Trabajo de Investigación

Diaz Francisco, Gonzalez Lautaro, Peirone Santiago, Zarza Valentín

Junio 2024

Sistemas Operativos II

Universidad Nacional de Noroeste de la Provincia de Buenos Aires

fdiaz892@comunidad.unnoba.edu.ar
lgonzalezbortel@comunidad.unnoba.edu.ar
sapeirone@comunidad.unnoba.edu.ar
vzarza@comunidad.unnoba.edu.ar

Actividad 9 - Terraform

UNNOBA
UNIVERSIDAD NACIONAL
NOROESTE ▪ BUENOS AIRES

¿Que es Terraform?

Terraform es una herramienta de infraestructura como código (IaaC) desarrollada por HashiCorp.

Permite crear, cambiar y versionar infraestructura de forma segura y eficiente. Esto incluye componentes de bajo nivel como instancias informáticas, almacenamiento y redes, así como componentes de alto nivel como entradas DNS y funciones SaaS.

¿Cómo funciona?

Terraform crea y administra recursos en plataformas en la nube y otros servicios a través de sus interfaces de programación de aplicaciones (API). Los proveedores permiten que Terraform funcione con prácticamente cualquier plataforma o servicio con una API accesible.

El flujo de trabajo principal de Terraform consta de tres etapas:

- **Write:**

Es la etapa donde se define los recursos, que pueden estar en varios proveedores y servicios de nube. Por ejemplo, puede crear una configuración para implementar una aplicación en máquinas virtuales en una red de nube privada virtual (VPC) con grupos de seguridad y un equilibrador de carga.

¿Cómo funciona?

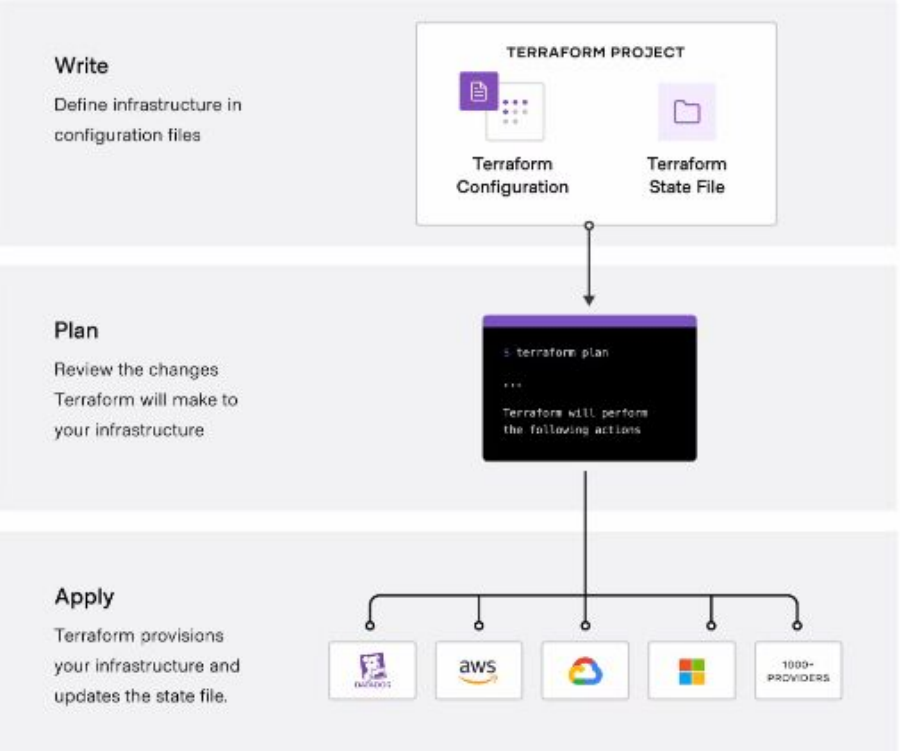
- **Plan:**

Terraform crea un plan de ejecución que describe la infraestructura que creará, actualizará o destruirá en función de la infraestructura existente y su configuración.

- **Apply:**

Tras la aprobación, Terraform ejecuta las operaciones propuestas en el orden correcto, respetando las dependencias de los recursos. Por ejemplo, si actualiza las propiedades de una VPC y cambia la cantidad de máquinas virtuales en esa VPC, Terraform recreará la VPC antes de escalar las máquinas virtuales.

Características Principales



- **Infraestructura como Código (IaaC):**

Terraform permite definir la infraestructura en archivos de texto plano usando el lenguaje de configuración de HashiCorp (HCL). Esto facilita la gestión y mantenimiento del entorno.

- **Proveedores de Servicios:**

Soporta una amplia gama de proveedores de servicios, incluidos AWS, Azure, Google Cloud, y muchos más. Esto permite gestionar recursos en múltiples entornos desde un solo lugar.

