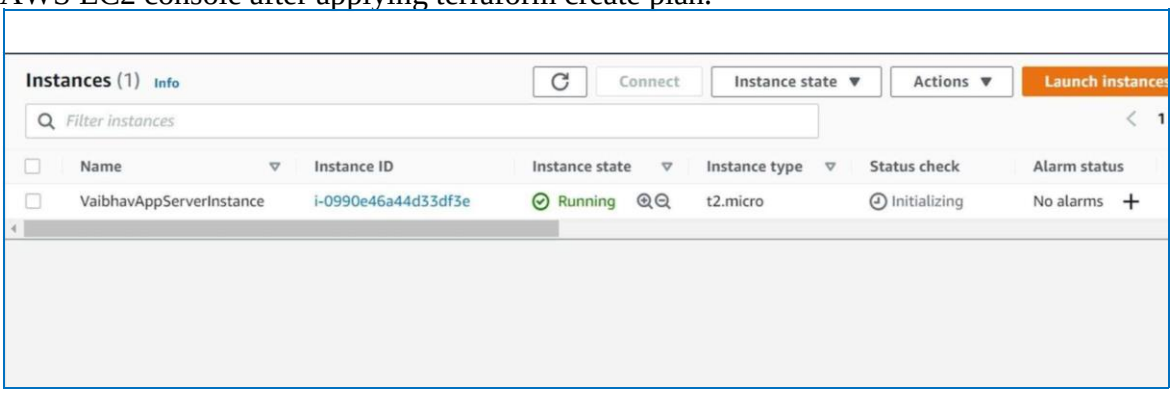
```
Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.app_server will be created
  + resource "aws_instance" "app_server" {
      + ami                                  = "ami-0c1a7f89451184c8b"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
```

Terraform will now pause and wait for your approval before proceeding. If anything in the plan

seems incorrect or dangerous, it is safe to abort here with no changes made to your infrastructure. In this case the plan is acceptable, so type yes at the confirmation prompt to proceed. Executing the plan will take a few minutes since Terraform waits for the EC2 instance to become available.

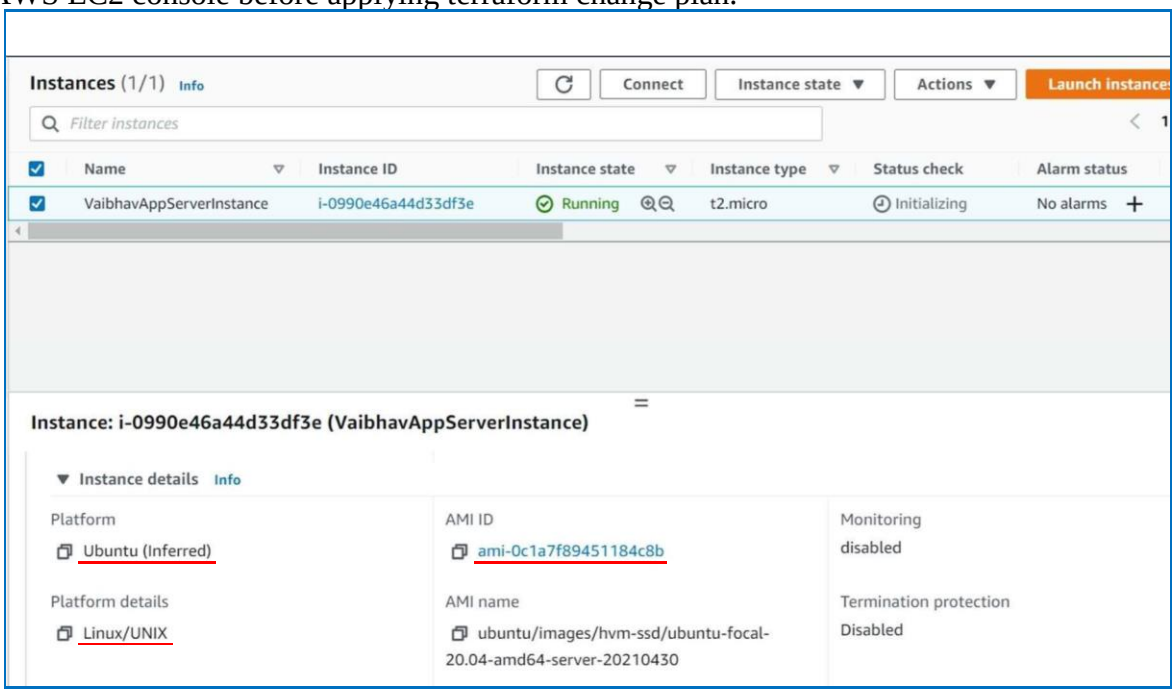
```
  Enter a value: yes

aws_instance.app_server: Creating...
aws_instance.app_server: Still creating... [10s elapsed]
aws_instance.app_server: Still creating... [20s elapsed]
aws_instance.app_server: Still creating... [30s elapsed]
aws_instance.app_server: Creation complete after 35s [id=i-0990e46a44d33df3e]

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

AWS EC2 console after applying terraform create plan.



Creation of AWS infrastructure has been completed. Next is changing the infrastructure.

AWS EC2 console before applying terraform change plan.



Configuration

Now update the ami of your instance. Edit the main.tf by replacing the current AMI ID with a new one.

main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.27"
    }
  }

  required_version = ">= 0.14.9"
}

provider "aws" {
  profile = "default"
  region  = "ap-south-1"
}

resource "aws_instance" "app_server" {
  ami           = "ami-0a23ccb2cdd9286bb"
  instance_type = "t2.micro"

  tags = {
    Name = "VaibhavAppServerInstance"
  }
}

