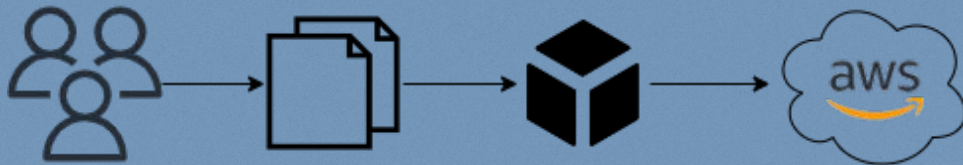


Hu Talks

Infrastructure as Code (IaC)

by Julián Tallar



August 5th, 2022



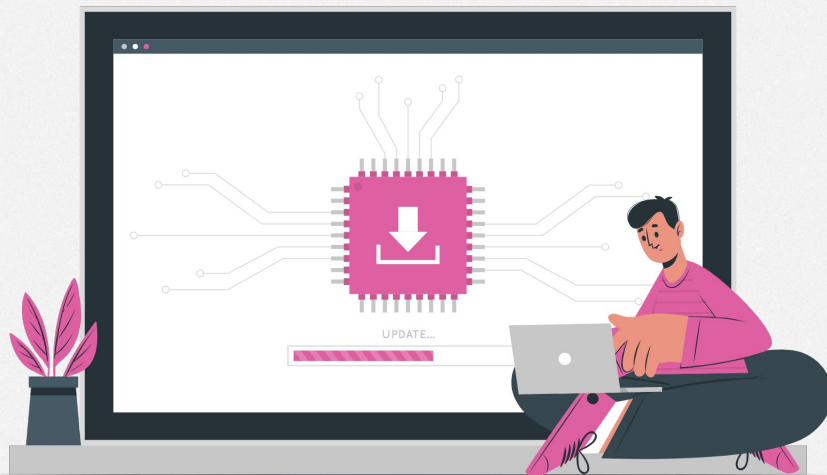
Infrastructure as Code

Idea General





LA VIDA SIN **laC**



QUÉ ES IaC

Infrastructure as Code (IaC)



Proceso de definición y configuración de infraestructura con código, en favor de su automatización.

Categorías

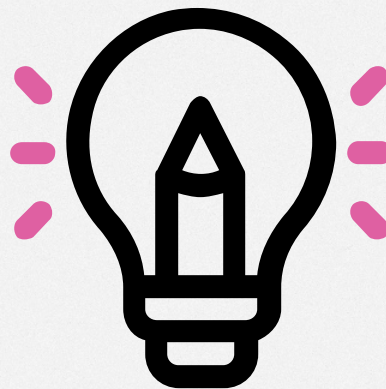


Scripts, Configuration Management (Chef, Ansible), Orchestration (Kubernetes), **Provisioning** (Terraform)

Provisioning Tools



Encargadas de aprovisionar nuevos recursos de infraestructura definidos en un template.





POR QUÉ IaC

01 Consistency & Repeatability

- Proceso automático, sin errores manuales
- Siempre que levantes un template, llegas a lo mismo
- Muy simple reutilizar o duplicar parte de la infraestructura




POR QUÉ IaC

02 Efficiency

- Al ser automatizado, los tiempos disminuyen
 - Time to Deploy
 - Change Time
- Se pierde menos tiempo en correcciones (hay menos error)

POR QUÉ IaC


03 Versioning

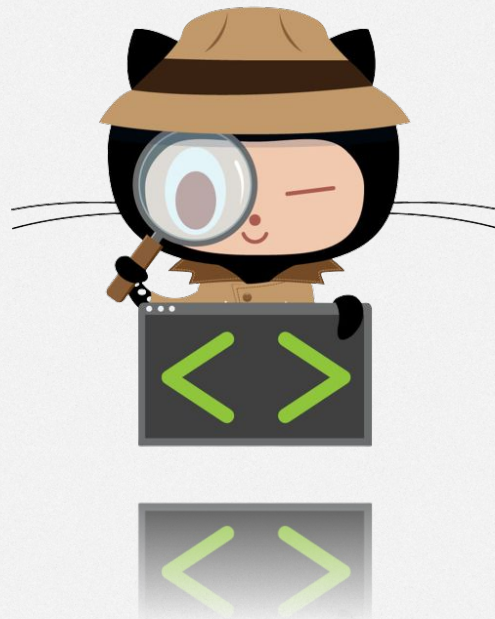
- Control de versiones de infraestructura
- CI/CD de infraestructura
- Rollback de versiones
 -  Mean time to recover



POR QUÉ IaC

04 Validation

- Pull Requests y Code reviews
- Testeos unitarios y de integración
- Menos errores debido a cambios
 -  Change Failure Rate



POR QUÉ IaC

05 Control

- Planificar antes de deploy
 - Deploys por Delta (cambios)
- Estimación de costos
- Trackeo de recursos
 - Evitar recursos “perdidos”





Terraform

Qué es Terraform





TERRAFORM

Provisioning Tool



Enfocada en aprovisionar infraestructura.

