

DevOps Módulo 17 - IT Talent

Escrito por: [Gabriel Oliveira dos Santos](#)

Github: <https://github.com/Hypothesis>

Introdução a Infraestrutura como Código (IaC)

Introdução ao Infrastructure as Code (IaC)

- Gerencie infraestrutura com arquivos de configuração.
- Elimine a necessidade de uma interface gráfica.
- Proporcione segurança, consistência e repetibilidade.

Podemos arquivos de infraestrutura, invés de interface gráfica, assim melhora a segurança e consistência. Assim podemos reutilizar as configurações e automatizar configurações de infraestrutura.

Terraform

O que é Terraform?

- Ferramenta de IaC da HashiCorp.
- Define recursos e infraestrutura em arquivos de configuração declarativos.
- Gerencia o ciclo de vida da infraestrutura.

Em arquivos de declaração, podemos configurar ambientes de infraestruturas legíveis como humanos

Vantagens do Terraform

Vantagens do Terraform

- Gerencia infraestrutura em várias plataformas de nuvem.
- Linguagem de configuração legível por humanos.
- Rastreamento de mudanças de recursos.
- Controle de versão para colaboração segura.

Podemos lidar com a nuvem de forma ágil, com linguagem de fácil entendimento para humanos, além do fato de poder ser rastreável os recursos com controle de versão e colaboração seguro para a equipe DevOps e Infraestrutura.

Providers e Fluxo de Trabalho

Gerencie Qualquer Infraestrutura

- Providers permitem interação com APIs de serviços de nuvem.
- Mais de 1.000 providers disponíveis.
- Suporte para AWS, Azure, GCP, Kubernetes, entre outros.

Os providers, ou plugins, são softwares que ajudam a interação do nosso Terraform com serviços na Nuvem via API.

Padronize seu Fluxo de Trabalho

- Providers definem unidades de infraestrutura como recursos.
- Componha recursos em módulos reutilizáveis.
- Linguagem declarativa descreve o estado final desejado.

Os providers vão definir unidades individuais de infraestrutura, como rede privadas e entre outras configurações, que podem ser reutilizada em outros ambientes no Terraform.

Fluxo de Trabalho

Fluxo de Trabalho de Implantação do Terraform

- 1. Escopo - Identifique a infraestrutura do projeto.
- 2. Autor - Escreva a configuração.
- 3. Inicialize - Instale os plugins necessários.
- 4. Planeje - Visualize as mudanças.
- 5. Aplique - Execute as mudanças planejadas.

2x

1.5x

Os passos da imagem diz, que para um fluxo de trabalho bem sucedido, temos que identificar a infraestrutura do projeto, escrever a configuração, escolher e instalar os plugins necessários, planejar e executar o fluxo de trabalho, com implantação no Terraform.

Rastreamento de Infraestrutura

- Arquivo de estado como fonte de verdade.
- Determina mudanças para alinhar a infraestrutura à configuração.

O arquivo de estado é como a fonte real, ele que mostra de onde vem as mudanças e configurações na nossa infraestrutura.

Instalando o Terraform

Via linux, o código abaixo é possível instalar o terraform, via terminal:

```
sudo apt-get update && sudo apt-get install -y gnupg software
```

GPG key:

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-
```

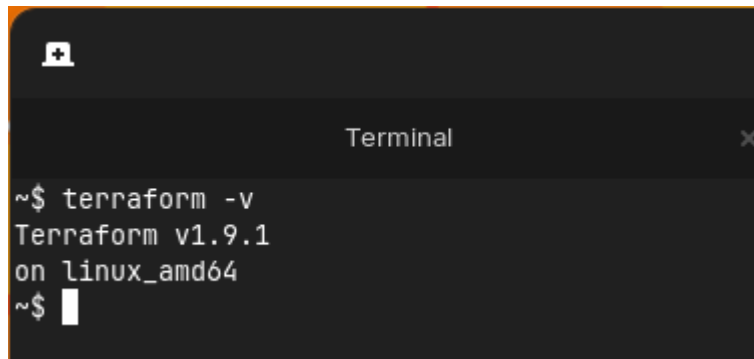
Repositorio HashiCorp:

```
sudo apt-add-repository "deb [arch=amd64] https://apt.release
```

Terraform:

```
sudo apt-get update  
sudo apt-get install terraform
```

Se tudo der certo, dando um `terraform -v`, irá aparecer no terminal a versão do terraform:

A screenshot of a terminal window titled "Terminal" with a close button. The terminal shows the command `terraform -v` being executed, resulting in the output `Terraform v1.9.1 on linux_amd64`. The prompt `~$` is visible at the end of the line.

```
~$ terraform -v  
Terraform v1.9.1  
on linux_amd64  
~$
```

Terraform Terminal Online

No link abaixo podemos rodar um teste de aplicação, usando terminal online para rodar um terraform instanciando um docker container:

<https://developer.hashicorp.com/terraform/tutorials/aws-get-s>

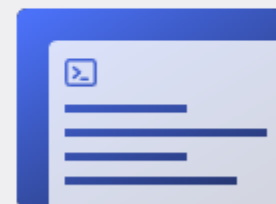
Quick Start

Use Terraform in a hosted terminal to build and destroy a Docker container.

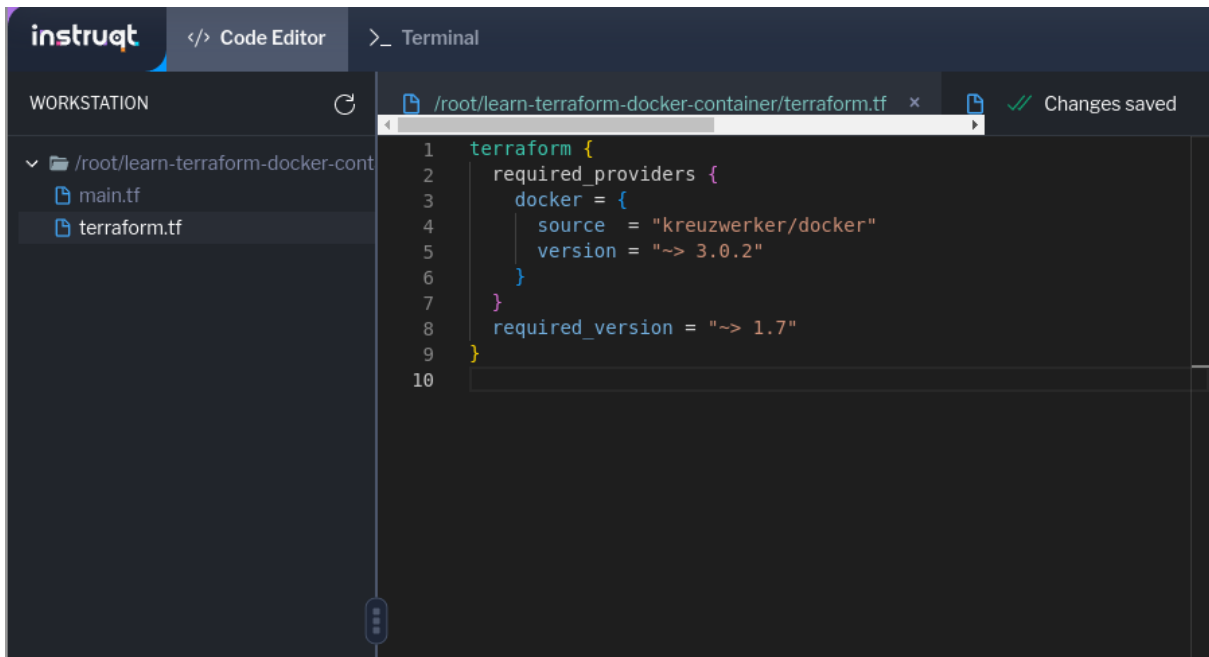
Launch Terminal

This tutorial includes a free interactive command-line lab that lets you follow along on actual cloud infrastructure.