```

Apply Changes

After changing the configuration, run terraform apply again to see how Terraform will apply this

```
praj@it14-V530-15ICB:~/learn-terraform-aws-instance$ terraform apply
```

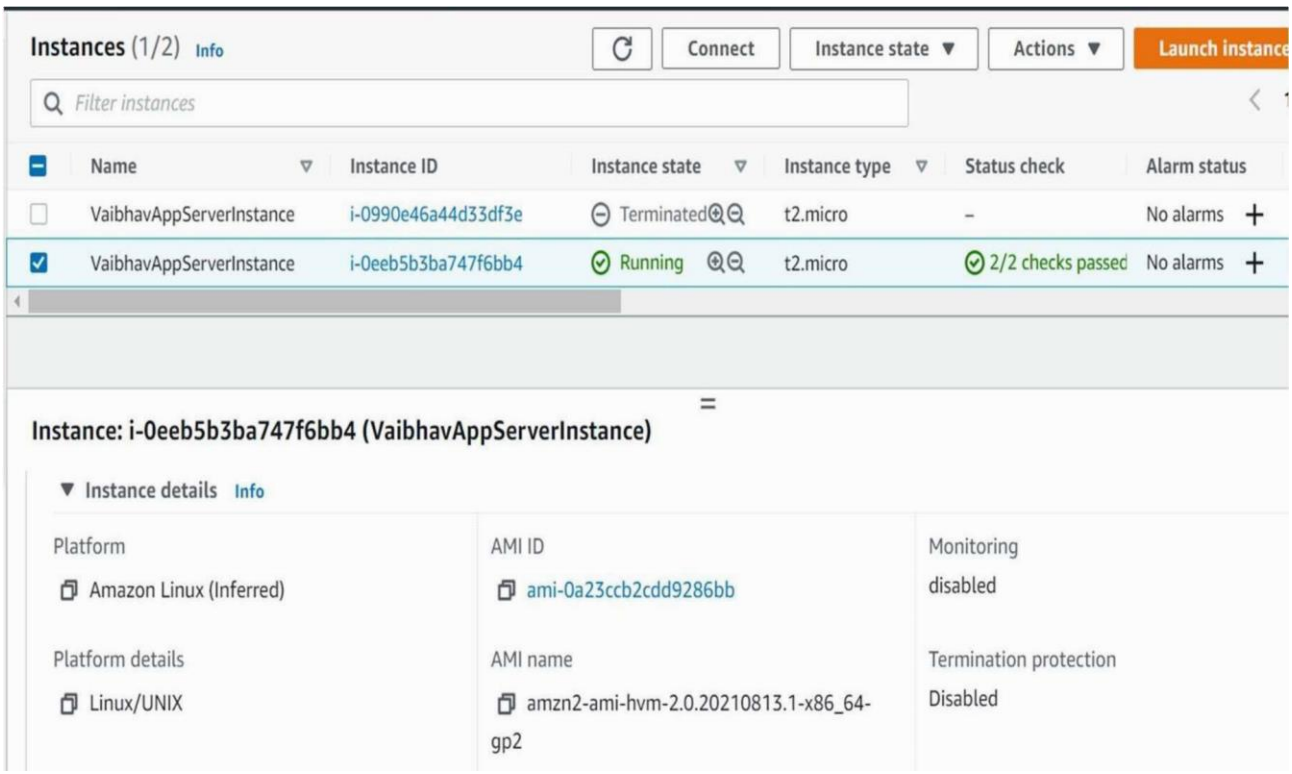
change to the existing resources.

```
aws_instance.app_server: Refreshing state... [id=i-0990e46a44d33df3e]

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_instance.app_server must be replaced
-/+ resource "aws_instance" "app_server" {
      ~ ami                                  = "ami-0c1a7f89451184c8b" -> "ami-0a23ccb2cdd9286
bb" # forces replacement
      ~ arn                                  = "arn:aws:ec2:ap-south-1:697529612744:instance/i
-0990e46a44d33df3e" -> (known after apply)
      ~ associate_public_ip_address          = true -> (known after apply)
      ~ availability_zone                    = "ap-south-1a" -> (known after apply)
      ~ cpu_core_count                       = 1 -> (known after apply)
      ~ cpu_threads_per_core                 = 1 -> (known after apply)
      ~ disable_api_termination              = false -> (known after apply)
      ~ ebs_optimized                        = false -> (known after apply)
      - hibernation                          = false -> null
      + host_id                              = (known after apply)
      ~ id                                   = "i-0990e46a44d33df3e" -> (known after apply)
      ~ instance_initiated_shutdown_behavior = "stop" -> (known after apply)
      ~ instance_state                       = "running" -> (known after apply)
```

