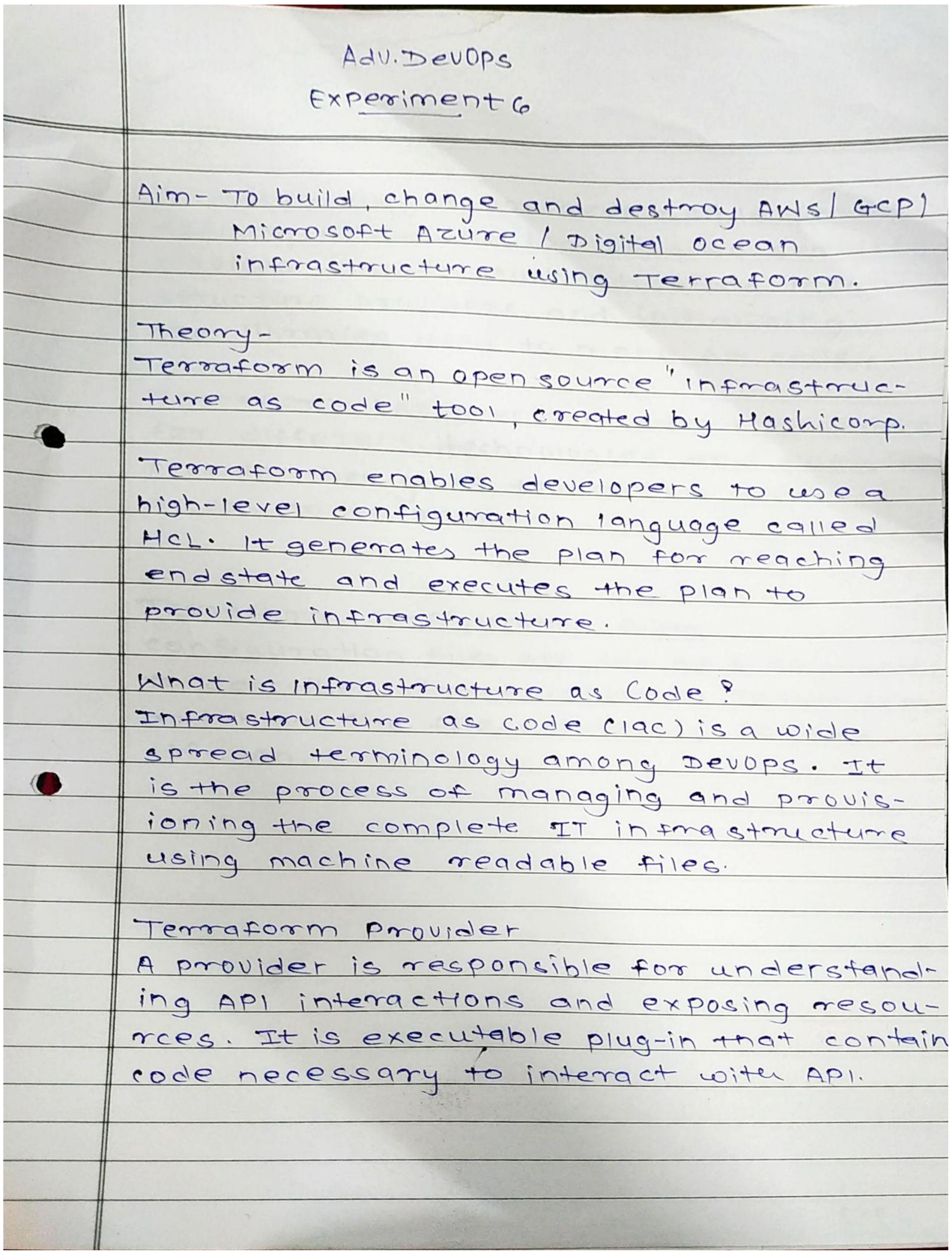
# Adv.DevOps Exp 06

**Name- Ishika Devare    RollNo- 14    Batch- A**



Adv.DevOps

Experiment 6

Aim- To build, change and destroy AWS / GCP / Microsoft Azure / Digital ocean infrastructure using Terraform.

Theory-

Terraform is an open source "infrastructure as code" tool, created by Hashicorp.

Terraform enables developers to use a high-level configuration language called HCL. It generates the plan for reaching endstate and executes the plan to provide infrastructure.

What is Infrastructure as Code?

Infrastructure as code (IaC) is a wide spread terminology among DevOps. It is the process of managing and provisioning the complete IT infrastructure using machine readable files.

Terraform Provider

A provider is responsible for understanding API interactions and exposing resources. It is executable plug-in that contain code necessary to interact with API.

Terraform plugins are responsible for defining resources for specific services. services include authenticating infra-structure providers and initializing the libraries used to make API calls.

Terraform has over a hundred providers for different technologies and each provider then gives terraform user access to its resources.

