

# Infrastructure as Code IaC

---

UPC - Cloud Computing Architecture

\$ whoami



## Marc Catrisse

- Ingeniero informático y Máster en Innovación y Investigación en la FIB
- Profesor asociado en la FIB
- Responsable técnico de proyectos y DevOPS en inLab FIB
  - Especializado en cloud (AWS)

<https://www.linkedin.com/in/marc-catrisse-99b065128/>

<https://inlab.fib.upc.edu/persones/marc-catrisse/>

# Índice - ¿Qué conceptos vamos a ver?

- Infrastructure as Code (IaC)
- Tipos de lenguaje
  - Imperativo
  - Declarativo
- Terraform
  - CLI
  - Backends
  - Módulos
  - Proveedores
  - Variables / Outputs
  - Recursos
  - Bloques
- Seguridad

# Introducción

- Las operaciones de nuestra empresa són ClickOPS y con algunos scripts en Python
  - La persona que creó todo el entorno ha salido de la empresa
  - Nadie sabe qué configuración se está utilizando
  - Los cambios de la infraestructura se han convertido en algo muy complejo
    - Recursos infrautilizados
    - Múltiples usuarios IAM / Roles definidos que no se están utilizando...
  - Se nos ha pedido replicar el entorno de producción en otra cuenta para crear uno de staging
    - Nadie sabe cómo hacerlo... y hay que volver a desplegarlo todo manualmente
      - + Tiempo y dinero
      - El entorno será idéntico?

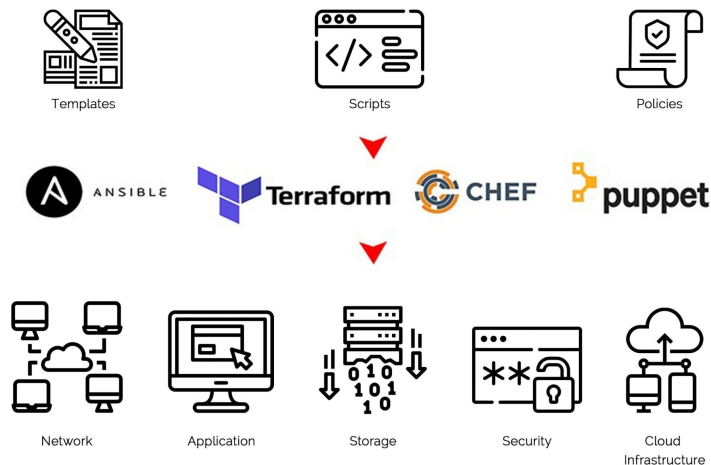
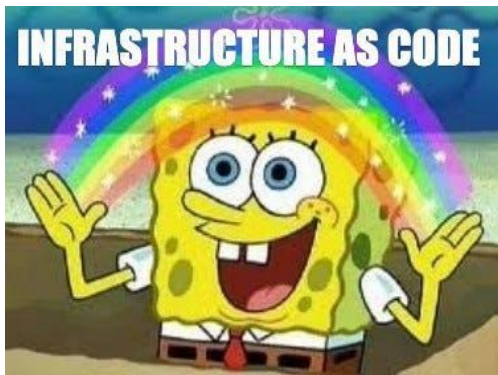


# Introducción

- El equipo a decidido que esto no puede ser
  - Tenemos que encontrar una manera de eliminar el ClickOps
  - Definir un conocimiento colectivo de la infraestructura
  - **Responsabilidad compartida de la infraestructura**
  - Incrementar la agilidad para introducir cambios

# Infrastructure as Code (IaC)

- Gestión y aprovisionamiento de infraestructura utilizando código
- Evitamos la configuración manual (ClickOps)
- Beneficios
  - Junto al uso de VC nos permite tener el histórico de la evolución de la infraestructura
  - Nos permite replicar el stack facilmente
  - Reducimos errores manuales
  - Mejora la consistencia



# Infrastructure as Code (IaC)

- Objetivos

- Evitar cambios manuales no documentados
- Facilitar la colaboración entre múltiples gestores de infraestructura

- Contrás

- Requiere el uso de herramientas externas
- Curva de aprendizaje
- Incompatible con cambios manuales