The execution plan shows that the AMI change is what forces Terraform to replace the instance. Using this information, you can adjust your changes to avoid destructive updates if necessary. Once again, Terraform prompts for approval of the execution plan before proceeding. Answer yes to execute the planned steps.

```
  Enter a value: yes

aws_instance.app_server: Destroying... [id=i-0990e46a44d33df3e]
aws_instance.app_server: Still destroying... [id=i-0990e46a44d33df3e, 10s elapsed]
aws_instance.app_server: Still destroying... [id=i-0990e46a44d33df3e, 20s elapsed]
aws_instance.app_server: Still destroying... [id=i-0990e46a44d33df3e, 30s elapsed]
aws_instance.app_server: Still destroying... [id=i-0990e46a44d33df3e, 40s elapsed]
aws_instance.app_server: Still destroying... [id=i-0990e46a44d33df3e, 50s elapsed]
aws_instance.app_server: Still destroying... [id=i-0990e46a44d33df3e, 1m0s elapsed]
aws_instance.app_server: Still destroying... [id=i-0990e46a44d33df3e, 1m10s elapsed]
aws_instance.app_server: Still destroying... [id=i-0990e46a44d33df3e, 1m20s elapsed]
aws_instance.app_server: Still destroying... [id=i-0990e46a44d33df3e, 1m30s elapsed]
aws_instance.app_server: Still destroying... [id=i-0990e46a44d33df3e, 1m40s elapsed]
```

```
aws_instance.app_server: Still destroying... [id=i-0990e46a44d33df3e, 2m0s elapsed]
aws_instance.app_server: Still destroying... [id=i-0990e46a44d33df3e, 2m10s elapsed]
aws_instance.app_server: Still destroying... [id=i-0990e46a44d33df3e, 2m20s elapsed]
aws_instance.app_server: Destruction complete after 2m25s
aws_instance.app_server: Creating...
aws_instance.app_server: Still creating... [10s elapsed]
aws_instance.app_server: Still creating... [20s elapsed]
aws_instance.app_server: Still creating... [30s elapsed]
aws_instance.app_server: Still creating... [40s elapsed]
aws_instance.app_server: Creation complete after 45s [id=i-0eeb5b3ba747f6bb4]

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



Changing of AWS infrastructure has been successfully completed. Next is destroying/deleting the infrastructure.

Destroy Infrastructure

Once you no longer need infrastructure, you may want to destroy it to reduce your security exposure and costs. For example, you may remove a production environment from service, or manage shortlived environments like build or testing systems. In addition to building and modifying infrastructure, Terraform can destroy or recreate the infrastructure it manages.

Destroy

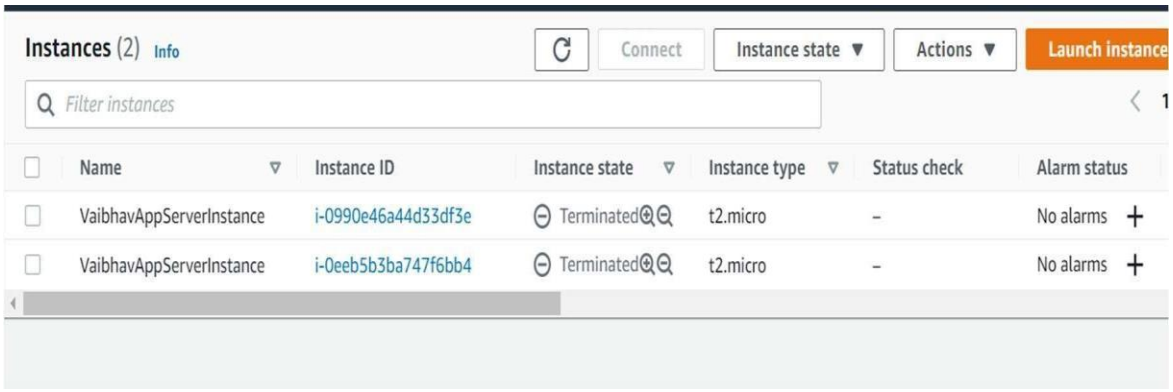
The terraform destroy command terminates resources managed by your Terraform project. This command is the inverse of terraform apply in that it terminates all the resources specified in your Terraform state. It does not destroy resources running elsewhere that are not managed by the current Terraform project.

Destroy the resources you create.