- **Planificación y Ejecución:**

Terraform proporciona una fase de "planificación" donde muestra los cambios que se realizarán sin aplicarlos. Esto permite a los usuarios revisar y confirmar cambios antes de que se implementen.

- **Gestión del Estado:**

Mantiene un estado de la infraestructura para rastrear los recursos gestionados. Este estado se puede almacenar localmente o en un backend remoto, como Amazon S3 o Terraform Cloud, para facilitar el trabajo en equipo.

- **Modularidad:**

Soporta la creación de módulos reutilizables que pueden ser compartidos y utilizados en diferentes proyectos. Esto promueve la reutilización de código y la consistencia en la implementación.

Casos de Uso

- **Aprovisionamiento de Infraestructura:**

Puedes definir y crear recursos como instancias de servidor, redes, balanceadores de carga, y bases de datos en la nube de manera automática y repetible.

- **Gestión de Configuraciones:**

Permite modificar configuraciones existentes y aplicar cambios de manera controlada.

- **Automatización y CI/CD:**

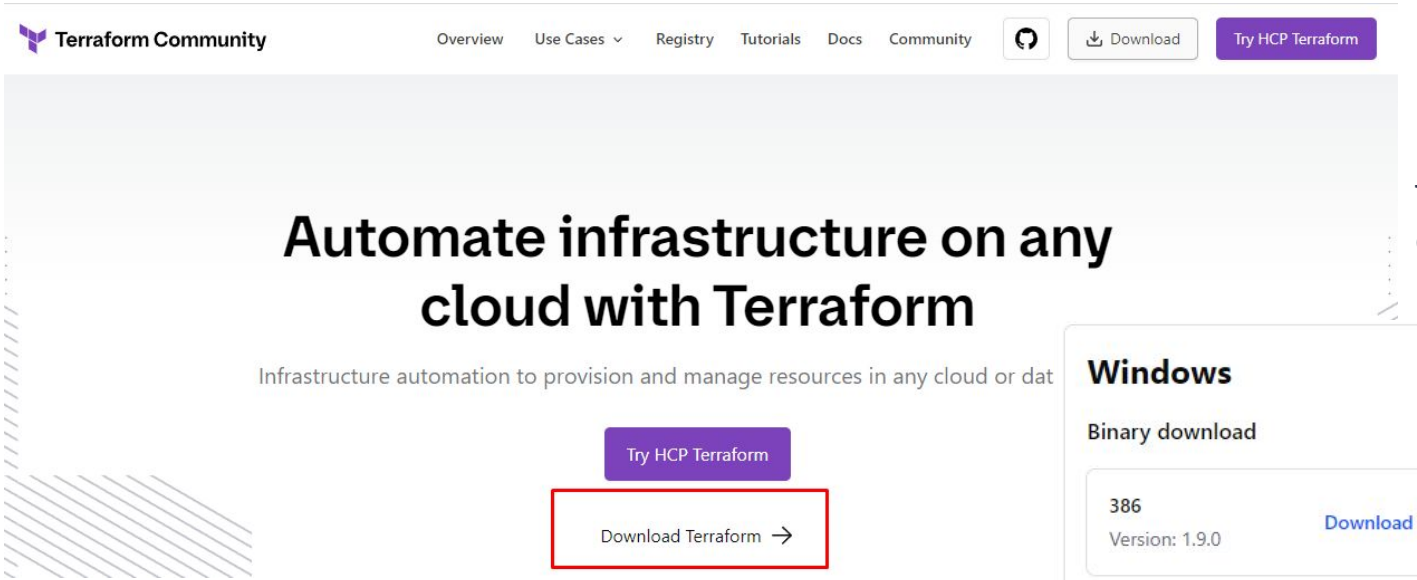
Integrar Terraform en pipelines de CI/CD (Continuous Integration/Continuous Deployment) para automatizar el despliegue de infraestructura junto con el despliegue de aplicaciones.

- **Multi-Nube y Híbrido:**

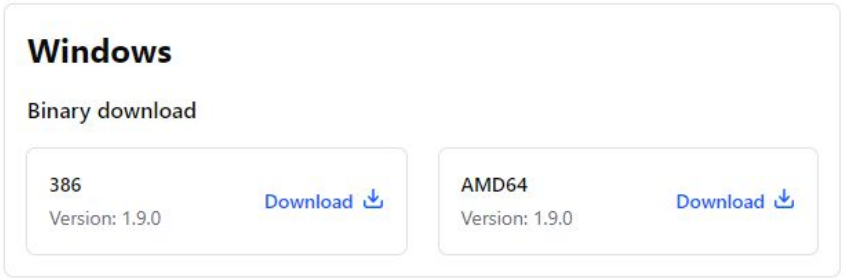
Gestionar infraestructura en múltiples proveedores de servicios en la nube y entornos híbridos desde una sola herramienta.