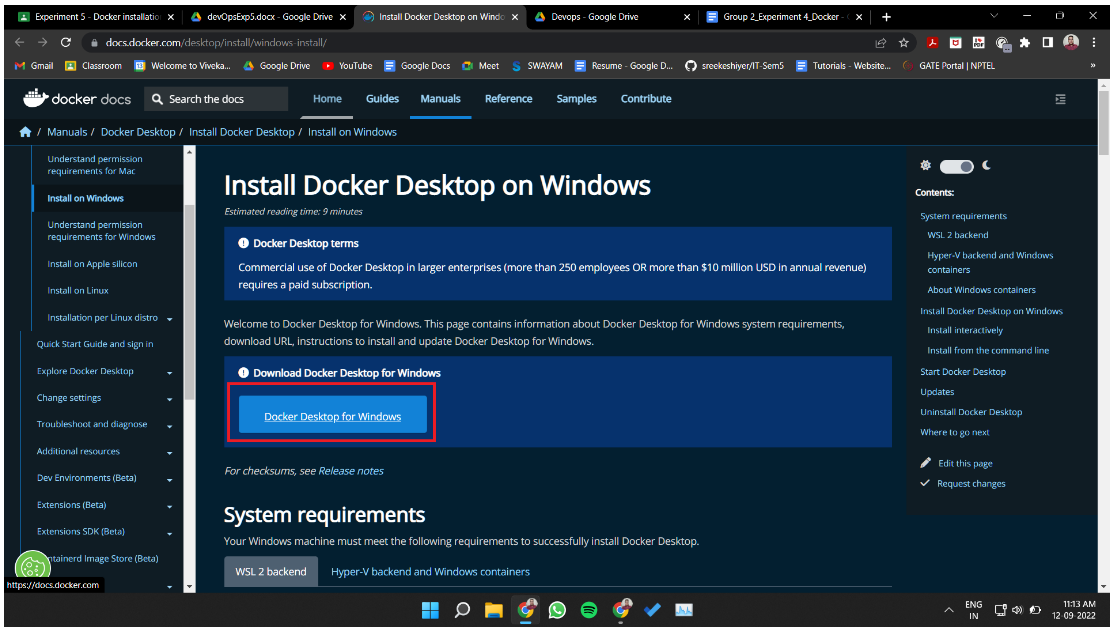
Terraform configuration files
configuration files are set of files used to describe infrastructure in Terraform and have the file extensions .tf and .tf.json.

**Implementation:**

**A. Creating docker image using terraform Prerequisite:**

**1) Download and Install Docker Desktop from [https://www.docker.com/](https://www.docker.com/)**

Step 1: Check the docker functionality


docs.docker.com/desktop/install/windows-install/
docker docs
Search the docs
Home
Guides
Manuals
Reference
Samples
Contribute
Manuals / Docker Desktop / Install Docker Desktop / Install on Windows
Install on Windows
Docker Desktop for Windows
Contents:
System requirements
WSL 2 backend
Hyper-V backend and Windows containers

Install Docker Desktop on Windows

*Estimated reading time: 9 minutes*

**Docker Desktop terms**

Commercial use of Docker Desktop in larger enterprises (more than 250 employees OR more than $10 million USD in annual revenue) requires a paid subscription.

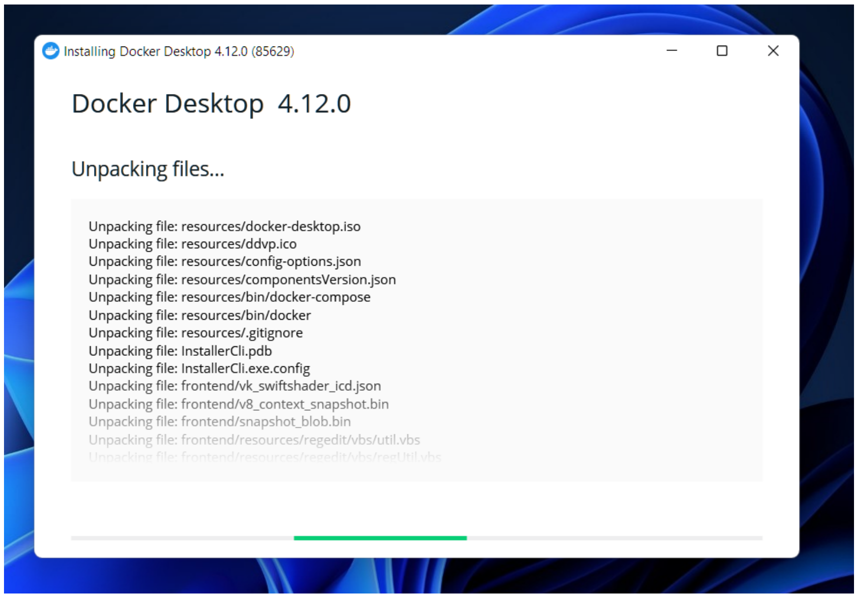
Welcome to Docker Desktop for Windows. This page contains information about Docker Desktop for Windows system requirements, download URL, instructions to install and update Docker Desktop for Windows.

