Terraform_

Guía de atajos · IaC y DevSecOps

```
user@dajomar:~$ terraform plan
```

Conceptos clave

Provider	Plugin que habla con una plataforma (AWS, Azure, GitHub, Proxmox).
Resource	Algo que Terraform crea y administra.
Data source	Lee algo que ya existe, sin administrarlo.
Variable	Entrada configurable del código.
Output	Valor que se muestra o se pasa a otro módulo.
Module	Conjunto reutilizable de recursos.
State	Registro de lo que existe realmente. ⚠ Puede contener secretos.
Backend	Dónde se guarda el state (local, S3, Azure, HCP).

Terraform es **declarativo**: describes el estado final que quieres y él calcula qué crear, cambiar o borrar.

Estructura típica

```
proyecto/
├── main.tf              # recursos
├── providers.tf         # providers y versiones
├── variables.tf         # entradas
├── outputs.tf           # salidas
├── backend.tf           # dónde va el state
├── terraform.tfvars     # valores (no secretos)
├── .terraform.lock.hcl  # ✓ sí va a Git
└── modules/
```

Flujo de trabajo

```
terraform init              # descarga providers
terraform fmt -recursive    # formatea
terraform validate          # revisa sintaxis
terraform plan -out=tfplan  # qué va a cambiar
terraform apply tfplan  # aplica ese plan exacto
terraform destroy           # ⚠ borra todo
```

-var-file=prod.tfvars	Usa otro archivo de valores
-var="region=ca-central-1"	Pasa una variable suelta
-target=aws_s3_bucket.logs	Solo un recurso (emergencias)
-replace=aws_instance.web	Fuerza recrear un recurso
-refresh-only	Detecta cambios hechos a mano (drift)
-upgrade	Con init: actualiza providers

Pro Tip:
Guarda el plan con `-out` y aplica **ese** archivo. Así lo que revisaste es exactamente lo que se ejecuta. En el plan busca siempre las líneas `destroy` y `must be replaced`.

Lectura del plan

+	Se va a crear
~	Se modifica sin recrear
-/+	Se destruye y se vuelve a crear ⚠
-	Se va a destruir ⚠
<=	Se lee un data source

HCL: lo esencial

Providers con versión fija

```
terraform {
  required_version = ">= 1.6"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

provider "aws" {
  region = var.region
  default_tags {
    tags = { Owner = "devsecops", Env = var.env }
  }
}
```

Recurso seguro por defecto

```
resource "aws_s3_bucket" "logs" {
  bucket = "dajomar-logs-${var.env}"
  lifecycle { prevent_destroy = true }
}

resource "aws_s3_bucket_public_access_block" "logs" {
  bucket                  = aws_s3_bucket.logs.id
  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}
```

Variables con validación

```
variable "env" {
  type        = string
  description = "Entorno de despliegue"
  validation {
    condition = contains(["dev","prod"], var.env)
    error_message = "env debe ser dev o prod."
  }
}

variable "db_password" {
  type      = string
  sensitive = true  # no se muestra
}
```

Outputs, locals y data sources

```
locals {
  nombre = "app-${var.env}"
}

data "aws_caller_identity" "actual" {}

output "bucket_arn" {
  value = aws_s3_bucket.logs.arn
}
```

Varios recursos a la vez

```
resource "aws_iam_user" "equipo" {
  for_each = toset(["ana", "luis", "dmaza"])
  name     = each.key
}

resource "aws_instance" "web" {
  count = var.env == "prod" ? 2 : 1
  # ...
}
```

Prefiere `for_each` sobre `count`: si quitas un elemento de la lista, `count` puede recrear los demás.

State

```
terraform state list
terraform state show aws_s3_bucket.logs
terraform state mv aws_s3_bucket.a aws_s3_bucket.b
# deja de administrarlo
terraform state rm aws_s3_bucket.logs
terraform output
terraform output -json
```

Importar algo creado a mano

```
import {
  to = aws_s3_bucket.legacy
  id = "bucket-creado-a-mano"
}
```

```
terraform plan -generate-config-out=generado.tf
```

El bloque `import` (Terraform 1.5+) se revisa en el plan como cualquier cambio, a diferencia del viejo `terraform import`.

Backend remoto

```
terraform {
  backend "s3" {
    bucket       = "dajomar-tfstate"
    key          = "app/prod.tfstate"
    region       = "ca-central-1"
    encrypt      = true  # cifrado en reposo
    use_lockfile = true  # bloqueo (1.10+)
  }
}
```

El **lock** evita que dos personas apliquen cambios a la vez y corrompan el state. En versiones anteriores a la 1.10 se usaba una tabla de DynamoDB.

Módulos y workspaces

```
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "5.8.1"  # versión fija
  name    = local.nombre
  cidr    = "10.0.0.0/16"
}
```

```
terraform workspace list
terraform workspace new staging
terraform workspace select prod
```

Para separar entornos con distintos permisos, carpetas o cuentas distintas son más seguras que los workspaces.

DevSecOps: secretos

✗ **Nunca**	Contraseñas o llaves escritas en `.tf` o `.tfvars` versionados.
✓ **Variables**	`TF_VAR_db_password` como variable de entorno desde el CI o un vault.
✓ **Gestores**	Leer de AWS Secrets Manager, Azure Key Vault o HashiCorp Vault con un data source.
✓ **sensitive**	Oculta el valor en la consola y los logs.
⚠ **State**	`sensitive` **no** cifra el state: ahí los secretos quedan en texto plano. Protege el backend.
✓ **OIDC**	En CI, credenciales temporales de la nube en lugar de llaves fijas.

.gitignore para Terraform

```
.terraform/
*.tfstate
*.tfstate.*
*.tfvars  # salvo ejemplos
!example.tfvars
tfplan
crash.log
```

DevSecOps: escaneo

```
terraform fmt -check -recursive
terraform validate
tflint --recursive  # errores y malas prácticas
checkov -d .        # misconfigs de seguridad
trivy config .      # misconfigs + secretos
gitleaks detect     # secretos en el repo
```

Hallazgos típicos: buckets públicos, security groups abiertos a `0.0.0.0/0`, cifrado desactivado, logs apagados, IAM con `"*"`.

Escanear también el plan

```
terraform plan -out=tfplan
terraform show -json tfplan > plan.json
checkov -f plan.json
conftest test plan.json  # políticas OPA
```

Pipeline de CI (GitHub Actions)

```
name: terraform
on: [pull_request]
permissions:
  id-token: write  # OIDC
  contents: read
jobs:
  plan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: hashicorp/setup-terraform@v3
      - run: terraform fmt -check -recursive
      - run: terraform init -input=false
      - run: terraform validate
      - uses: bridgecrewio/checkov-action@master
        with: { directory: . }
      - run: terraform plan -input=false
```

El `apply` va en otro job, solo en `main` y con aprobación manual (environment protegido).

OpenTofu y depuración

```
tofu init ; tofu plan ; tofu apply  # mismo flujo
export TF_LOG=DEBUG  # logs detallados
export TF_LOG_PATH=./terraform.log
terraform console  # probar expresiones
terraform graph | dot -Tpng > grafo.png
```

OpenTofu es el fork open source de Terraform (Linux Foundation). Usa los mismos comandos con `tofu` y agrega cifrado nativo del state.

Pro Tip:
Nadie aplica desde su laptop en producción: todo cambio pasa por **pull request → fmt, validate y escaneo → plan revisado → apply automatizado**. Si alguien cambia algo a mano en la consola, `plan -refresh-only` lo detecta.