# Tipos de lenguajes - Imperativo

- Definimos los comandos para llegar al estado deseado
- Los comandos se tienen que ejecutar en orden
- **Cuidado!**
  - Los comandos deben ser condicionados al estado actual de la infraestructura
- Ejemplo:

```
1  #!/bin/bash
2
3  AWS_DEFAULT_REGION=us-east-1
4
5  #Create VPC
6  VPC=aws ec2 create-vpc \
7      --cidr-block 10.0.0.0/16 \
8      --tag-specification "ResourceType=vpc,Tags=[{Key=OwnedBy,Value=BashScript}]" \
9      --query Vpc.VpcId \
10     --output text
11
12 # Create Subnet
13 aws ec2 create-subnet \
14     --vpc-id $VPC \
15     --cidr-block 10.0.1.0/24 \
16     --availability-zone us-east-1a \
17     --tag-specification "ResourceType=subnet,Tags=[{Key=OwnedBy,Value=BashScript}]"
```



# Tipos de lenguajes - Declarativo

- Definimos el estado final deseado
  - Recursos y propiedades
- Debemos mantener el estado del estado actual de sistema
- Compara el estado actual con el deseado y automáticamente genera la acciones requeridas
- **La mayoría de herramientas IaC son declarativas**



```
1 resource "aws_vpc" "main" {
2     cidr_block      = "10.0.0.0/16"
3     tags = {
4         OwnedBy = "BashScript"
5     }
6 }
7
8 resource "aws_subnet" "main" {
9     vpc_id      = aws_vpc.main.id
10    cidr_block = "10.0.1.0/24"
11    tags = {
12        Name = "Main"
13    }
14 }
```

Terraform HCL

# Lab10 - Imperative

- Practicar la automatización imperativa
- Observaremos las limitaciones del uso de lenguajes imperativos
- Que haremos? Crearemos recursos de red
  - VPC
  - Subnet
  - IGW
  - Route table
  - Security group

<https://github.com/upcschool-cloud-arch/contenido/tree/main/09-gitops-terraform/labs/lab10-imperative>

# Infrastructure as Code (IaC) - Herramientas

- **Salstack**
  - Salt Agent
  - Via SSH
  - YAML
- **Ansible**
  - Agentless
  - Via SSH
  - YAML (PlayBooks)
- **Terraform**
  - HCL (HashiCorp Configuration Language)
  - Via API
  - Multiple Cloud providers (AWS, GCP, Azure...)
- **AWS CloudFormation**
  - AWS
  - JSON/YAML



# Terraform



- Creado por HashiCorp en 2014
- Declarativo
- Utiliza un lenguaje propio de alto nivel
  - HCL (HashiCorp Configuration Language)
- Define un grafo de dependencias para optimizar la creación y modificación de recursos
- Múltiples proveedores
  - Cloud: AWS, GCP, Azure, Oracle Cloud....
  - On Premise: VMWare vSphere, OpenStack....
- Compatible con múltiples backends
  - Almacenado del estado de la infraestructura
- Licencia BUSL (**No es OpenSource**)