**Download Docker Desktop for Windows**

*For checksums, see Release notes*

System requirements

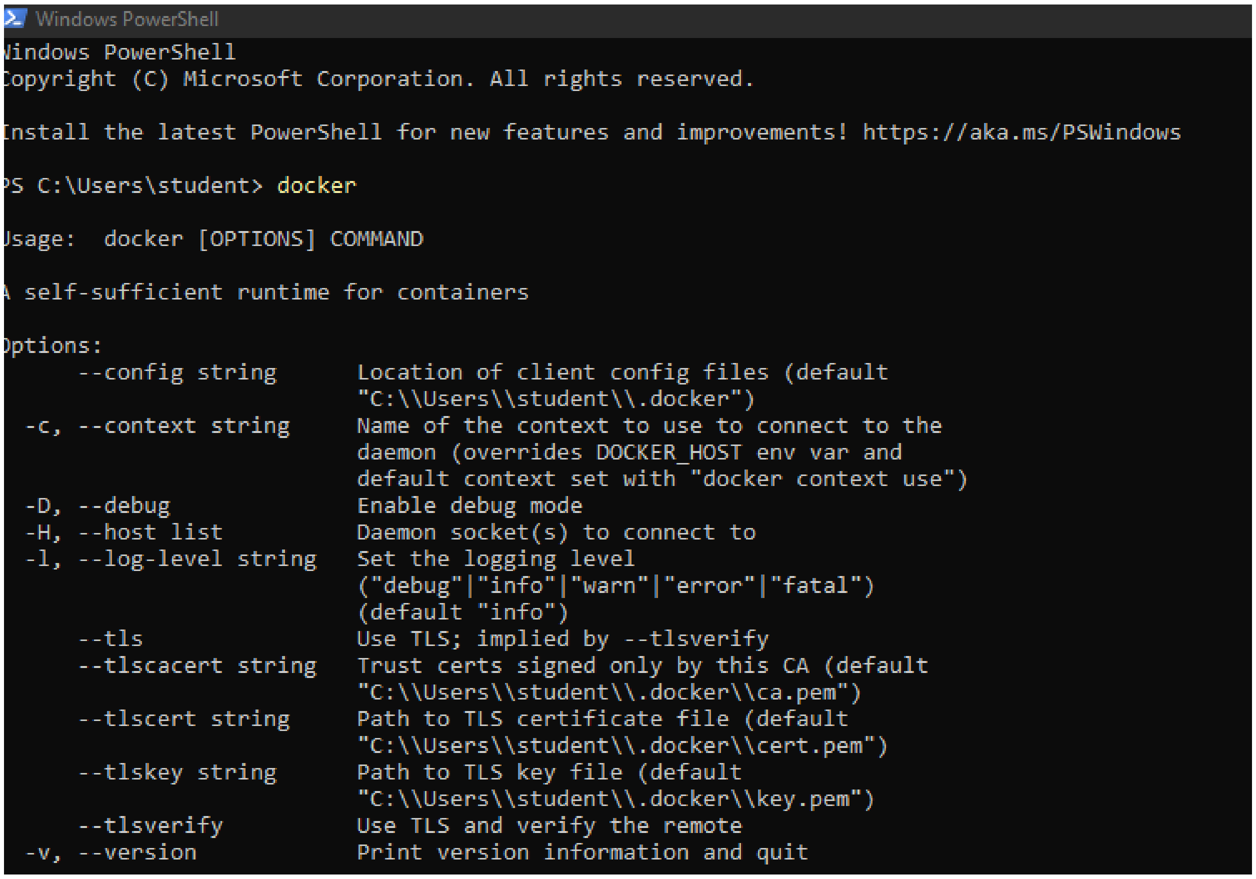
Your Windows machine must meet the following requirements to successfully install Docker Desktop.



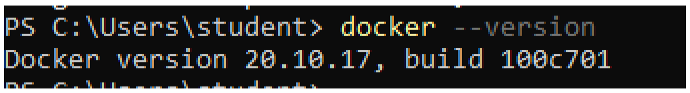
Docker Desktop 4.12.0

Unpacking files...

```
Unpacking file: resources/docker-desktop.iso
Unpacking file: resources/ddvp.ico
Unpacking file: resources/config-options.json
Unpacking file: resources/componentsVersion.json
Unpacking file: resources/bin/docker-compose
Unpacking file: resources/bin/docker
Unpacking file: resources/.gitignore
Unpacking file: InstallerCli.pdb
Unpacking file: InstallerCli.exe.config
Unpacking file: frontend/vk_swiftshader_icd.json
Unpacking file: frontend/v8_context_snapshot.bin
Unpacking file: frontend/snapshot_blob.bin
Unpacking file: frontend/resources/regedit/vbs/util.vbs
Unpacking file: frontend/resources/regedit/vbs/regUtil.vbs
```



```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\student> docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default
                           "C:\\Users\\student\\.docker")
  -c, --context string     Name of the context to use to connect to the
                           daemon (overrides DOCKER_HOST env var and
                           default context set with "docker context use")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level
                           ("debug"|"info"|"warn"|"error"|"fatal")
                           (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default
                           "C:\\Users\\student\\.docker\\ca.pem")
      --tlscert string     Path to TLS certificate file (default
                           "C:\\Users\\student\\.docker\\cert.pem")
      --tlskey string      Path to TLS key file (default
                           "C:\\Users\\student\\.docker\\key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit
```



