Advanced DevOps Lab
Experiment 6

Roll No.	24
Name	Iyer Sreekesh Subramanian
Class	D15-A
Subject	Advanced DevOps Lab

Aim: To understand terraform lifecycle, core concepts/terminologies and install it on a Linux Machine.

Theory:

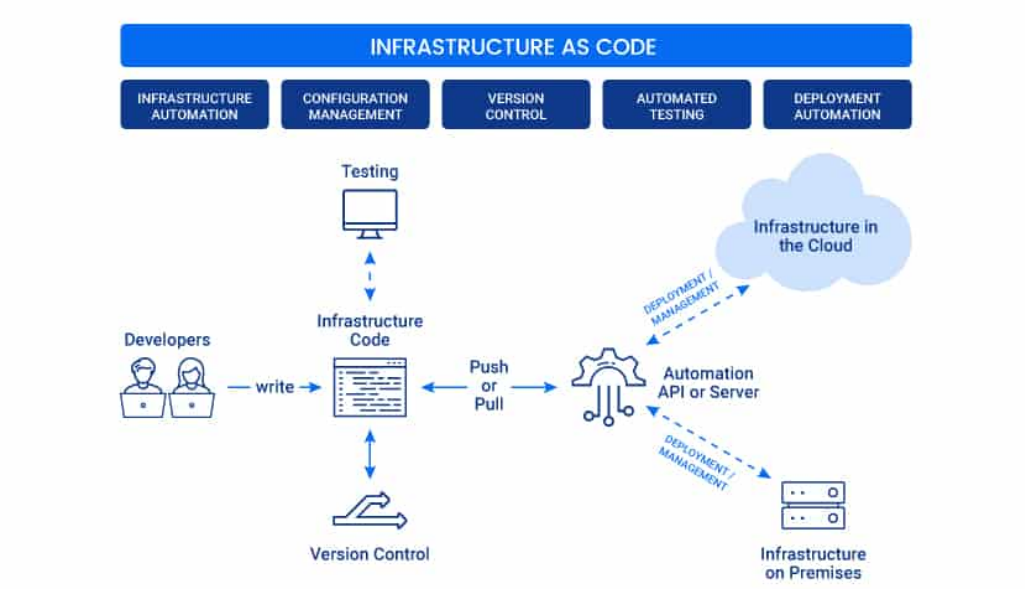
Terraform is an open-source “Infrastructure as Code” tool, created by HashiCorp.

A declarative coding tool, Terraform enables developers to use a high-level configuration language called HCL (HashiCorp Configuration Language) to describe the desired “end-state” cloud or on-premises infrastructure for running an application. It then generates a plan for reaching that end-state and executes the plan to provide the infrastructure.

Because Terraform uses simple syntax, can provision infrastructure across multiple cloud and on-premises data centres, and can safely and efficiently re-provision infrastructure in response to configuration changes, it is currently one of the most popular infrastructure automation tools available. If your organization plans to deploy a hybrid cloud or multi-cloud environment, you’ll likely want or need to get to know Terraform.

What is Infrastructure as Code?

Infrastructure as Code (IaC) is a widespread terminology among DevOps professionals and a key DevOps practice in the industry. It is the process of managing and provisioning the complete IT infrastructure (comprises both physical and virtual machines) using machine-readable definition files. It helps in automating the complete data centre by using programming scripts.