Open Source

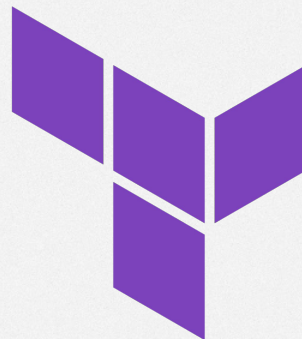


De código abierto, administrado por HashiCorp.

Cloud-agnostic



Capaz de deployar recursos en múltiples proveedores de nube.



HashiCorp
Terraform





LENGUAJE HCL

```
resource "security_group" "one" {  
  name     = "${var.name}-sg"  
  vpc_id   = local.vpc_id  
  
  tags = {  
    Name = "${var.name}-sg"  
  }  
}
```



Declarative

Escribís código con lo que querés lograr,
Terraform determinará el orden de ejecución



Limited Expressiveness

Al ser declarativo, el poder de expresión es
limitado (e.g. no poder definir funciones)





CÓMO FUNCIONA

01

PARSE CONFIGURATION

Parsea los templates para ver qué recursos crear

02

DEPENDENCY GRAPH

Determina el orden de ejecución de recursos

03

TRANSFORM TO API CALLS

Maape los recursos a API calls del proveedor

04

EXECUTE API CALLS

Realiza las API calls en tu nombre a los proveedores





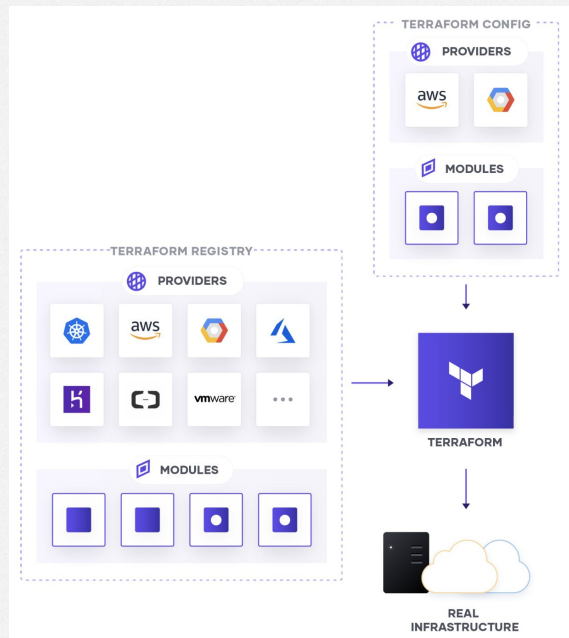
Componentes

Cómo se arma un template



Componentes - Modules

- Configuraciones reutilizables que pueden ser invocadas por otras
 - Composability
- Suelen agrupar recursos que se usan en conjunto para un proveedor
- **Registry**
 - Repo de providers y módulos
 - Público (providers y aportes de la comunidad) o privado
 - ~ Docker Hub



Componentes - File Structure

```

└─ sample
  └─ .terraform
    └─ modules
      └─ {} modules.json
    └─ providers/registry.terraform.io/hashicorp/aws/4.24.0/linux_amd64
      └─ terraform-provider-aws_v4.24.0_x5
    └─ .terraform.lock.hcl
    └─ dependencies.tf
    └─ main.tf
    └─ outputs.tf
    └─ providers.tf
    └─ ⓘ README.md
    └─ {} terraform.tfstate
    └─ terraform.tfstate.backup
    └─ variables.tf
    └─ versions.tf
```


Componentes - providers.tf

- Bloque que define el proveedor a utilizar
- Al inicializar la infraestructura, se descargará dicho proveedor
- Puede tener configuraciones generales
 - Región
 - Project ID (GCP)
 - Credenciales 😞
 - Tags comunes

```
provider "aws" {  
  region = var.region  
  default_tags {  
    tags = {  
      Author = "Terraform"  
      Region = var.region  
    }  
  }  
}
```

Componentes - versions.tf

- Especifica las versiones compatibles con el template
 - ~ package.json, pero ≠
- Al inicializarlo, lockea las versiones en un .terraform.lock.tf
- Puede tener configuraciones de Backends/Cloud
 - Lugar donde se guarda el state file

```
terraform {  
  required_version = ">= 1.0.0, < 2.0.0"  
  
  required_providers {  
    aws = {  
      source = "hashicorp/aws"  
      version = "~> 4.0"  
    }  
  }  
}
```


Componentes - variables.tf

- Parámetros de entrada de un módulo.
- Pueden recibirse por
 - Variable de entorno
 - Consola (Interactivo)
 - Valor default
- Toda variable debe tener un valor asignado
- También pueden definirse variables locales

```
# Can be read from TF_VAR_region
variable "region" {
  description = "The region to use in AWS"
  type       = string
}

variable "server_port" {
  description = "The port to serve"
  type       = number
  default    = 8080

  validation {
    condition     = var.server_port > 0
    error_message = "The server_port must be > 0."
  }
}

locals {
  db_port = 5432
}
```