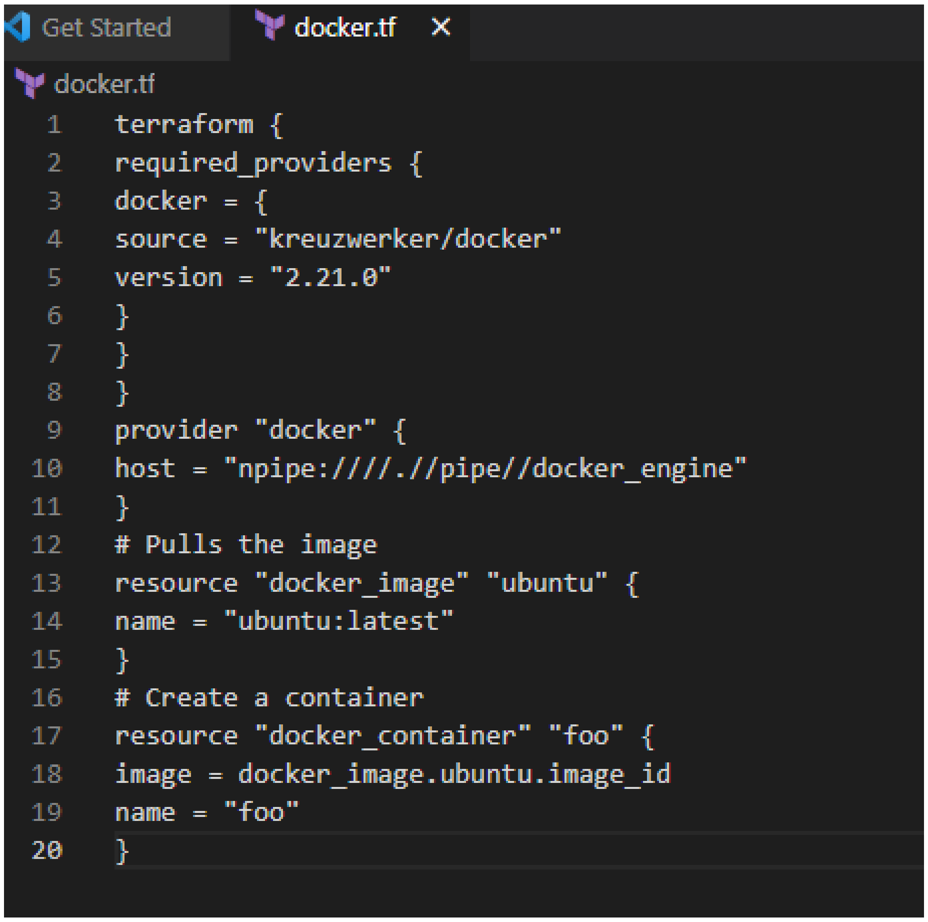
```
PS C:\Users\student> docker --version
Docker version 20.10.17, build 100c701
```

**Now, create a folder named ‘Terraform Scripts’ in which we save our different types of scripts which will be further used in this experiment.**

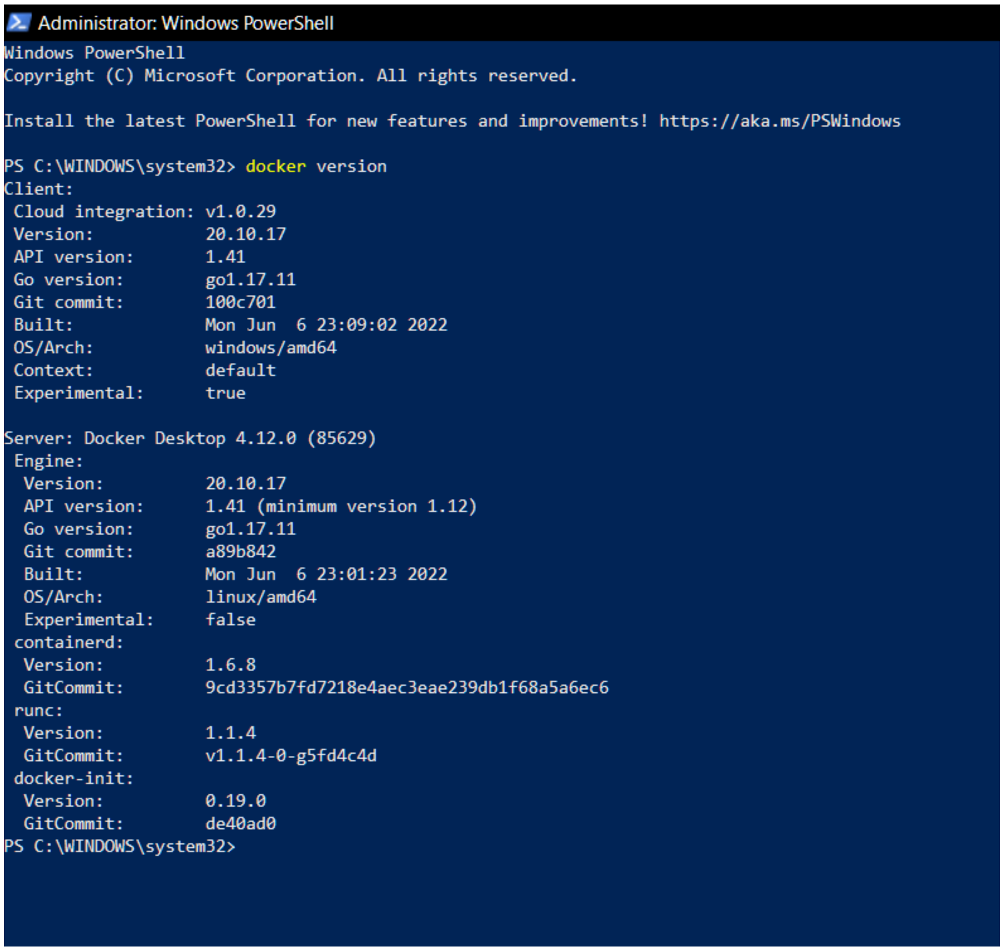
Step 2: Firstly create a new folder named ‘Docker’ in the ‘Terraform Scripts’ folder. Then create a new docker.tf file using Atom editor and write the following contents into it to create a Ubuntu Linux container.