Terraform providers

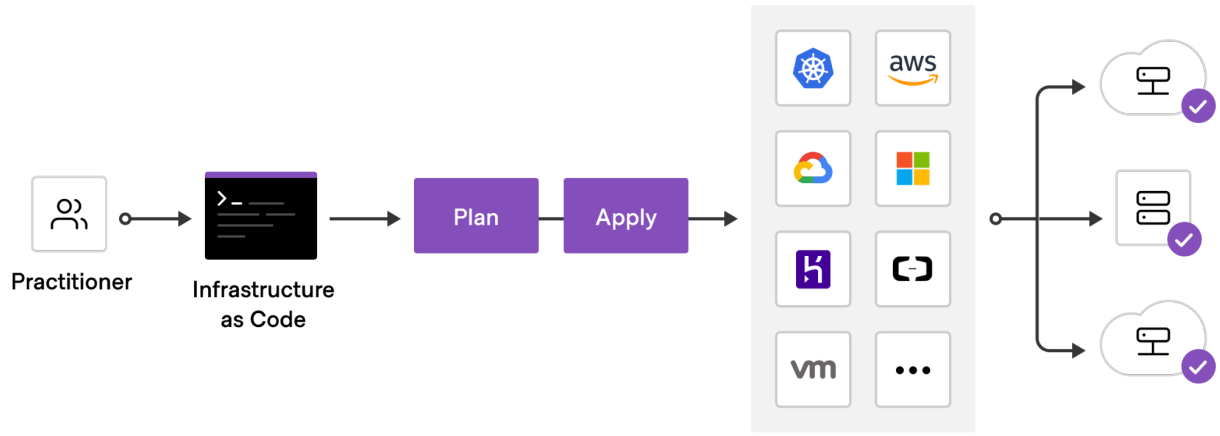
A provider is responsible for understanding API interactions and exposing resources. It is an executable plug-in that contains code necessary to interact with the API of the service. Terraform configurations must declare which providers they require so that Terraform can install and use them.

Terraform Plugins are responsible for defining resources for specific services. This includes authenticating infrastructure providers and initializing the libraries used to make API calls. Terraform Plugins are written in Go as executable binaries that can either be used as a specific service or as a provisioner. (Provisioner plugins are used to execute commands for a designated resource.)

Terraform has over a hundred providers for different technologies, and each provider then gives terraform user access to its resources. So through AWS provider, for example, you have access to hundreds of AWS resources like EC2 instances, the AWS users, etc.

Terraform Configuration Files

Configuration files are a set of files used to describe infrastructure in Terraform and have the file extensions .tf and .tf.json. Terraform uses a declarative model for defining infrastructure. Configuration files let you write a configuration that declares your desired state. Configuration files are made up of resources with settings and values representing the desired state of your infrastructure.

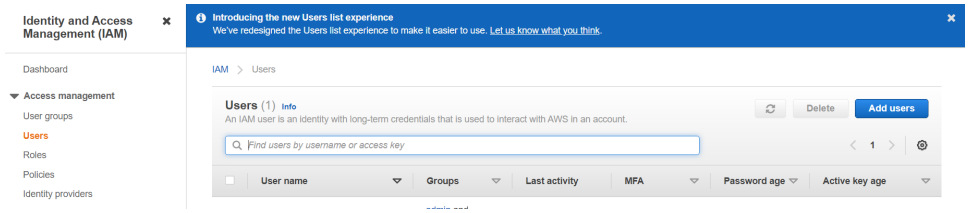


Creating an EC2 instance using Terraform

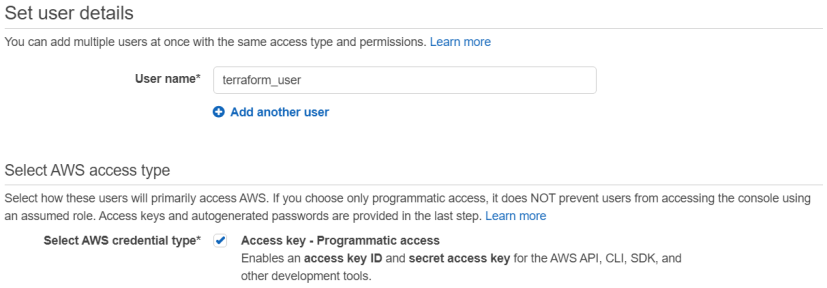
Steps:

1. Create a new IAM user in AWS.

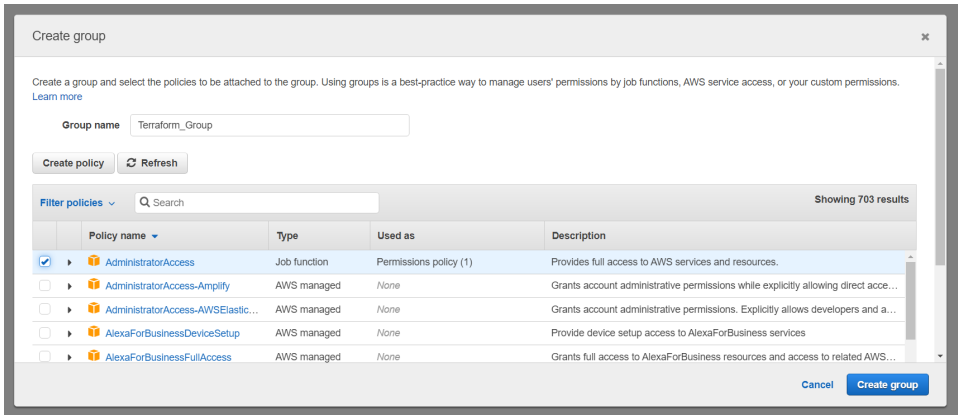
Open the IAM control panel and click on Add users.



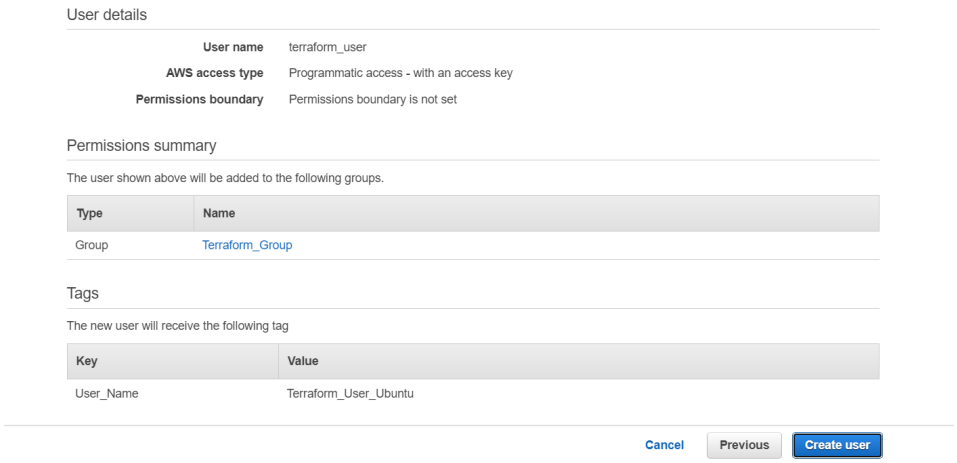
Enter a user name and choose Access Key credential type.



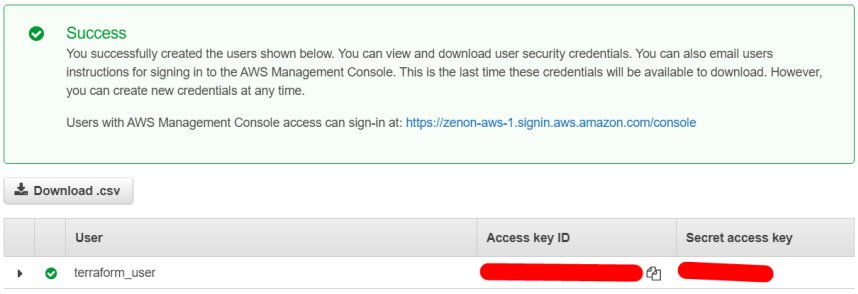
Create a new user group and provide Administrator Access.



Review changes and create the user.



Copy the Access Key ID and Secret Key and save them in a Notepad.