```
aws_instance.app_server: Refreshing state... [id=i-0eeb5b3ba747f6bb4]

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_instance.app_server will be destroyed
  - resource "aws_instance" "app_server" {
      - ami                                  = "ami-0a23ccb2cdd9286bb" -> null
      - arn                                  = "arn:aws:ec2:ap-south-1:697529612744:instance/i
-0eeb5b3ba747f6bb4" -> null
      - associate_public_ip_address          = true -> null
      - availability_zone                    = "ap-south-1a" -> null
      - cpu_core_count                       = 1 -> null
      - cpu_threads_per_core                 = 1 -> null
      - disable_api_termination              = false -> null
      - ebs_optimized                        = false -> null
      - get_password_data                    = false -> null
      - hibernation                          = false -> null
      - id                                   = "i-0eeb5b3ba747f6bb4" -> null
      - instance_initiated_shutdown_behavior = "stop" -> null
```

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

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_instance.app_server: Destroying... [id=i-0eeb5b3ba747f6bb4]
aws_instance.app_server: Still destroying... [id=i-0eeb5b3ba747f6bb4, 10s elapsed]
aws_instance.app_server: Still destroying... [id=i-0eeb5b3ba747f6bb4, 20s elapsed]
aws_instance.app_server: Still destroying... [id=i-0eeb5b3ba747f6bb4, 30s elapsed]
aws_instance.app_server: Still destroying... [id=i-0eeb5b3ba747f6bb4, 40s elapsed]
aws_instance.app_server: Destruction complete after 42s

Destroy complete! Resources: 1 destroyed.
```

AWS EC2 console after applying destroy terraform plan.



Conclusion -

Any company when wants their product to deployed, then choose the cloud platforms. The infrastructure design can be very huge and creating and maintaining such an infrastructure require lots of work and time. Terraform makes it easy to create, change and destroy such infrastructure very easily with human-readable and understandable script language and just running the script will make the project live to the world. Thus, we have successfully created, changed and destroyed the infrastructure using Terraform.