**Script:**

```
terraform {
required_providers {
docker = {
source = "kreuzwerker/docker"
version = "2.21.0"
}
}
}
provider "docker" {
host = "npipe:////.//pipe//docker_engine"
}
# Pulls the image
resource "docker_image" "ubuntu" {
name = "ubuntu:latest"
}
# Create a container
resource "docker_container" "foo" {
image = docker_image.ubuntu.image_id
name = "foo"
}
```

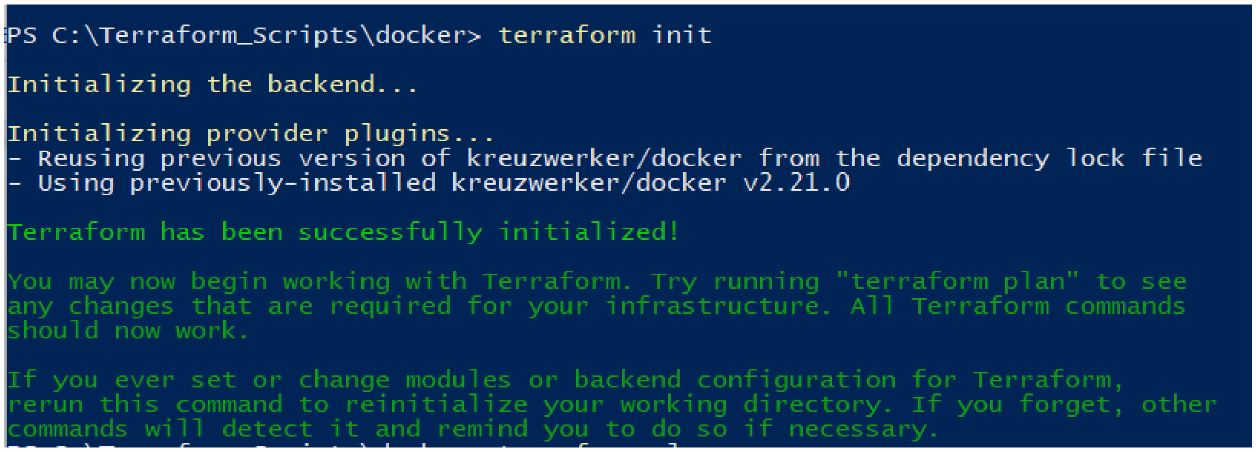


```
terraform {
required_providers {
docker = {
source = "kreuzwerker/docker"
version = "2.21.0"
}
}
}
provider "docker" {
host = "npipe:////.//pipe//docker_engine"
}
# Pulls the image
resource "docker_image" "ubuntu" {
name = "ubuntu:latest"
}
# Create a container
resource "docker_container" "foo" {
image = docker_image.ubuntu.image_id
name = "foo"
}
```