Start interactive lab

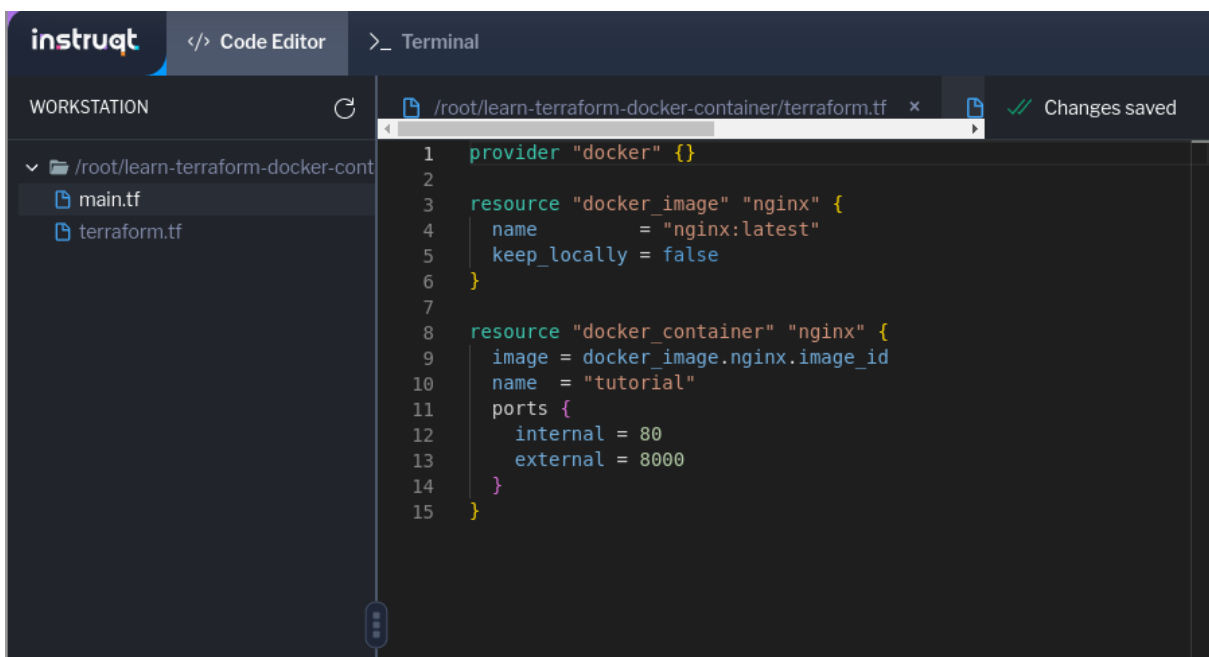


Na imagem abaixo temos o arquivo terraform.tf, que define nosso provider (plugin), em que o nosso terraform main.tf vai rodar, ou seja, com o **terraform init** vamos inicializar com o plugin do docker.



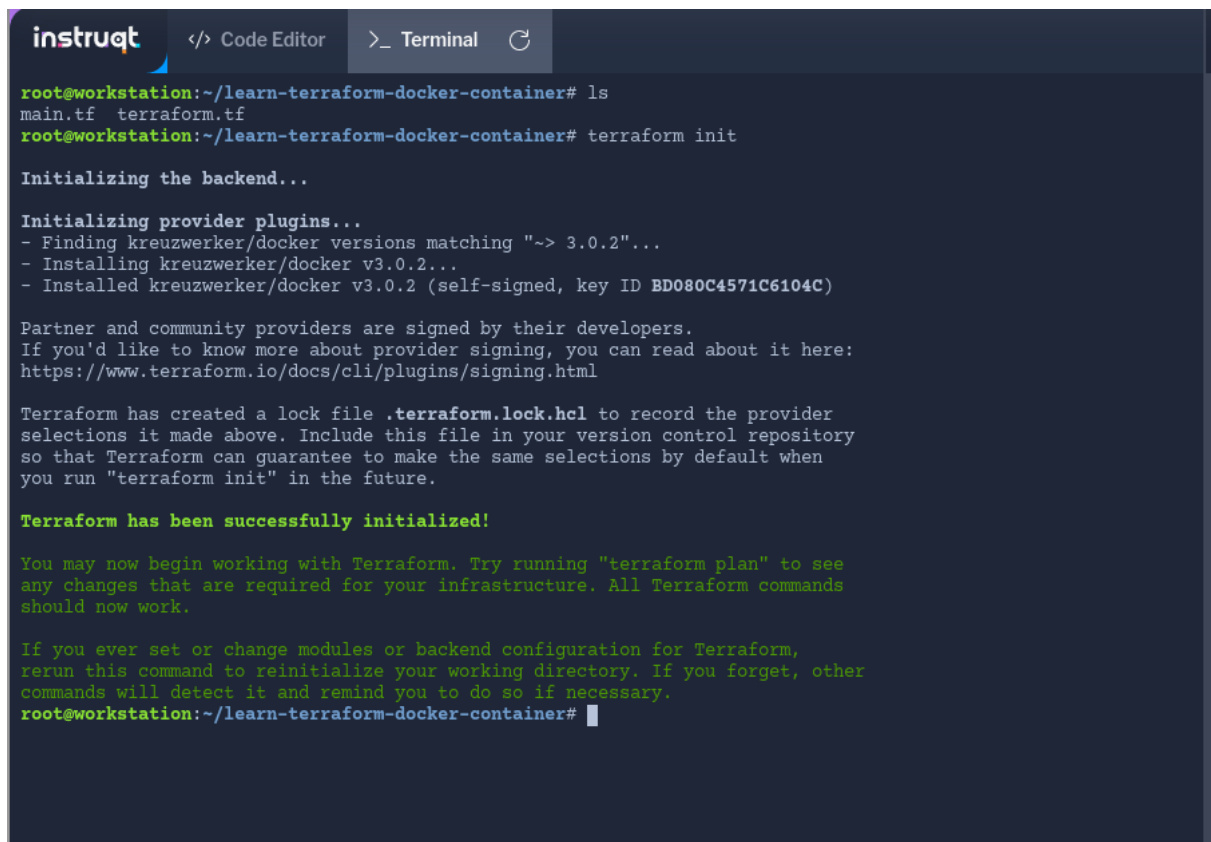
```
1 terraform {
2   required_providers {
3     docker = {
4       source = "kreuzwerker/docker"
5       version = "~> 3.0.2"
6     }
7   }
8   required_version = "~> 1.7"
9 }
10
```

No arquivo main.tf temos isso:



