UD01 · Introducción y fundamentos de Terraform

Guía explicativa y visual en tema claro para comprender la **Infraestructura como Código** y el flujo de trabajo de **Terraform**.

{ } 1 · Infraestructura como Código (IaC)

Idea principal: Automatizar la creación y configuración de la infraestructura mediante archivos de texto, como si programaras tus servidores.

En lugar de crear máquinas o redes manualmente, describes su estado deseado. Terraform luego se encarga de construirlo, igual que un arquitecto sigue los planos de un edificio.

> **Ventajas:** reproducibilidad, control de versiones, rapidez en despliegues y facilidad para probar y destruir entornos.

- **Imperativo:** dices *cómo* hacerlo paso a paso.
- **Declarativo:** defines *qué* quieres obtener, y la herramienta decide cómo lograrlo.

2 · ¿Qué es Terraform?

Es una herramienta de **Infraestructura como Código** creada por **HashiCorp**. Usa el lenguaje **HCL** para describir tu infraestructura en archivos `.tf`.

Funciona con múltiples proveedores: AWS, Azure, GCP, GitHub, local, etc.

> Terraform es **declarativo**: tú defines el resultado y él se encarga de ejecutarlo.

Casos de uso:

- Crear redes, máquinas virtuales o bases de datos.
- Automatizar entornos de testing.
- Gestionar DNS, usuarios o servicios SaaS.

3 · Instalación

- **Linux:** `sudo apt install terraform`
- **macOS:** `brew install hashicorp/tap/terraform`
- **Windows:** `choco install terraform`

Verifica con `terraform version`. Si ves la versión, todo está correcto.

4 · Primer contacto con el CLI

Un ejemplo básico que genera un archivo local `hello.txt`:

```
 provider "local" {}
resource "local_file" "hello" {
  filename = "${path.module}/hello.txt"
  content  = "¡Hola mundo Terraform!"
}
```

- `terraform init`: descarga plugins necesarios.
- `terraform plan`: muestra los cambios.
- `terraform apply`: aplica la configuración.
- `terraform destroy`: borra los recursos.

> Usa `plan` antes de `apply` para evitar errores.

5 · Estructura de proyecto

- `main.tf` : recursos principales.
- `variables.tf` : variables.
- `outputs.tf` : salidas.

```
 variable "message" { default = "Hola Terraform" }
resource "local_file" "msg" {
  filename = "mensaje.txt"
  content  = var.message
}
output "file_path" {
  value = local_file.msg.filename
}
```

6 · Ciclo de vida

`init → plan → apply → destroy`

> Este flujo te asegura despliegues reproducibles y controlados.

7 · Estado (tfstate) y seguridad

Terraform guarda el estado en `terraform.tfstate`, con todos los recursos creados.

- No lo edites ni lo subas a GitHub.
- Usa backends remotos (S3, GCS).

```
terraform {
 backend "s3" {
   bucket = "mi-bucket"
   key    = "infra/terraform.tfstate"
   region = "eu-west-1"
```

```
  }
}
```

8 · Providers

Permiten que Terraform se comunique con APIs externas.

- **local:** crea archivos locales.
- **null:** ejecuta comandos.
- **random:** genera contraseñas seguras.

9 · Caso práctico: Token + archivo

Automatiza un flujo completo:

```
 resource "random_password" "token" {
  length  = 12
  special = true
}
resource "local_file" "file" {
  filename = "${path.module}/token.txt"
  content  = "Token: ${random_password.token.result}"
}
resource "null_resource" "notify" {
  provisioner "local-exec" {
    command = "echo 'Archivo creado'"
  }
}
```

Cheat Sheet UD01