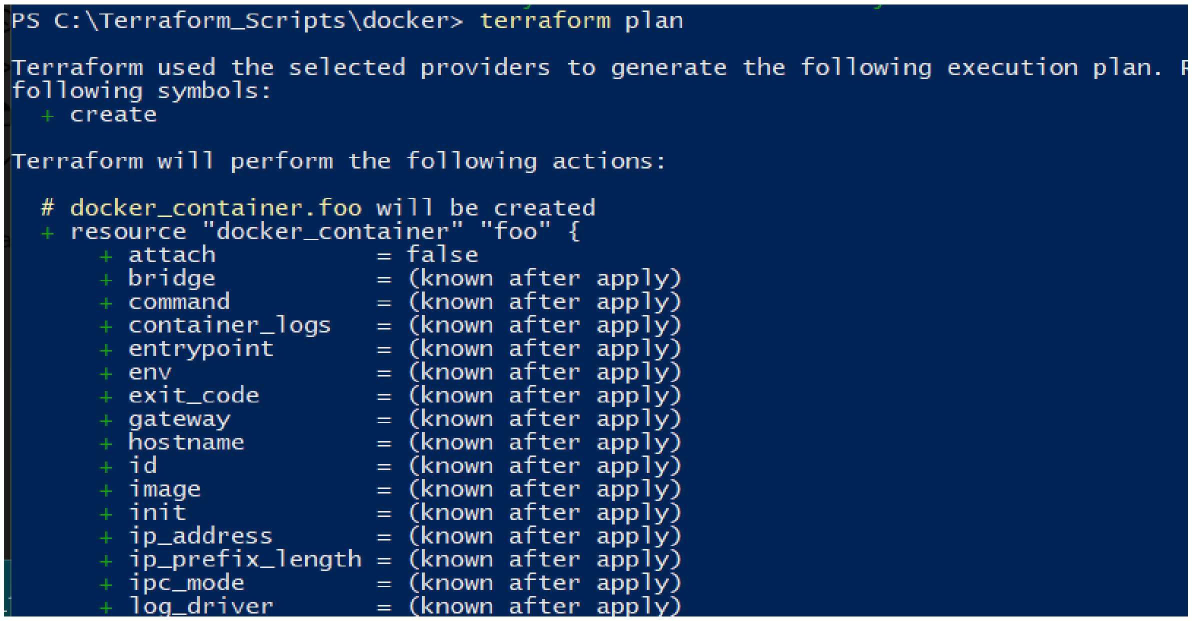
```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\WINDOWS\system32> docker version
Client:
 Cloud integration: v1.0.29
 Version:           20.10.17
 API version:       1.41
 Go version:        go1.17.11
 Git commit:        100c701
 Built:             Mon Jun  6 23:09:02 2022
 OS/Arch:           windows/amd64
 Context:           default
 Experimental:      true

Server: Docker Desktop 4.12.0 (85629)
 Engine:
  Version:          20.10.17
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.17.11
  Git commit:       a89b842
  Built:            Mon Jun  6 23:01:23 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.8
  GitCommit:        9cd3357b7fd7218e4aec3eae239db1f68a5a6ec6
 runc:
  Version:          1.1.4
  GitCommit:        v1.1.4-0-g5fd4c4d
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
PS C:\WINDOWS\system32>
```

Step 3: Execute Terraform Init command to initialize the resources using **terraform init**



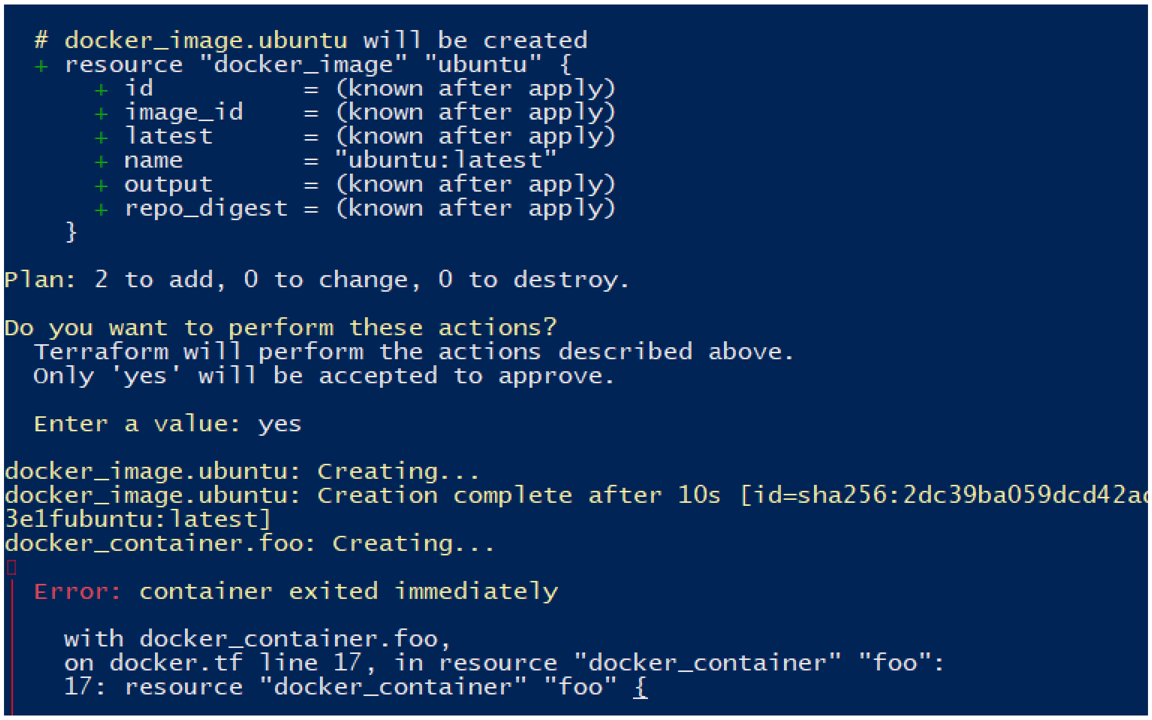
```
PS C:\Terraform_Scripts\docker> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of kreuzwerker/docker from the dependency lock file
- Using previously-installed kreuzwerker/docker v2.21.0

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.
```