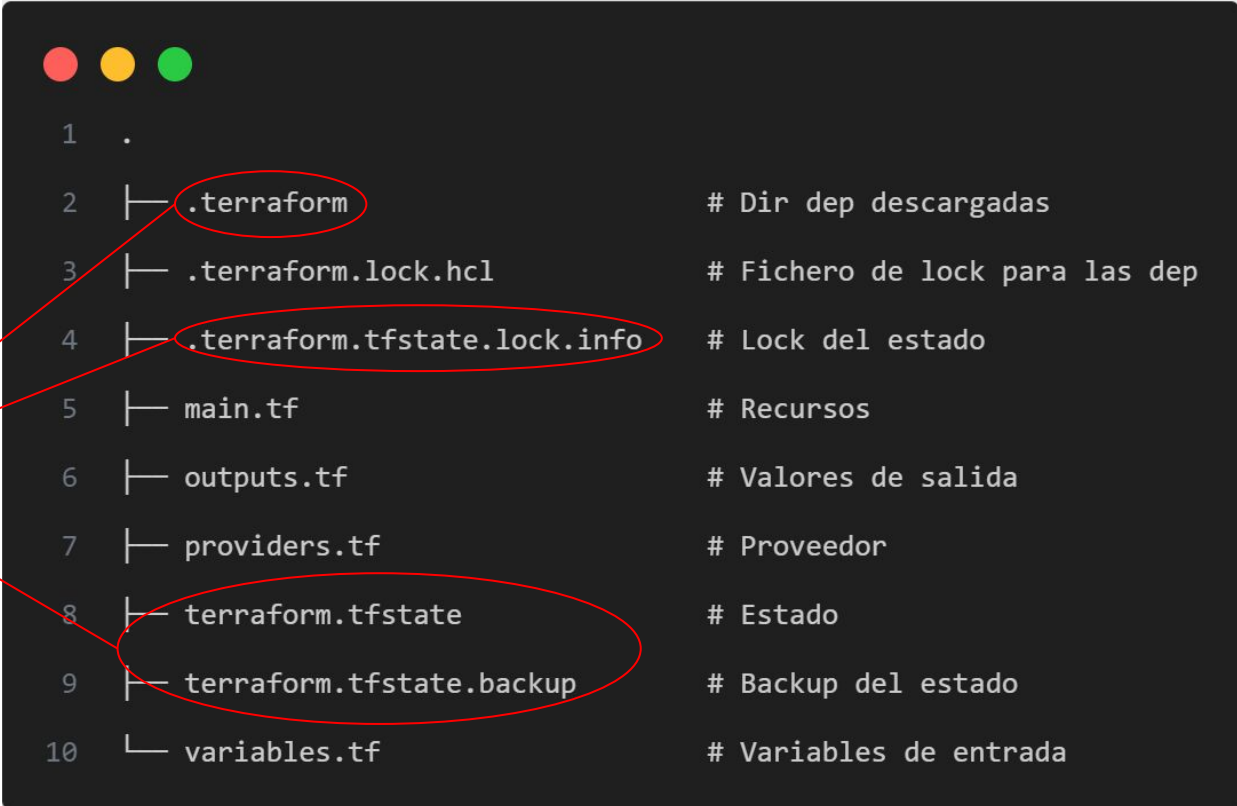
# Terraform - Alternativas?



- OpenTofu
- 10 d'agost de 2023 Hashicorp cambia la licencia de Terraform
  - **Antes:** MPV v2.0 (Mozilla Public License) (**Open Source**)
  - **Ahora:** BUSL (Business Source License) (**NO Open Source**)
    - Limita el uso del software para ofrecer *productos competitivos*
- Negativa de Hashicorp de volver a la licencia Open Source
  - 25 de Agosto de 2023
    - La comunidad hace un Fork de la última versión Open Source (OpenTofu)
    - Mantenida por **The Linux Foundation**
- Compatible con el código HCL existente

# Terraform - Ficheros

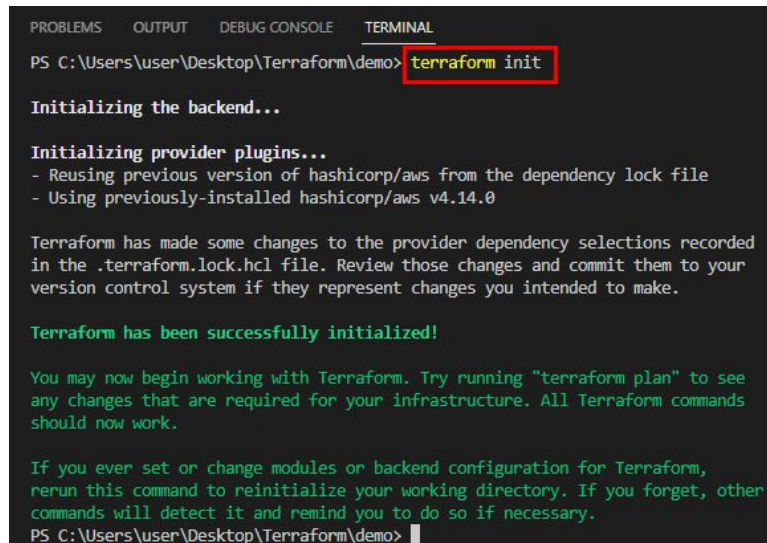
No incluir en VC



```
1  .
2  ├── .terraform                # Dir dep descargadas
3  ├── .terraform.lock.hcl      # Fichero de lock para las dep
4  ├── .terraform.tfstate.lock.info # Lock del estado
5  ├── main.tf                  # Recursos
6  ├── outputs.tf               # Valores de salida
7  ├── providers.tf             # Proveedor
8  ├── terraform.tfstate         # Estado
9  ├── terraform.tfstate.backup  # Backup del estado
10 └── variables.tf              # Variables de entrada
```

# Terraform - CLI

- **\$ terraform init**
  - Inicializa el directorio de trabajo
  - Descarga e inicializa las dependencias del proveedor
  - Crea un backend si no hay uno definido
    - Local path por defecto



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\Users\user\Desktop\Terraform\demo> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.14.0

Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.