Componentes - main.tf

- Recursos definidos en el módulo, o invocaciones a otros módulos
 - Configuración per sé
- Cada recurso tiene
 - Tipo de recurso
 - Nombre del recurso
 - Argumentos del recurso
- Cada módulo tiene
 - Nombre del módulo
 - Ubicación del módulo
 - Variables del módulo

```
resource "aws_security_group" "this" {  
  name     = "${var.server_name}-instance"  
  vpc_id   = module.vpc.id  
  
  tags = {  
    Name = "${var.server_name}-instance"  
  }  
}  
  
module "mi_vpc" {  
  source = "../modules/vpc"  
  
  name       = "sample-vpc"  
  cidr_block = "172.168.0.0/20"  
}
```


Componentes - dependencies.tf

- Buscar información externa al módulo en cuestión
- Ejemplos de uso
 - Obtener AMI ID de una imagen
 - Obtener información de Región o Availability Zones
 - Crear una IAM Policy
 - Obtener información de salida de otro módulo

```
data "aws_ami" "ubuntu" {  
  filter {  
    name     = "name"  
    values   = ["ubuntu-20.04-server-*"]  
  }  
  
  owners = ["099720109477"] # Canonical  
}  
  
data "aws_vpc" "default" {  
  default = true  
}  
  
data "aws_region" "current" {}
```

Componentes - outputs.tf

- Parámetros de salida de un módulo.
- Pueden ser
 - Argumentos de un recurso
 - Atributos de un recurso
 - Salidas de otros módulos
 - Valores constantes
- Se muestran en consola al aplicar (salvo que sean *sensitive*)

```
output "vpc_id" {  
  value      = module.vpc.id  
  description = "The ID of the VPC"  
}  
  
output "instance_arn" {  
  value      = aws_instance.server.arn  
  description = "The arn of the instance"  
}  
  
output "constant" {  
  value      = "Constant value"  
  description = "A constant value"  
}
```


Componentes - terraform.tfstate

- Archivo JSON que mapea recursos reales con la configuración
 - Almacena el estado actual *conocido*
- Pueden importarse recursos preexistentes al estado
`terraform import aws_instance.web i-12345678`
- 👁️ Todo se guarda en texto plano, inclusive valores *sensitive*
 - No debe almacenarse en Git
 - Debería guardarse encriptado

```
{
  "version": 4,
  "terraform_version": "1.2.6",
  "serial": 18,
  "lineage": "289aba6b-883d-eab2-b16a-4d2b486fe336",
  "outputs": {
    "instance_arn": {
      "value": "arn:aws:ec2:us-west-2:887841176879:instance/i-0be32e7a82c5c5f8c",
      "type": "string"
    },
    "instance_public_ip": {
      "value": "35.87.140.107",
      "type": "string"
    }
  },
  "resources": [
    {
      "mode": "data",
      "type": "aws_ami",
      "name": "ubuntu",
      "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "architecture": "x86_64",
            "arn": "arn:aws:ec2:us-west-2::image/ami-0aab355e1bfale72e",
            "block_device_mappings": [
              {
                "device_name": "/dev/sda1",
                "ebs": {
                  "delete_on_termination": "true",
                  "encrypted": "false",
                  "iops": "0",
                  "snapshot_id": "snap-0a0efa5c87d915063",
                  "throughput": "0",
                  "volume_size": "8",
                  "volume_type": "gp2"
                }
              }
            ]
          }
        }
      ]
    }
  ]
}
```

CLI COMMANDS

- terraform init
 - Actualizar referencias a módulos e instalar plugins (providers)
- terraform plan
 - Calcula el **delta** de recursos entre el template y el state file
- terraform apply
 - Aplicar los cambios de los templates en el proveedor
- terraform destroy
 - Destruir todos los recursos del state file
- terraform graph | dot -Tsvg > graph.svg
 - Ver el grafo de dependencias



DEMO

Terraform en acción





ALGUNAS NOTAS

- N1** Al escribir código que maneja infraestructura, un bug es más grave. Por eso se recomienda usar múltiples “mecanismos de seguridad”.
- N2** El compromiso con Terraform debe ser total (o casi). Si una parte de la infraestructura la maneja Terraform, evitar realizar cambios manuales.
- N3** Algunos servicios muy particulares de un proveedor pueden no estar 100% soportados. OJO con las versiones O.X.Y (lockearlas)
- N4** Si bien ya es un producto establecido en el mercado, hay algunas cosas en las que está algo verde (e.g. manejo de errores en algunos casos)





DESAFÍOS PENDIENTES

01

SHARED & SECURE STATE FILE

Backend remoto,
encriptado y con bloqueo

02

DIFFERENT PROVIDERS

Similitudes y diferencias
con Azure, GCP

03

PROVIDER MIGRATION

Migrando de un
proveedor a otro

04

CLOUD AGNOSTIC

Creando templates para
múltiples proveedores





Q&A





¡Gracias!

Julián Tallar





REFERENCIAS

- R1** Repo de GitHub
<https://github.com/jtallar/iac-terraform>
- R2** Terraform: Up and Running, Second Edition by Yevgeniy Brikman (O'Reilly). Copyright 2017 Yevgeniy Brikman, 978-1-492-04690-5
- R3** Mi tesis (ETA Enero 2023)
- R4** Terraform Best Practices
<https://www.terraform-best-practices.com/>