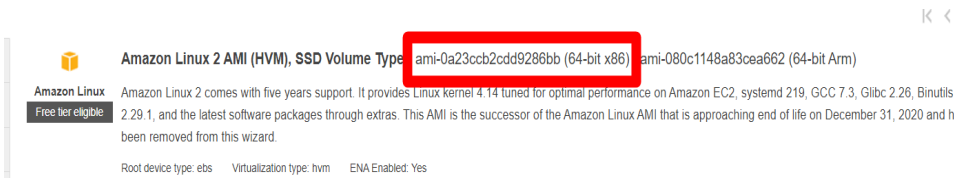


2. Create a new folder and create a new .tf file in which we'll have the script.

```
mkdir terraform_scripts
gedit ec2-on-terraform.tf
```

```
provider “aws” {
  access_key = “<Key>”
  secret_key = “<Key>”
}

resource “aws_instance” “terraform-ec2” {
  ami = “<ami-code>”
  instance_type=”t2.micro”
}
```

```
provider "aws" {
        access_key = "[redacted]"
        secret_key = "[redacted]"
        region = "ap-south-1"
}

resource "aws_instance" "terraform-ec2" {
        ami = "ami-0a23ccb2cdd9286bb"
        instance_type = "t2.micro"
}
```



3. Perform Terraform Commands to initialize backend and then apply your scripts to start an EC2 instance.

`terraform init`

```
zenon@yeetus:~/terraform_scripts$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v3.61.0...
- Installed hashicorp/aws v3.61.0 (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 plan`

```
zenon@yeetus:~/terraform_scripts$ terraform plan

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

Terraform will perform the following actions:

  # aws_instance.terraform-ec2 will be created
  + resource "aws_instance" "terraform-ec2" {
      + ami                                  = "ami-0a23ccb2cdd9286bb"
      + 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)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = (known after apply)
      + monitoring                           = (known after apply)
```

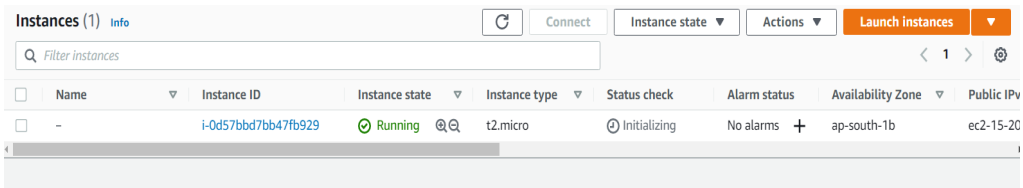
`terraform apply`

```
zenon@yeetus:~/terraform_scripts$ 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_instance.terraform-ec2 will be created
  + resource "aws_instance" "terraform-ec2" {
      + ami                                  = "ami-0a23ccb2cdd9286bb"
      + 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)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = (known after apply)
      + monitoring                           = (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (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)

      + capacity_reservation_specification {
          + capacity_reservation_preference = (known after apply)
```

```
Plan: 1 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.terraform-ec2: Creating...
aws_instance.terraform-ec2: Still creating... [10s elapsed]
aws_instance.terraform-ec2: Still creating... [20s elapsed]
aws_instance.terraform-ec2: Still creating... [30s elapsed]
aws_instance.terraform-ec2: Creation complete after 32s [id=i-0d57bbd7bb47fb929]

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

4. Check if your instance is running in the AWS EC2 console.



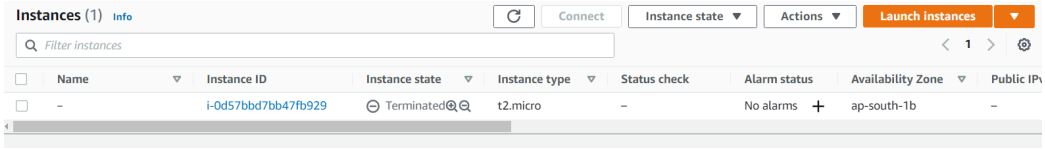
5. Use the destroy command to terminate the EC2 instance

```
terraform destroy
```

```
zenon@yeetus:~/terraform_scripts$ terraform destroy
aws_instance.terraform-ec2: Refreshing state... [id=i-0d57bbd7bb47fb929]

Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since the last "terraform apply":

  # aws_instance.terraform-ec2 has been changed
  ~ resource "aws_instance" "terraform-ec2" {
      id                                   = "i-0d57bbd7bb47fb929"
    + tags                                 = {}
      # (28 unchanged attributes hidden)

      # (5 unchanged blocks hidden)
    }

Unless you have made equivalent changes to your configuration, or ignored the relevant attributes using ignore_changes, th

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

Terraform will perform the following actions:

  # aws_instance.terraform-ec2 will be destroyed
  - resource "aws_instance" "terraform-ec2" {
      - ami                                  = "ami-0a23ccb2cdd9286bb" -> null
      - arn                                  = "arn:aws:ec2:ap-south-1:050055370753:instance/i-0d57bbd7bb47fb929" -> null
      - associate_public_ip_address          = true -> null
      - availability_zone                    = "ap-south-1b" -> null
```

```
aws_instance.terraform-ec2: Destroying... [id=i-0d57bbd7bb47fb929]
aws_instance.terraform-ec2: Still destroying... [id=i-0d57bbd7bb47fb929, 10s elapsed]
aws_instance.terraform-ec2: Still destroying... [id=i-0d57bbd7bb47fb929, 20s elapsed]
aws_instance.terraform-ec2: Destruction complete after 30s

Destroy complete! Resources: 1 destroyed.
zenon@yeetus:~/terraform_scripts$
```