**Step 4:** Execute Terraform plan to see the available resources using **Terraform plan**



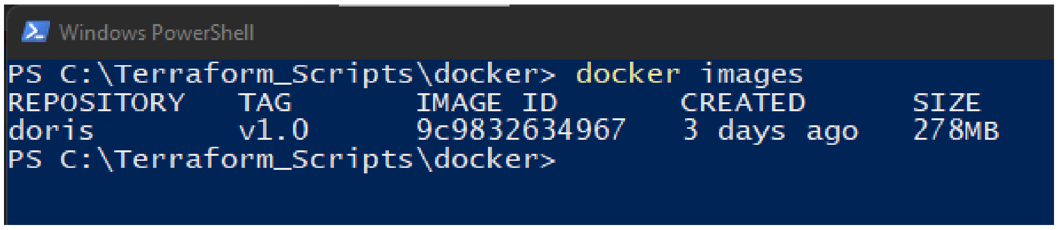
```
PS C:\Terraform_Scripts\docker> terraform plan

Terraform used the selected providers to generate the following execution plan. R
following symbols:
  + create

Terraform will perform the following actions:

  # docker_container.foo will be created
  + resource "docker_container" "foo" {
      + attach            = false
      + bridge            = (known after apply)
      + command           = (known after apply)
      + container_logs    = (known after apply)
      + entrypoint        = (known after apply)
      + env               = (known after apply)
      + exit_code         = (known after apply)
      + gateway           = (known after apply)
      + hostname          = (known after apply)
      + id                = (known after apply)
      + image             = (known after apply)
      + init              = (known after apply)
      + ip_address        = (known after apply)
      + ip_prefix_length  = (known after apply)
      + ipc_mode          = (known after apply)
      + log_driver        = (known after apply)
```

Step 5: Execute Terraform apply to apply the configuration, which will automatically create and run the Ubuntu Linux container based on our configuration. Using command : **"terraform apply"**



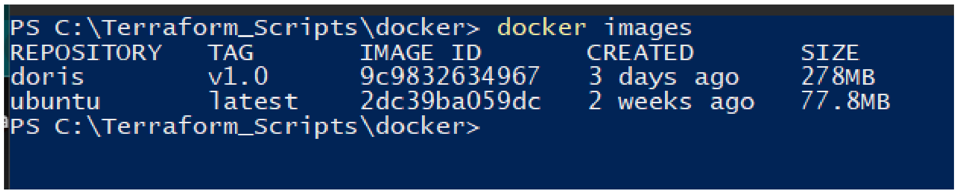
```
  # docker_image.ubuntu will be created
  + resource "docker_image" "ubuntu" {
      + id          = (known after apply)
      + image_id    = (known after apply)
      + latest      = (known after apply)
      + name        = "ubuntu:latest"
      + output      = (known after apply)
      + repo_digest = (known after apply)
    }

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

docker_image.ubuntu: Creating...
docker_image.ubuntu: Creation complete after 10s [id=sha256:2dc39ba059dcd42ac
3e1fubuntu:latest]
docker_container.foo: Creating...

  Error: container exited immediately

    with docker_container.foo,
    on docker.tf line 17, in resource "docker_container" "foo":
    17: resource "docker_container" "foo" {
```