```
1 provider "docker" {}
2
3 resource "docker_image" "nginx" {
4   name = "nginx:latest"
5   keep_locally = false
6 }
7
8 resource "docker_container" "nginx" {
9   image = docker_image.nginx.image_id
10  name = "tutorial"
11  ports {
12    internal = 80
13    external = 8000
14  }
15 }
```

Agora rodamos com um terraform init:



The screenshot shows a terminal window with a dark background. At the top, there is a header bar with the 'instruct' logo on the left and two tabs labeled 'Code Editor' and 'Terminal' on the right. The 'Terminal' tab is active. The terminal content shows a user at a 'root@workstation' prompt in the directory '~/learn-terraform-docker-container'. They run 'ls', showing 'main.tf' and 'terraform.tf'. Then they run 'terraform init'. The output shows the backend being initialized, provider plugins being installed (kreuzwerker/docker v3.0.2), and a lock file being created. It concludes with 'Terraform has been successfully initialized!' and instructions on how to use 'terraform plan' and 'terraform init' again if needed.

```
root@workstation:~/learn-terraform-docker-container# ls
main.tf  terraform.tf
root@workstation:~/learn-terraform-docker-container# terraform init

Initializing the backend...

Initializing provider plugins...
- Finding kreuzwerker/docker versions matching "~> 3.0.2"...
- Installing kreuzwerker/docker v3.0.2...
- Installed kreuzwerker/docker v3.0.2 (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.
root@workstation:~/learn-terraform-docker-container#
```

Rodamos o terraform apply, para rodar e aplicar, assim rodar nosso container docker:


```
instruqt </> Code Editor >_ Terminal ↻

+ 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
+ stop_signal   = (known after apply)
+ stop_timeout = (known after apply)
+ tty          = false
+ wait         = false
+ wait_timeout = 60

+ ports {
  + external = 8000
  + internal = 80
  + ip       = "0.0.0.0"
  + protocol = "tcp"
}

# docker_image.nginx will be created
+ resource "docker_image" "nginx" {
  + id          = (known after apply)
  + image_id    = (known after apply)
  + keep_locally = false
  + name        = "nginx:latest"
  + repo_digest = (known after apply)
}

Plan: 2 to add, 0 to change, 0 to destroy.

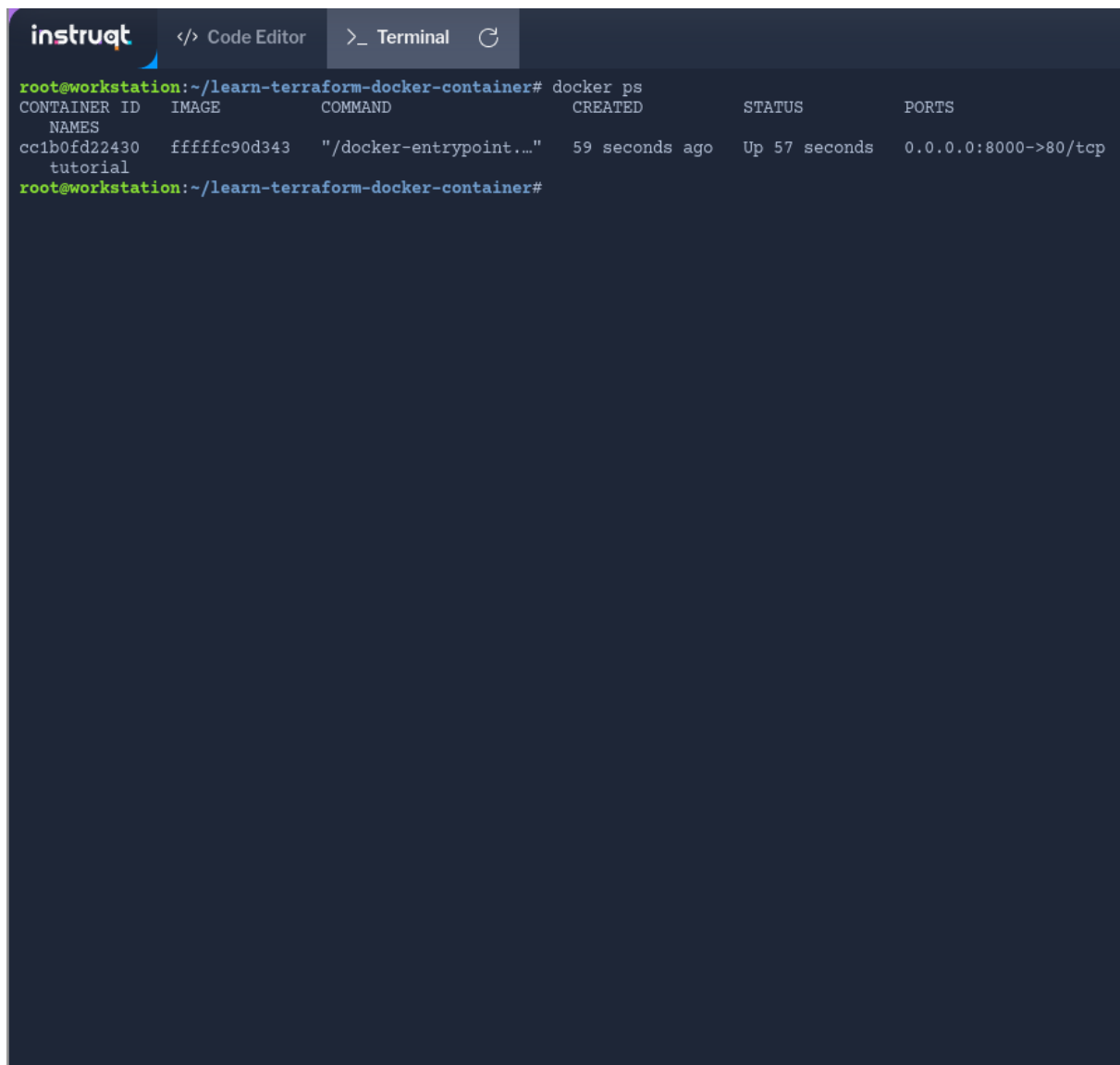
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