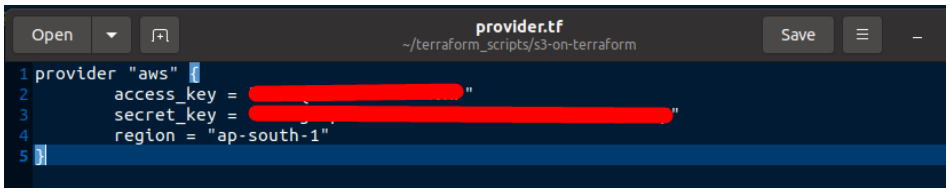
6. Check updates back in the Ec2 console. The instance should be terminated.



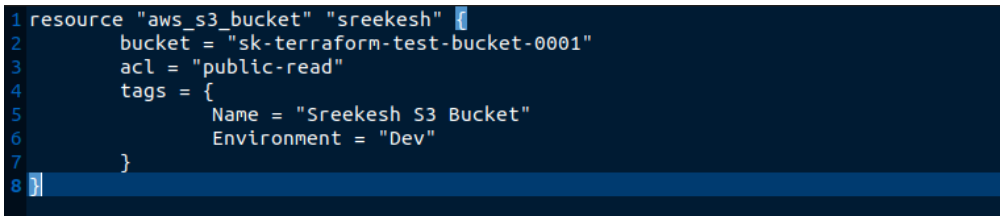
Creating an S3 Bucket using Terraform

1. Create a new directory and two .tf files, provider.tf and s3-on-terraform.tf

   provider.tf

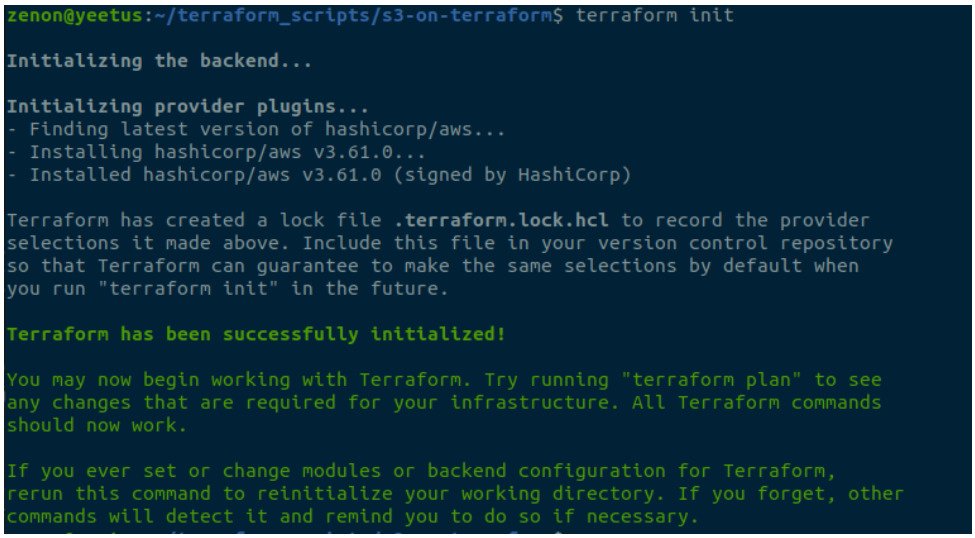


   s3-on-terraform.tf



2. Perform Terraform commands.

   `terraform init`



terraform plan

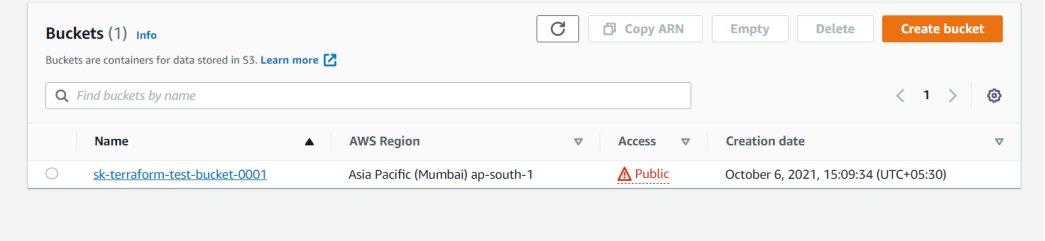
```
zenon@yeetus:~/terraform_scripts/s3-on-terraform$ terraform plan

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

Terraform will perform the following actions:

  # aws_s3_bucket.sreekesh will be created
  + resource "aws_s3_bucket" "sreekesh" {
      + acceleration_status         = (known after apply)
      + acl                         = "public-read"
      + arn                         = (known after apply)
      + bucket                      = "sk-terraform-test-bucket-0001"
      + bucket_domain_name          = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + region                      = (known after apply)
      + request_payer               = (known after apply)
      + tags                        = {
          + "Environment" = "Dev"
          + "Name"        = "Sreekesh S3 Bucket"
        }
      + tags_all                    = {
          + "Environment" = "Dev"
          + "Name"        = "Sreekesh S3 Bucket"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)

      + versioning {
          + enabled    = (known after apply)
          + mfa_delete = (known after apply)
        }
    }

