

## A. Creating docker image using terraform

### Prerequisite:

1) Download and Install Docker Desktop from <https://www.docker.com/>

#### Step 1: Check the docker functionality

```
Microsoft Windows [Version 10.0.22631.4037]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Ayush Maurya>docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
run      Create and run a new container from an image
exec     Execute a command in a running container
ps       List containers
build    Build an image from a Dockerfile
pull     Download an image from a registry
push     Upload an image to a registry
images   List images
login    Log in to a registry
logout   Log out from a registry
search   Search Docker Hub for images
version  Show the Docker version information
info     Display system-wide information

Management Commands:
builder  Manage builds
buildx*  Docker Buildx
checkpoint Manage checkpoints
compose* Docker Compose
container Manage containers
context  Manage contexts
debug*   Get a shell into any image or container
desktop* Docker Desktop commands (Alpha)
dev*     Docker Dev Environments
extension* Manages Docker extensions
feedback* Provide feedback, right in your terminal!
image    Manage images
init*    Creates Docker-related starter files for your project
manifest Manage Docker image manifests and manifest lists
network  Manage networks
plugin   Manage plugins
sbom*    View the packaged-based Software Bill Of Materials (SBOM) for an image

C:\Users\Ayush Maurya>docker --version
Docker version 27.0.3, build 7d4bcd8

C:\Users\Ayush Maurya>|
```

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 'TerraformScripts' folder. Then create a new docker.tf file using Atom editor and write the following contents into it to create a Ubuntu Linux container.

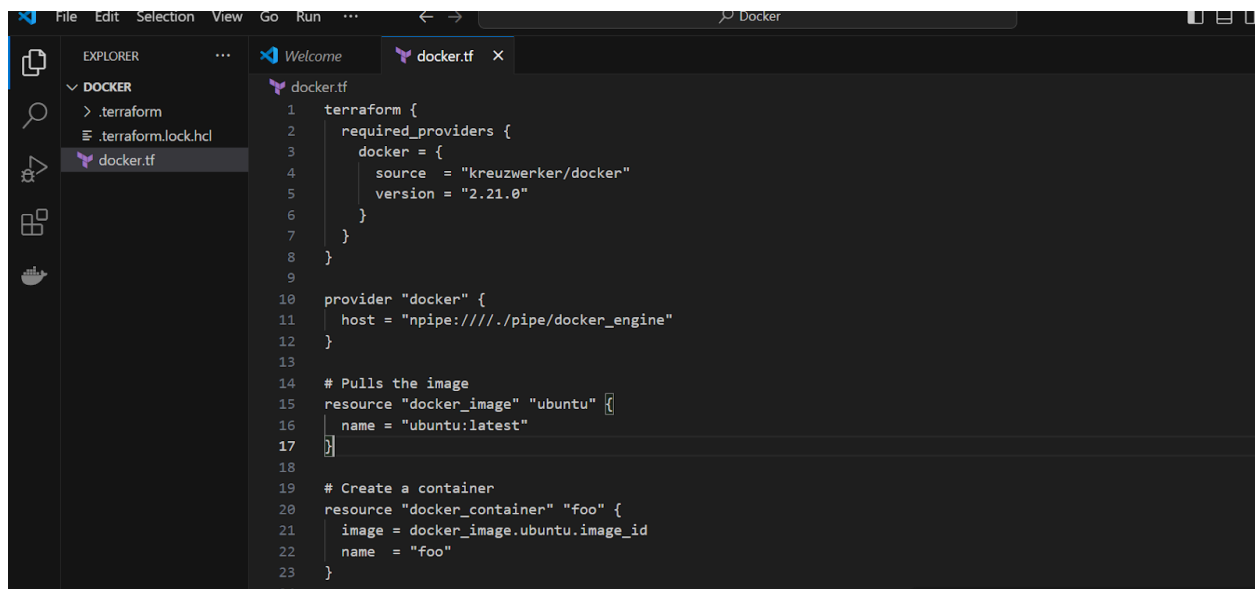
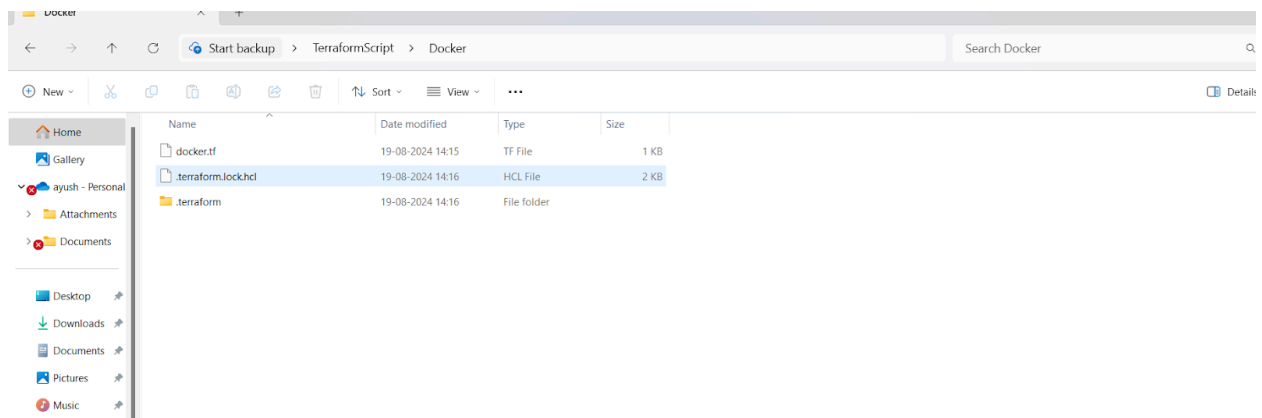
Script:  
terraform

```
{ 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"
}