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.
PS C:\Users\user\Desktop\Terraform\demo>
```

# Terraform - CLI

- **\$ terraform apply**
  - Crea o modifica la infraestructura
  - Define un plan si no existe previamente y te pide confirmación

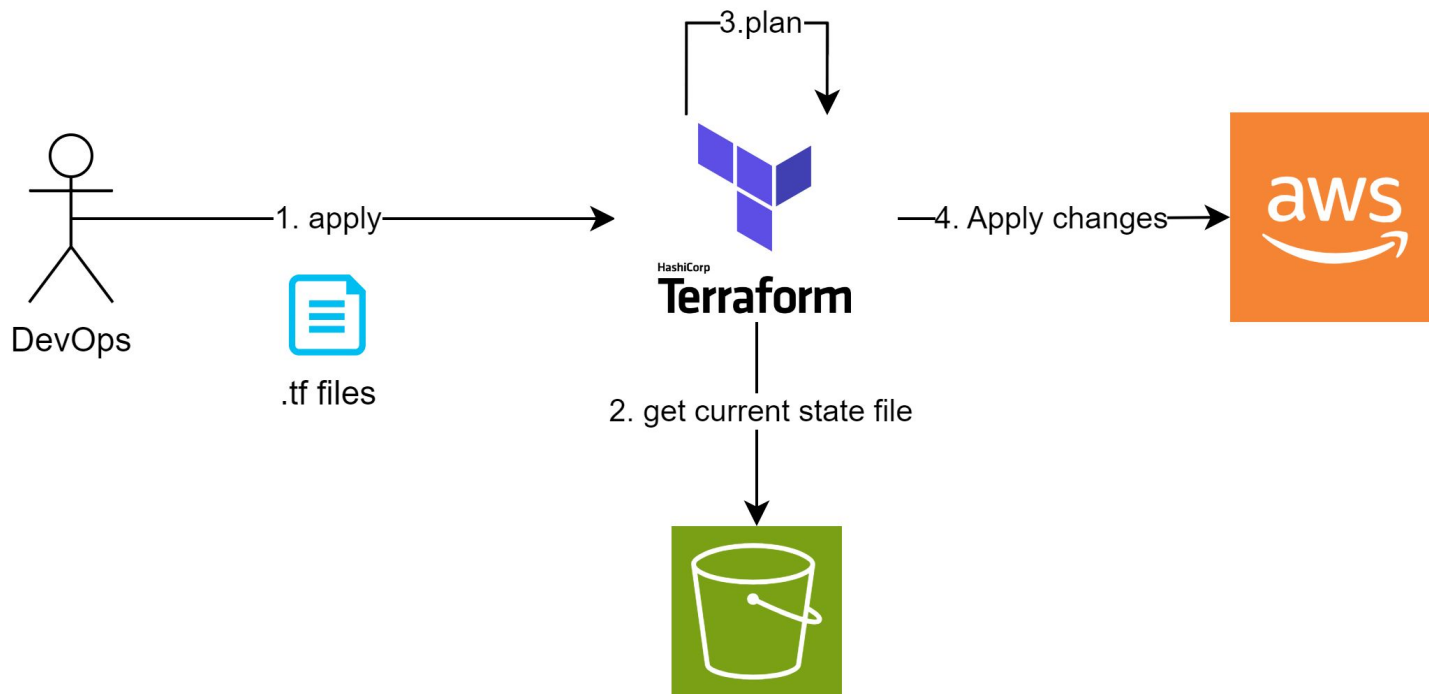