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

terraform apply

```
zenon@yeetus:~/terraform_scripts/s3-on-terraform$ 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_s3_bucket.sreekesh will be created
  + resource "aws_s3_bucket" "sreekesh" {
      + acceleration_status         = (known after apply)
      + acl                         = "public-read"
      + arn                         = (known after apply)
      + bucket                      = "sk-terraform-test-bucket-0001"
      + bucket_domain_name          = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + region                      = (known after apply)
      + request_payer               = (known after apply)
      + tags                        = {
          + "Environment" = "Dev"
          + "Name"        = "Sreekesh S3 Bucket"
        }
      + tags_all                    = {
          + "Environment" = "Dev"
          + "Name"        = "Sreekesh S3 Bucket"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)

      + versioning {
          + enabled    = (known after apply)
          + mfa_delete = (known after apply)
        }
    }

Plan: 1 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_s3_bucket.sreekesh: Creating...
aws_s3_bucket.sreekesh: Creation complete after 3s [id=sk-terraform-test-bucket-0001]

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

3. After terraform apply, check your S3 Console if you have a new bucket that you just created.



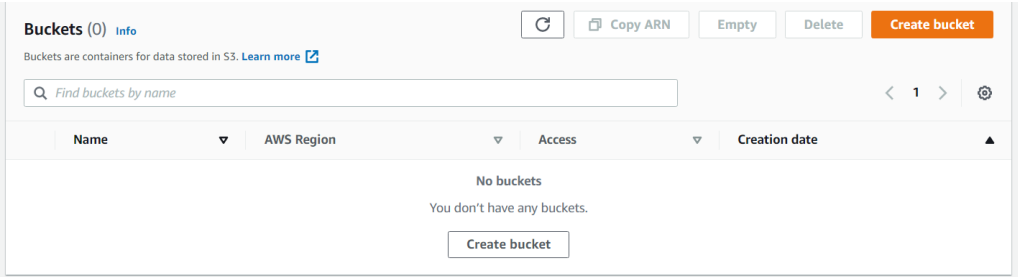
4. After you've created your bucket and verified it, you can delete this bucket by using terraform destroy.