```



### Step 3: Execute Terraform Init command to initialize the resources

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

```
PS C:\Users\Ayush Maurya\Desktop\TerraformScript\Docker> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding kreuzwerker/docker versions matching "2.21.0"...
- Installing kreuzwerker/docker v2.21.0...
- Installed kreuzwerker/docker v2.21.0 (self-signed, key ID BD080C4571C6104C)
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html
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.

### Step 4: Execute Terraform plan to see the available resources

```
PS C:\Users\Ayush Maurya\Desktop\TerraformScript\Docker> 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:

```
# 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)
  + logs        = false
  + must_run    = true
  + name        = "foo"
  + network_data = (known after apply)
  + read_only   = false
  + remove_volumes = true
  + restart     = "no"
  + rm          = false
  + runtime     = (known after apply)
  + security_opts = (known after apply)
  + shm_size    = (known after apply)
  + start       = true
  + stdin_open  = false
```

```
  + start       = true
  + stdin_open  = false
  + stop_signal = (known after apply)
  + stop_timeout = (known after apply)
  + tty         = false
```

```
  + healthcheck (known after apply)
```

```
  + labels (known after 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.

**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”

```
PS C:\Users\Ayush Maurya\Desktop\TerraformScript\Docker> 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:

# 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)
  + logs        = false
  + must_run    = true
  + name        = "foo"
  + network_data = (known after apply)
  + read_only   = false
  + remove_volumes = true
  + restart     = "no"
  + rm          = false
  + runtime     = (known after apply)

  + stdin_open    = false
  + stop_signal   = (known after apply)
  + stop_timeout  = (known after apply)
  + tty           = false

  + healthcheck (known after apply)

  + labels (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

docker_container.foo: Creating...
docker_container.foo: Creation complete after 1s [id=25e618f8f29715a205c36e0d31bfdd4a326dc83f513f2c7779aebd492ce9f602]

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

Docker images, Before Executing Apply step:

```
C:\Users\Ayush Maurya>docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
react-img     latest   d8b8903ee063   8 days ago    320MB
```

Docker images, After Executing Apply step:

```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
PS C:\Users\Ayush Maurya\Desktop\TerraformScript\Docker> docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
react-img     latest   d8b8903ee063   8 days ago    320MB
ubuntu        latest   edbfe744c41f8   2 weeks ago   78.1MB
```

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

```
PS C:\Users\Ayush Maurya\Desktop\TerraformScript\Docke> terraform destroy
docker_image.ubuntu: Refreshing state... [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598aubuntu:latest]
docker_container.foo: Refreshing state... [id=601782417e17994f75628bb648b3e4cbbb989037e3a3d302303c4a675a39599b]

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_container.foo will be destroyed
- resource "docker_container" "foo" {
  - attach      = false -> null
  - command    = [
    - "sh",
    - "-c",
    - "while true; do sleep 1; done",
  ] -> null
  - cpu_shares = 0 -> null
  - dns        = [] -> null
  - dns_opts   = [] -> null
  - dns_search = [] -> null
  - entrypoint = [] -> null
  - env        = [] -> null
  - gateway    = "172.17.0.1" -> null
  - group_add  = [] -> null
  - hostname   = "601782417e17" -> null
  - id         = "601782417e17994f75628bb648b3e4cbbb989037e3a3d302303c4a675a39599b" -> null
  - image      = "sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598a" -> null
  - init       = false -> null
  - ip_address = "172.17.0.2" -> null
  - ip_prefix_length = 16 -> null
  - ipc_mode   = "private" -> null
  - links      = [] -> null
  - log_driver = "json-file" -> null
  - log_opts   = {} -> null
  - logs       = false -> null
  - max_retry_count = 0 -> null
  - memory     = 0 -> null
  - memory_swap = 0 -> null
  - must_run   = true -> null
  - rm         = false -> null
  - runtime    = "runc" -> null
  - security_opts = [] -> null
  - shm_size    = 64 -> null
  - start       = true -> null
  - stdin_open  = false -> null
  - stop_timeout = 0 -> null
  - storage_opts = {} -> null
  - sysctls     = {} -> null
  - tmpfs       = {} -> null
  - tty         = false -> null
  # (8 unchanged attributes hidden)
}

# docker_image.ubuntu will be destroyed
- resource "docker_image" "ubuntu" {
  - id          = "sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598aubuntu:latest" -> null
  - image_id    = "sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598a" -> null
  - latest      = "sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598a" -> null
  - name        = "ubuntu:latest" -> null
  - repo_digest = "ubuntu@sha256:8a37d68f4f73ebf3d4efafbcf66379bf3728902a8038616808f04e34a9ab63ee" -> null
}

Plan: 0 to add, 0 to change, 2 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_container.foo: Destroying... [id=601782417e17994f75628bb648b3e4cbbb989037e3a3d302303c4a675a39599b]
docker_container.foo: Destruction complete after 0s
docker_image.ubuntu: Destroying... [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598aubuntu:latest]
docker_image.ubuntu: Destruction complete after 1s

Destroy complete! Resources: 2 destroyed.
PS C:\Users\Ayush Maurya\Desktop\TerraformScript\Docke> |
```

Docker images After Executing Destroy step

```
Destroy complete! Resources: 2 destroyed.
PS C:\Users\Ayush Maurya\Desktop\TerraformScript\Docke> docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
react-img latest d8b8903ee063 8 days ago 320MB
PS C:\Users\Ayush Maurya\Desktop\TerraformScript\Docke> |
```

Copilot