

Lab Terraform: Infraestructura como código

Creá un bucket S3 y desplegué 2 EC2s con frontend + backend — todo desde código.

PARTE 1 — PREREQUISITOS

1 Instalá Terraform

```
# Mac
$ brew install terraform

# Windows (con Chocolatey)
$ choco install terraform

# Verificar
$ terraform --version
Terraform v1.x.x
```

Si no tenés Chocolatey/Homebrew, descargá Terraform directo desde developer.hashicorp.com/terraform/install

2 Verificá que AWS CLI esté configurada

```
$ aws sts get-caller-identity
{
  "Account": "123456789012",
  "Arn": "arn:aws:iam::123..."
}
```

Si da error, corré `aws configure` con tu Access Key y Secret Key (los de la sesión 2).

PARTE 2 — DESCARGÁ LOS ARCHIVOS

3 Cloná el repo y entrá a la carpeta

```
$ git clone https://github.com/micadevops/camino-devops-en-aws.git
$ cd camino-devops-en-aws/sesion4/sesion4-lab-home-terraform
```

4 Explorá la estructura

```
sesion4-lab-home-terraform/  
  01-s3-bucket/           # Ejercicio 1: un bucket S3  
    main.tf  
  02-ec2-app/             # Ejercicio 2: frontend + backend en EC2  
    main.tf               # Security group + 2 EC2s  
    variables.tf          # Variables (region, AMI, nombre)  
    outputs.tf            # IPs de salida  
    install-backend.sh    # Script que instala Flask  
    install-frontend.sh  # Script que instala nginx  
    .gitignore
```

PARTE 3 — EJERCICIO 1: BUCKET S3

Empezamos con lo más simple: crear un bucket S3 con **3 comandos**.

5 Mirá el código

```
# 01-s3-bucket/main.tf  
provider "aws" {  
  region = "us-east-1"  
}  
  
resource "aws_s3_bucket" "mi_bucket" {  
  bucket = "mi-primer-bucket-terraform-2026"  
}
```

7 líneas. `provider` dice "usé AWS", `resource` dice "creá un bucket con este nombre".

6 Init → Plan → Apply

```
$ cd 01-s3-bucket  
  
# 1. Descargar el provider de AWS  
$ terraform init  
Terraform has been successfully initialized!  
  
# 2. Ver qué va a hacer (sin tocar nada)  
$ terraform plan  
Plan: 1 to add, 0 to change, 0 to destroy.
```

```
# 3. Crear el recurso – escribí "yes"
$ terraform apply
Do you want to perform these actions? (yes)
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```



Entrá a la consola de AWS → S3 y vas a ver tu bucket recién creado

7

Destruir el bucket

```
$ terraform destroy
Plan: 0 to add, 0 to change, 1 to destroy.
Do you want to destroy? (yes)
Destroy complete! Resources: 1 destroyed.
```

El bucket desapareció de AWS. Terraform borró todo lo que él creó.



Ejercicio 1 completado

Creaste y destruiste un recurso real en AWS con código. Eso es Infrastructure as Code.

PARTE 4 – EJERCICIO 2: FRONTEND + BACKEND EN EC2

Ahora vamos a crear lo mismo que en la sesión 2 (2 EC2s con la app) pero **automáticamente**.

8

Entendé los archivos

`main.tf` crea 3 recursos:

```
# 1. Security Group – permite SSH, HTTP y puerto 5000
resource "aws_security_group" "app_sg" {
  name_prefix = "app-"
  vpc_id      = aws_vpc.vpc.id
}

# 2. EC2 Backend – instala Python + Flask
resource "aws_instance" "backend" {
  ami           = aws_ami.ubuntu.id
  instance_type = "t2.micro"
  user_data     = file("install-backend.sh")
}

# 3. EC2 Frontend – instala nginx + inyecta la IP del backend
```

```
resource "aws_instance" "frontend" {
  user_data = templatefile("install-frontend.sh", {
    backend_ip = aws_instance.backend.public_ip
  })
}
```

`user_data` es un script que la EC2 ejecuta al arrancar. `templatefile` inyecta la IP del backend en el script del frontend.

9 Verificá tu Key Pair

Necesitás el nombre del Key Pair que creaste en la sesión 2:

```
$ aws ec2 describe-key-pairs --query "KeyPairs[].KeyName" --output text
mi-key-sesion2
```

Anotá el nombre — lo vas a necesitar en el próximo paso.

10 Init + Apply

```
$ cd ../02-ec2-app

$ terraform init
Terraform has been successfully initialized!

$ terraform apply -var="key_name=mi-key-sesion2" -var="nombre=tunombre"
Plan: 3 to add, 0 to change, 0 to destroy.
Do you want to perform these actions? (yes)

Apply complete! Resources: 3 added.

Outputs:
ip_backend  = "54.210.123.45"
ip_frontend = "3.87.200.12"
url_frontend = "http://3.87.200.12"
```

Reemplazá `mi-key-sesion2` por el nombre de tu Key Pair real y `tunombre` por tu nombre.

ESPERÁ 2-3 MINUTOS

Las EC2 necesitan tiempo para arrancar e instalar todo (Python, Flask, nginx). Si no funciona de inmediato, esperá un poco.

11 Probá la app

```
# Probar el backend directo
$ curl http://54.210.123.45:5000/info?nombre=Mica
{"az":"us-east-1a","fecha":"29/07/2026 · 19:30 hs","host":"ip-172-31-..."}
```

Y abrí en el navegador la URL del frontend: `http://<ip_frontend>`



La misma app de la sesión 2, pero creada con un solo comando

12 Ver los outputs de nuevo

```
$ terraform output
ip_backend   = "54.210.123.45"
ip_frontend  = "3.87.200.12"
url_backend  = "http://54.210.123.45:5000/info?nombre=test"
url_frontend = "http://3.87.200.12"
```

13 Destruir todo

```
$ terraform destroy -var="key_name=mi-key-sesion2" -var="nombre=tunombre"
Plan: 0 to add, 0 to change, 3 to destroy.
Do you want to destroy? (yes)
Destroy complete! Resources: 3 destroyed.
```

Terraform borró las 2 EC2s y el Security Group. Sin recursos huérfanos, sin factura sorpresa.

IMPORTANTE: NO TE OLVIDES DEL DESTROY

Las EC2 cuestan plata mientras estén corriendo. Siempre corré `terraform destroy` cuando termines de practicar.



Lab completado

Creaste infraestructura real en AWS con código: un bucket S3, un Security Group y 2 EC2s con tu app desplegada.

B1 Ver qué recursos maneja Terraform

```
$ terraform state list
aws_instance.backend
aws_instance.frontend
aws_security_group.app_sg
```

B2 Ver detalles de un recurso

```
$ terraform show
# Muestra TODO: IPs, AMIs, security groups, tags...
```

B3 Formatear y validar el código

```
# Formatear (arregla indentación y espacios)
$ terraform fmt

# Validar sintaxis
$ terraform validate
Success! The configuration is valid.
```

B4 Modificar y re-aplicar

Cambiá algo en `main.tf` (por ejemplo el tag Name) y volvé a aplicar:

```
$ terraform plan
Plan: 0 to add, 1 to change, 0 to destroy.

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

Terraform solo modifica lo que cambió. No recrea todo desde cero.