`terraform destroy`

```
zenon@yeetus:~/terraform_scripts/s3-on-terraform$ terraform destroy
aws_s3_bucket.sreekesh: Refreshing state... [id=sk-terraform-test-bucket-0001]

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

Terraform will perform the following actions:

  # aws_s3_bucket.sreekesh will be destroyed
  - resource "aws_s3_bucket" "sreekesh" {
      - acl                         = "public-read" -> null
      - arn                         = "arn:aws:s3:::sk-terraform-test-bucket-0001" -> null
      - bucket                      = "sk-terraform-test-bucket-0001" -> null
      - bucket_domain_name          = "sk-terraform-test-bucket-0001.s3.amazonaws.com" -> null
      - bucket_regional_domain_name = "sk-terraform-test-bucket-0001.s3.ap-south-1.amazonaws.com" -> null
      - force_destroy               = false -> null
      - hosted_zone_id              = "Z11RGJOFQNVJUP" -> null
      - id                          = "sk-terraform-test-bucket-0001" -> null
      - region                      = "ap-south-1" -> null
      - request_payer               = "BucketOwner" -> null
      - tags                        = {
          - "Environment" = "Dev"
          - "Name"        = "Sreekesh S3 Bucket"
        } -> null
      - tags_all                    = {
          - "Environment" = "Dev"
          - "Name"        = "Sreekesh S3 Bucket"
        } -> null

      - versioning {
          - enabled    = false -> null
          - mfa_delete = false -> 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_s3_bucket.sreekesh: Destroying... [id=sk-terraform-test-bucket-0001]
aws_s3_bucket.sreekesh: Destruction complete after 0s

Destroy complete! Resources: 1 destroyed.
```

5. You can look back in the Console if the Bucket is destroyed.



Creating a docker image using terraform

1. Verify your docker installation by using

   `docker version`

```
zenon@yeetus:~$ docker version
Client: Docker Engine - Community
 Version:           20.10.9
 API version:       1.41
 Go version:        go1.16.8
 Git commit:        c2ea9bc
 Built:             Mon Oct  4 16:08:29 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true
```

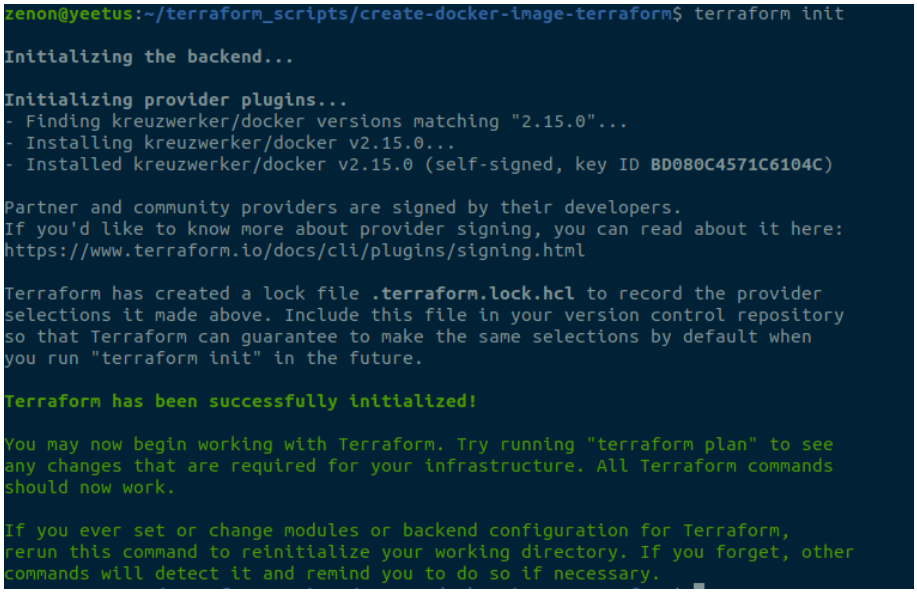
2. Create a new directory and a create-docker-image.tf file inside.