docker_image.nginx: Creating...
docker_image.nginx: Creation complete after 7s [id=sha256:ffffffc90d343cbcb01a5032edac86db5998c536cd0a366514121a45c6723765cnginx:latest]
docker_container.nginx: Creating...
docker_container.nginx: Creation complete after 2s [id=cc1b0fd2243072f65b12201039a604519fba30164b44a4a78c8ceffab764fb56]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

com o docker ps, vemos o container rodando:



```
instruct </> Code Editor >_ Terminal ↻
root@workstation:~/learn-terraform-docker-container# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
NAMES
cc1b0fd22430   fffffc90d343  "/docker-entrypoint...."  59 seconds ago Up 57 seconds  0.0.0.0:8000->80/tcp
tutorial
root@workstation:~/learn-terraform-docker-container#
```

e para destruir é so rodar terraform destroy:

```
instruqt  </> Code Editor  >_ Terminal  ↻

- rm                                = false -> null
- runtime                          = "runc" -> null
- security_opts                    = [] -> null
- shm_size                         = 64 -> null
- start                           = true -> null
- stdin_open                      = false -> null
- stop_signal                     = "SIGQUIT" -> null
- stop_timeout                    = 0 -> null
- storage_opts                    = {} -> null
- sysctls                        = {} -> null
- tmpfs                           = {} -> null
- tty                             = false -> null
- wait                           = false -> null
- wait_timeout                    = 60 -> null

- ports {
  - external = 8000 -> null
