# ∞ UD04 · Bloques avanzados, bucles y dependencias en Terraform

Guía práctica para usar lógica dinámica en Terraform: bucles, condicionales, dependencias y módulos reutilizables.

## 🚀 1 · Control dinámico de recursos

- `count` y `for_each` para crear recursos repetidos
- `condicionales` con operadores ternarios
- `depends_on` para definir orden de creación

> ✓ Mejora la **escalabilidad**, **mantenibilidad** y **reutilización** del código

## 123 2 · Repetición con count

Para crear múltiples instancias del mismo recurso:

```
resource "docker_container" "web" {
 count = 3
 name  = "web-${count.index}"
 image = "nginx:latest"
}
```

> 💡 Accede a cada uno con `docker_container.web[0]` , `[1]` , etc.

### Activación condicional

```
count = var.enable_logs ? 1 : 0
```

Ideal para entornos con recursos opcionales.

## 3 · Iteración con for_each

Perfecto para recursos con valores únicos:

```
variable "users" {
  default = ["alice", "bob"]
}
resource "local_file" "user_file" {
  for_each = toset(var.users)
  filename = "${each.key}.txt"
  content  = "Hola ${each.key}"
}
```

### Uso con mapas

```
variable "contenedores" {
  default = {
    nginx1 = 8081
    nginx2 = 8082
  }
}
resource "docker_container" "nginx" {
  for_each = var.contenedores
  name     = each.key
  ports {
    internal = 80
    external = each.value
  }
}
```

## 4 · Dependencias con depends_on

Fuerza el orden de creación entre recursos:

```
resource "docker_container" "nginx" {
  depends_on = [docker_network.red_local]
```

```
  ...
}
```

- ✅ Útil cuando no hay referencia directa entre recursos
- ⚠️ No abuses del uso innecesario

## </> 5 · Condicionales

Uso de operadores ternarios:

```
image = var.env == "prod" ? "nginx:stable" : "nginx:alpine"
```

### Activar/desactivar recursos

```
count = var.deploy_container ? 1 : 0
```

> ⚠️ Ambas ramas se evalúan, incluso si una no se usa

## 🛠️ 6 · Funciones integradas

- `length(var.list)` → cuenta elementos
- `join("-", ["web", "01"])` → "web-01"
- `split("-", "web-prod")` → ["web", "prod"]
- `lookup(var.map, "key", "default")`
- `replace("v1.0", ".", "-")` → "v1-0"

### Combinación con outputs

```
 locals {
  nombre = upper("web")
}
output "nombre_upper" {
```

```
  value = local.nombre
}
```

## 7 · Módulos en Terraform

Permiten reutilizar lógica y organizar mejor tu infraestructura:

### Estructura mínima:

- `main.tf`
- `variables.tf`
- `outputs.tf`

### Uso del módulo

```
module "file1" {
  source   = "./modules/files"
  filename = "saludo.txt"
  content  = "Hola desde el módulo"
}
output "archivo" {
  value = module.file1.file_path
}
```

> Puedes usar módulos locales o del Terraform Registry