   create-docker-image.tf

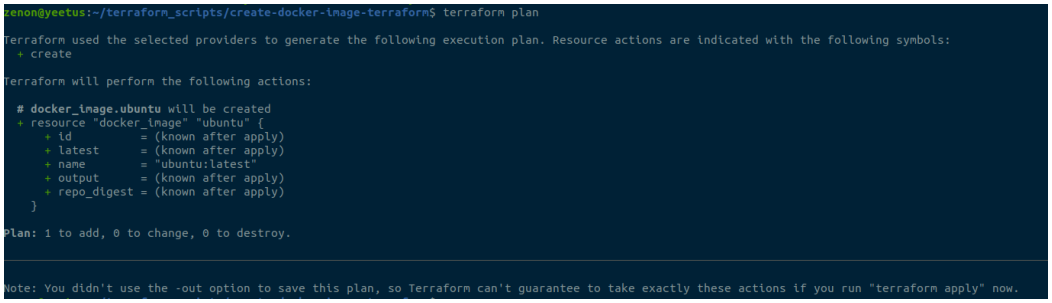
```
Open   create-docker-image.tf   Save
~/terraform_scripts/create-docker-image-terraform

terraform {
        required_providers {
                docker = {
                        source  = "kreuzwerker/docker"
                        version = "2.15.0"
                }
        }
}

provider "docker" {
  host = "unix:///var/run/docker.sock"
}

resource "docker_image" "ubuntu" {
        name = "ubuntu:latest"
}
```

3. Use terraform commands to create the docker image.

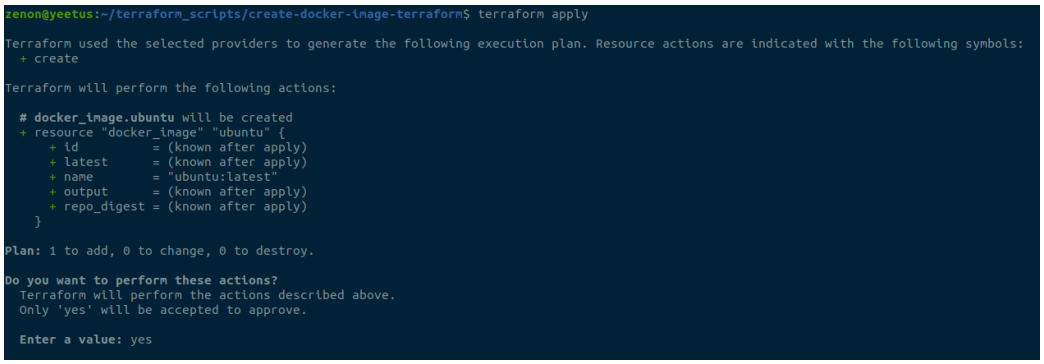
`terraform init`



`terraform plan`



`terraform apply`



```
docker_image.ubuntu: Creating...
docker_image.ubuntu: Still creating... [10s elapsed]
docker_image.ubuntu: Creation complete after 19s [id=sha256:597ce1600

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

4. Once terraform apply is complete, we can check the newly created docker image using the command -

`docker images`

```
zenon@yeetus:~/terraform_scripts/create-docker-image-terraform$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
ubuntu       latest    597ce1600cf4   5 days ago   72.8MB
```

`terraform destroy`

```
docker_image.ubuntu: Destroying... [id=sha256:597ce1600cf4ac5f449b66e75e840657bb53
docker_image.ubuntu: Destruction complete after 0s

Destroy complete! Resources: 1 destroyed.
zenon@yeetus:~/terraform_scripts/create-docker-image-terraform$ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
zenon@yeetus:~/terraform_scripts/create-docker-image-terraform$
```

You can run docker images again to confirm that the image was deleted successfully.

Conclusion:

In this experiment, we used Terraform to create and destroy an AWS EC2 Instance, an Amazon S3 bucket and a Docker image.