```
1 root@ip-x-x-x-x:~/workspace# terraform apply
2 data.aws_availability_zones.available: Reading...
3 data.aws_availability_zones.available: Read complete after 0s [id=us-east-1]
4
5 Terraform will perform the following actions:
6
7 # aws_vpc.vpc will be created
8 + resource "aws_vpc" "vpc" {
9     + arn                               = (known after apply)
10    + cidr_block                         = "10.0.0.0/16"
11    + default_network_acl_id            = (known after apply)
12    + default_route_table_id            = (known after apply)
13    + default_security_group_id         = (known after apply)
14    + ...
15 }
16
17 Plan: 1 to add, 0 to change, 0 to destroy.
18
19
20 Do you want to perform these actions?
21   Terraform will perform the actions described above.
22   Only 'yes' will be accepted to approve.
23
24   Enter a value:
```



# Terraform - CLI

- `$ terraform apply`



# Terraform - CLI

- **\$ terraform destroy**
  - Elimina todos los recursos gestionados por Terraform registrados en el estado



# Terraform - CLI

- **\$ terraform fmt**
  - Reescribe los ficheros de configuración para seguir el formato y estilo canónico de Terraform
- **\$ terraform validate**
  - Valida la configuración existente
  - Revisa la sintaxis y consistencia de los ficheros
  - No accede a ningún servicio remoto



# Terraform - CLI

- **\$ terraform import**

- Permite importar la infraestructura actual al fichero de estado de Terraform
- Permite gestionar recursos creados externamente en Terraform
- Para ello debemos definir nuestra infraestructura en HCL

- **Terraformer**

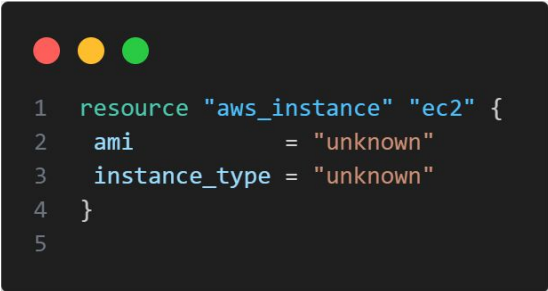
- <https://github.com/GoogleCloudPlatform/terraformer>



# Terraform - CLI

- **\$ terraform import**

- Definimos el recurso que queremos importar en HCL



```
1 resource "aws_instance" "ec2" {  
2   ami           = "unknown"  
3   instance_type = "unknown"  
4 }  
5
```

- **\$ terraform import aws\_instance.ec2 <Instance ID>**

- Incluirá la instancia en el estado de Terraform

# Terraform - CLI

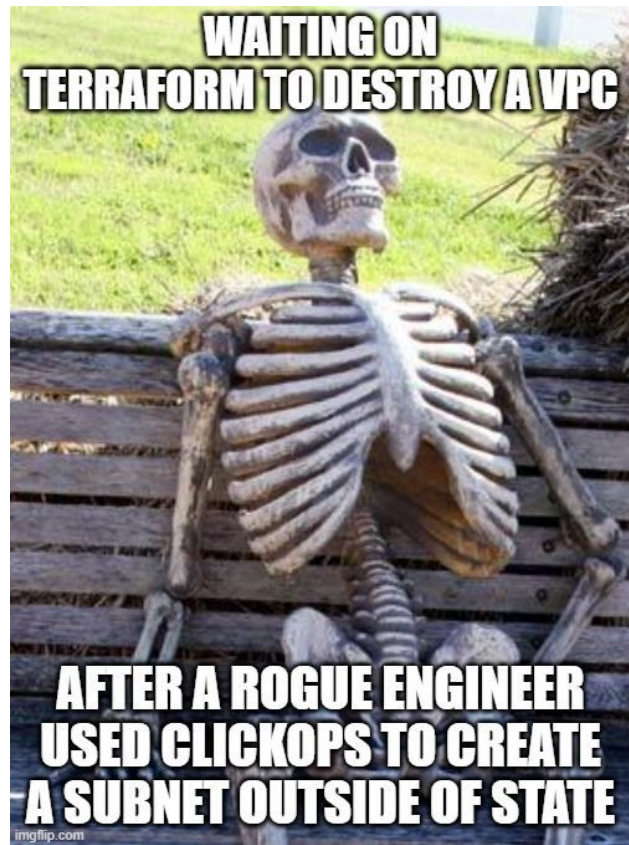
- **\$ terraform import**
  - Ejecutamos **\$ terraform apply**