Docker images before executing Apply step-



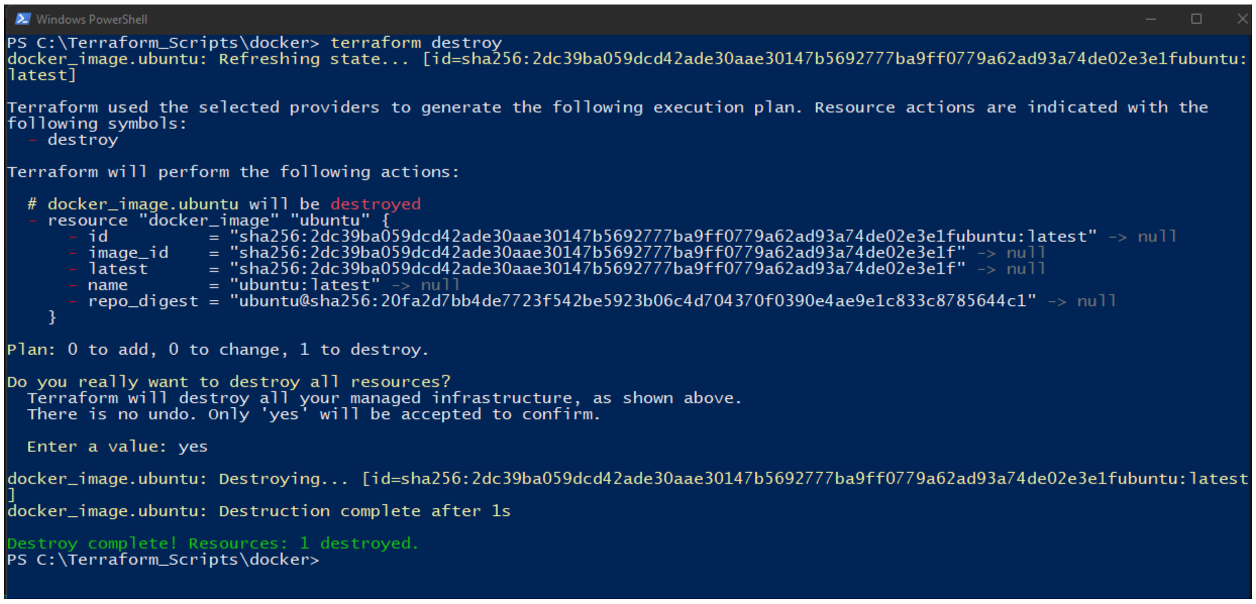
```
PS C:\Terraform_Scripts\docker> docker images
REPOSITORY    TAG        IMAGE ID       CREATED       SIZE
doris         v1.0       9c9832634967   3 days ago    278MB
PS C:\Terraform_Scripts\docker>
```

Docker images after executing Apply step-



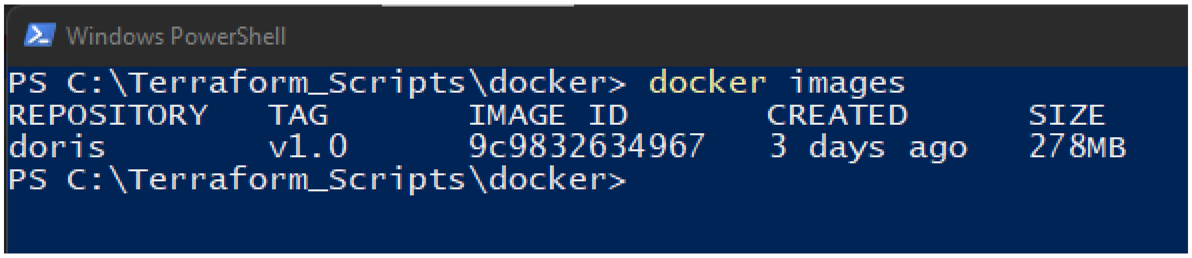
```
PS C:\Terraform_Scripts\docker> docker images
REPOSITORY    TAG        IMAGE ID       CREATED        SIZE
doris         v1.0       9c9832634967   3 days ago     278MB
ubuntu        latest     2dc39ba059dc   2 weeks ago    77.8MB
PS C:\Terraform_Scripts\docker>
```

Step 6: Execute Terraform destroy to delete the configuration, which will automatically delete the Ubuntu Container using **command Terraform destroy**



```
PS C:\Terraform_Scripts\docker> terraform destroy
docker_image.ubuntu: Refreshing state... [id=sha256:2dc39ba059dcd42ade30aae30147b5692777ba9ff0779a62ad93a74de02e3e1fubuntu:
latest]

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:

  # docker_image.ubuntu will be destroyed
  - resource "docker_image" "ubuntu" {
      - id           = "sha256:2dc39ba059dcd42ade30aae30147b5692777ba9ff0779a62ad93a74de02e3e1fubuntu:latest" -> null
      - image_id     = "sha256:2dc39ba059dcd42ade30aae30147b5692777ba9ff0779a62ad93a74de02e3e1f" -> null
      - latest       = "sha256:2dc39ba059dcd42ade30aae30147b5692777ba9ff0779a62ad93a74de02e3e1f" -> null
      - name         = "ubuntu:latest" -> null
      - repo_digest = "ubuntu@sha256:20fa2d7bb4de7723f542be5923b06c4d704370f0390e4ae9e1c833c8785644c1" -> 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

docker_image.ubuntu: Destroying... [id=sha256:2dc39ba059dcd42ade30aae30147b5692777ba9ff0779a62ad93a74de02e3e1fubuntu:latest
]
docker_image.ubuntu: Destruction complete after 1s

Destroy complete! Resources: 1 destroyed.
PS C:\Terraform_Scripts\docker>
```

Docker images After Executing Destroy step:



```
PS C:\Terraform_Scripts\docker> docker images
REPOSITORY    TAG        IMAGE ID       CREATED       SIZE
doris         v1.0       9c9832634967   3 days ago    278MB
PS C:\Terraform_Scripts\docker>
```

Conclusion- in this experiment, we installed docker and initialized it, planned, apply and displayed docker images before & after applying, and also destroyed it.