  - internal = 80 -> null
  - ip       = "0.0.0.0" -> null
  - protocol = "tcp" -> null
}

# docker_image.nginx will be destroyed
- resource "docker_image" "nginx" {
  - id          = "sha256:fffffc90d343cbcb01a5032edac86db5998c536cd0a366514121a45c6723765cnginx:latest"
-> null
  - image_id    = "sha256:fffffc90d343cbcb01a5032edac86db5998c536cd0a366514121a45c6723765c" -> null
  - keep_locally = false -> null
  - name        = "nginx:latest" -> null
  - repo_digest = "nginx@sha256:67682bda769fae1ccf5183192b8daf37b64cae99c6c3302650f6f8bf5f0f95df" -> null
}

Plan: 0 to add, 0 to change, 2 to destroy.

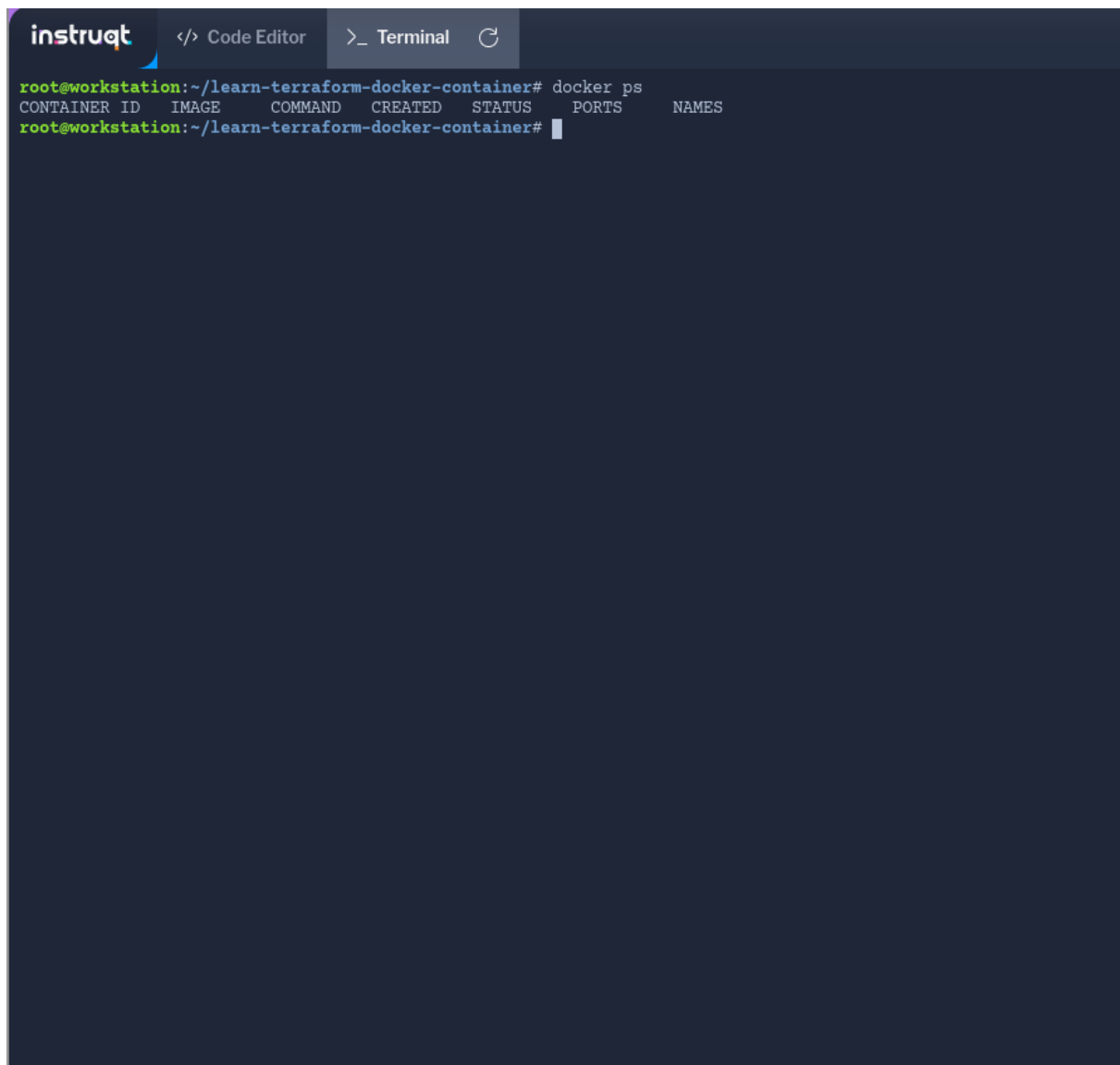
Do you really want to destroy all resources?
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes

docker_container.nginx: Destroying... [id=cc1b0fd2243072f65b12201039a604519fba30164b44a4a78c8ceffab764fb56]
docker_container.nginx: Destruction complete after 0s
docker_image.nginx: Destroying... [id=sha256:fffffc90d343cbcb01a5032edac86db5998c536cd0a366514121a45c6723765cnginx:latest]
docker_image.nginx: Destruction complete after 0s

Destroy complete! Resources: 2 destroyed.
```

Docker ps para checar se nao tem mesmo nenhum container rodando:



```
instruct </> Code Editor >_ Terminal ↻
root@workstation:~/learn-terraform-docker-container# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@workstation:~/learn-terraform-docker-container#
```

e é assim que podemos configurar infraestruturas de formas práticas e rápidas, trazendo o ambiente de teste e software mais próximo ao ambiente do usuário final.