```
1 # aws_instance.ec2 must be replaced
2 -/+ resource "aws_instance" "ec2" {
3     ~ ami                               = "ami-053b0d53c279acc90" -> "unknown" # forces replacement
4     ~ arn                               = "arn:aws:ec2:us-east-1:459137896070:instance/i-001e797e05799bf0b" -> (known after apply)
5     ~ associate_public_ip_address      = true -> (known after apply)
6     ~ availability_zone                 = "us-east-1c" -> (known after apply)
7     ~ cpu_core_count                    = 1 -> (known after apply)
8     ~ cpu_threads_per_core              = 1 -> (known after apply)
9     ~ disable_api_stop                  = false -> (known after apply)
10 }
```

**Ojo!** Nos pide un reemplazo ya que la definición no concuerda con la información importada  
**Debemos modificar la definición hasta que no nos pida un reemplazo**

# Terraform - CLI

- **Importante!** Evitaremos el uso de ClickOps sobre infraestructura desplegada con IaC



# Terraform - Backends

- Un backend define la localización donde Terraform guarda los ficheros de estado
- Opciones:
  - **local** (por defecto)
    - Utiliza un fichero en local (terraform.tfstate)
  - **remoto**
    - S3
    - Azure Storage
    - Google Cloud Storage
    - Terraform Cloud
- State-Lock
  - Bloquea ejecuciones simultáneas de operaciones de escritura
  - Solo si el backend lo soporta

```
1 terraform {  
2   backend "s3" {  
3     bucket = "mybucket"  
4     key    = "path/to/my/key"  
5     region = "us-east-1"  
6   }  
7 }
```



# Terraform - Providers

- Plugin que interacciona con proveedores cloud, SaaS o API's
- Las configuraciones de Terraform deben definir los proveedores como dependencias (**terraform init**)
- Se define en el fichero **providers.tf**
- Incluyen
  - Conjuntos de tipos de recursos (**resources**)
  - Fuentes de datos (**data sources**)
- Que providers tenemos disponibles?
  - **Terraform registry**

<https://registry.terraform.io/browse/providers>

Dependencias

Configuración

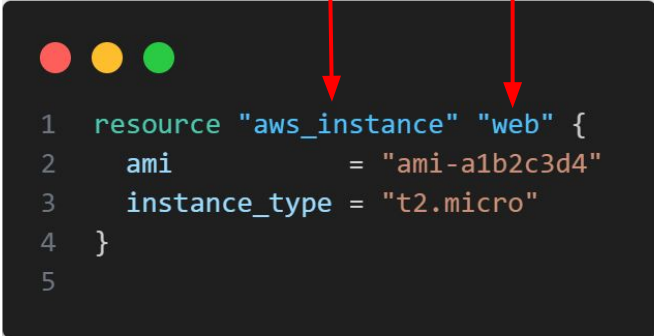
```
1 terraform {
2   required_providers {
3     aws = {
4       source = "hashicorp/aws"
5       version = "~> 4.0"
6     }
7   }
8   required_version = ">= 1.1.5"
9 }
10
11 provider "aws" {
12   region = "us-east-1"
13 }
```

# Terraform - Resources

- El elemento más importante de Terraform
- Describe uno o más objetos de infraestructura
  - VPN
  - Instancias VM
  - Elementos de alto nivel (entradas de DNS)
- Los recursos son proporcionados por los proveedores

Tipo

Nombre



```
1 resource "aws_instance" "web" {  
2     ami           = "ami-a1b2c3d4"  
3     instance_type = "t2.micro"  
4 }  
5
```

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance>

# Terraform - Resources - Meta-Arguments

- Permite cambiar el comportamiento de los recursos
- Tipos:
  - **depends\_on**: crea una dependencia entre recursos
  - **count**: permite crear múltiples instancias
  - **for\_each**: crea múltiples instancias a partir de un mapa o conjunto de strings
  - **provider**: para seleccionar un proveedor custom
  - **lifecycle**: permite cambiar la configuración del ciclo de vida
  - **provisioner**: permite definir acciones posterior a la creación del recurso

# Terraform - Resources - Meta-Arguments

- Ejemplo for\_each:

```
1 resource "aws_iam_user" "the-accounts" {  
2   for_each = toset( ["Todd", "James", "Alice", "Dottie"] )  
3   name     = each.key  
4 }  
5
```

- Ejemplo count:

```
1 resource "aws_instance" "server" {  
2   count = 4  
3   ami   = "ami-a1b2c3d4"  
4   instance_type = "t2.micro"  
5   tags = {  
6     Name = "Server ${count.index}"  
7   }  
8 }  
9
```

# Terraform - Variables

- Permite customizar parámetros de los recursos, sin tener que reescribir el código
- Permite crear código reusable de forma fácil



```
1 variable "image_id" {
2   type      = string
3   description = "The id of the machine image (AMI) to use for the server."
4   default    = "ami-0e467150185aba1a0"
5   validation {
6     condition     = length(var.image_id) > 4 && substr(var.image_id, 0, 4) == "ami-"
7     error_message = "The image_id value must be a valid AMI id, starting with \"ami-\"."
8   }
9 }
```



```
1 variable "docker_ports" {
2   type = list(object({
3     internal = number
4     external = number
5     protocol = string
6   }))
7   default = [
8     {
9       internal = 8300
10      external = 8300
11      protocol = "tcp"
12    }
13  ]
14 }
```

# Terraform - Outputs

- Devuelve información adicional de la infraestructura
- Se obtiene del resultado del comando ``terraform apply``
- También se puede extraer con el comando ``terraform output``
- Un modulo hijo puede exponer estos valores al padre
  - `module.<module_name>.<output_name>`
  - `module.web_server.instance_ip_addr`



```
1  output "instance_ip_addr" {  
2    value = aws_instance.server.private_ip  
3  }
```

# Terraform - Módulos

- Son contenedores para múltiples recursos, que se utilizan de forma conjunta
- Se trata de una colección de ficheros .tf en un mismo directorio
- Terraform trata cada directorio local como un módulo de por sí
- Estructura
  - Licencia
    - Normativa de distribución
  - Readme
    - Descripción de uso del módulo
  - main.tf
    - Configuración principal
  - variables.tf
    - Definiciones de variables
  - outputs.tf
    - Definiciones de variables de salida



# Terraform - Módulos - Ejemplo

- Definimos el módulo aws-s3-static-website-bucket
- Crea un bucket S3 y lo configuramos para servir una página web estática

```
1 # modules/aws-s3-static-website-bucket/main.tf
2
3 resource "aws_s3_bucket" "s3_bucket" {
4     bucket = var.bucket_name
5     tags = var.tags
6 }
7
8 resource "aws_s3_bucket_website_configuration" "s3_bucket" {
9     bucket = aws_s3_bucket.s3_bucket.id
10    index_document {
11        suffix = "index.html"
12    }
13    error_document {
14        key = "error.html"
15    }
16 }
17
18 resource "aws_s3_bucket_acl" "s3_bucket" {
19     bucket = aws_s3_bucket.s3_bucket.id
20     acl = "public-read"
21 }
```

```
1 # modules/aws-s3-static-website-bucket/variables.tf
2
3 variable "bucket_name" {
4     description = "Name of the s3 bucket. Must be unique."
5     type        = string
6 }
7
8 variable "tags" {
9     description = "Tags to set on the bucket."
10    type        = map(string)
11    default     = {}
12 }
```

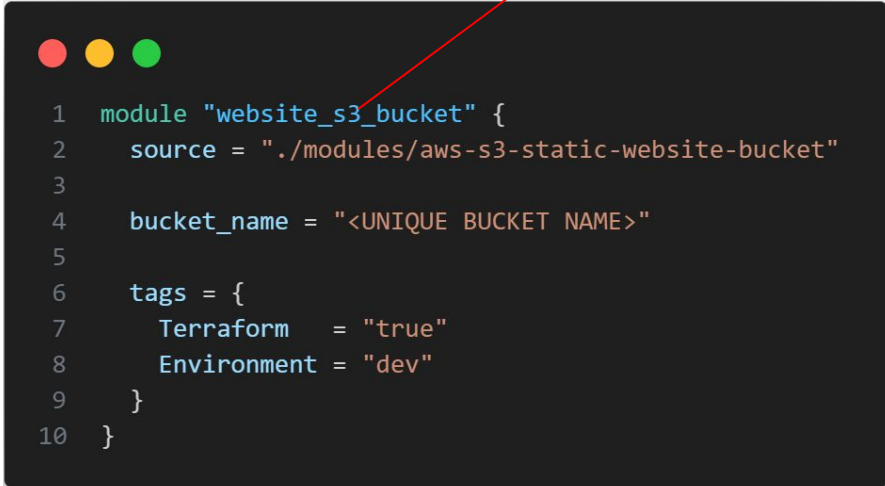


# Terraform - Módulos - Ejemplo

- Mediante la anotación `module` hacemos uso del módulo local

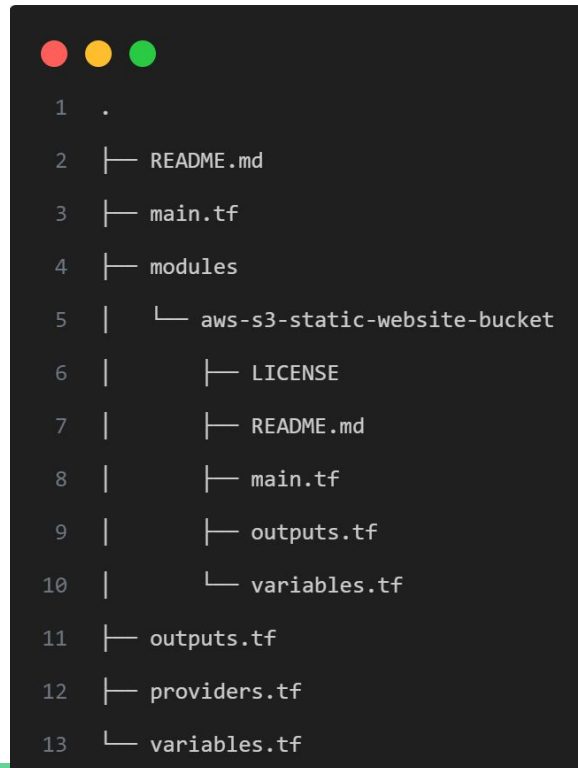
Puede ser remoto:

hashicorp/repo/s3-static-bucket  
github.com/s3-static-bucket



```
1 module "website_s3_bucket" {
2   source = "../modules/aws-s3-static-website-bucket"
3
4   bucket_name = "<UNIQUE BUCKET NAME>"
5
6   tags = {
7     Terraform = "true"
8     Environment = "dev"
9   }
10 }
```

A red arrow points from the `source` value in the Terraform code to the text "Puede ser remoto:" and the remote repository links.



```
1 .
2 |— README.md
3 |— main.tf
4 |— modules
5 |   └─ aws-s3-static-website-bucket
6 |       └─ LICENSE
7 |       └─ README.md
8 |       └─ main.tf
9 |       └─ outputs.tf
10 |       └─ variables.tf
11 |— outputs.tf
12 |— providers.tf
13 └─ variables.tf
```

# Terraform - Data source

- Nos permiten acceder a información definida fuera de Terraform
- Los providers pueden ofrecer fuentes de datos junto a los recursos

```
1 data "aws_ami" "ubuntu" {
2   most_recent = true
3   owners      = ["099720109477"] /* Ubuntu Canonical owner*/
4   filter {
5     name   = "name"
6     values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*"]
7   }
8 }
```

```
1 resource "aws_instance" "workstation" {
2   ami           = data.aws_ami.ubuntu.id
3   instance_type = "t3.medium"
4
5   subnet_id          = module.vpc.public_subnets[0]
6   vpc_security_group_ids = [aws_security_group.app_sg.id]
7   associate_public_ip_address = true
8
9   iam_instance_profile = "LabInstanceProfile"
10
11   user_data = file("userdata.sh")
12
13   root_block_device {
14     volume_size = 20
15     volume_type = "gp3"
16   }
17 }
```

# Lab30 - Declarative

- Practicar la automatización declarativa con Terraform
- Que haremos? Crearemos los mismos recursos de red que en el lab10
  - VPC
  - Subnet
  - IGW
  - Route table
  - Security group

<https://github.com/upcschool-cloud-arch/contenido/tree/main/09-gitops-terraform/labs/lab30-declarative>

# Dia 2

- Quizz
  - <https://quizizz.com/admin/quiz/664e52ee4715da41ec9f0ba2>