Instalación

En la página principal de terraform nos dirigimos al apartado de Download



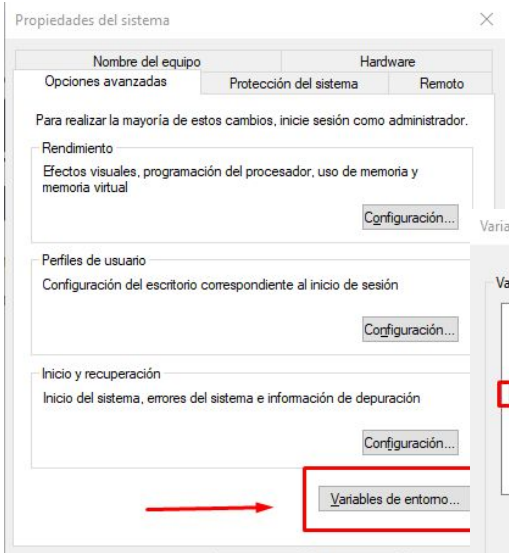
Y en nuestro caso descargamos el windows



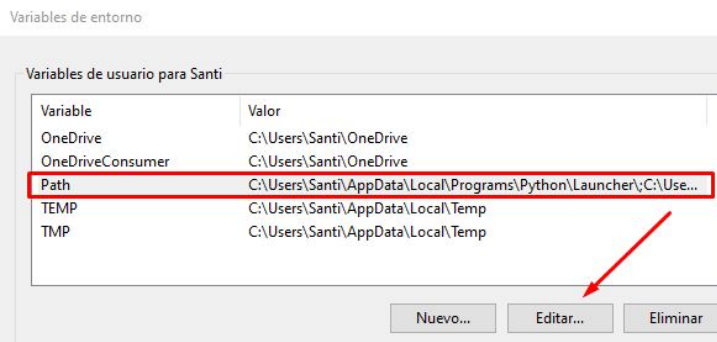
Instalación/Configuración

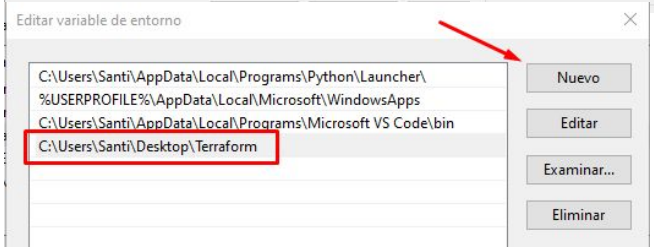
En este apartado procedemos a una configuración óptima para el uso de esta herramienta.

Nos dirigimos a “variables de entorno”



Nos dirigimos a “Path” y lo editamos



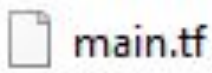


Y agregamos una nueva ruta donde tenemos el ejecutable de terraform previamente descargado

Pruebas de laboratorio

Como primer instancia debemos crear una carpeta donde se almacenará la configuración de nuestro proyecto.

Luego dentro de la misma creamos un archivo .tf (Haciendo referencia a terraform). En nuestro caso le ponemos de nombre “main”.



Y a continuación necesitaremos también de un editor de código, en nuestro caso usamos Visual Studio Code para desarrollar nuestro configuración para el lanzamiento del “Plan”.

```
provider "azurerm" {
  features {}
}
```

Proveedor y grupo de recursos

```
resource "azurerm_resource_group" "resource_group" {
  name     = "grupo-de-recursos"
  location = "brazilsouth"
}
```

Pruebas de laboratorio

Establecemos los parámetros de la red y la subnet

```
resource "azurerm_virtual_network" "virtual_network" {
  name                = "red-prueba"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.resource_group.location
  resource_group_name = azurerm_resource_group.resource_group.name
}

resource "azurerm_subnet" "subnet" {
  name                 = "subneteo-prueba"
  resource_group_name  = azurerm_resource_group.resource_group.name
  virtual_network_name = azurerm_virtual_network.virtual_network.name
  address_prefixes     = ["10.0.2.0/24"]
}
```

```
resource "azurerm_network_interface" "network_interface" {
  name                = "interface-prueba"
  location            = azurerm_resource_group.resource_group.location
  resource_group_name = azurerm_resource_group.resource_group.name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.subnet.id
    private_ip_address_allocation = "Dynamic"
  }
}
```

Configuramos una interfaz de red con una IP dinámica.

Pruebas de laboratorio

Una vez creado toda la configuración, solo queda iniciarlo. Esto lo hacemos desde nuestra PowerShell/CMD en el caso de windows, con el comando **`Terraform init`** hará un nuevo entorno de configuración o configura un entorno existente.

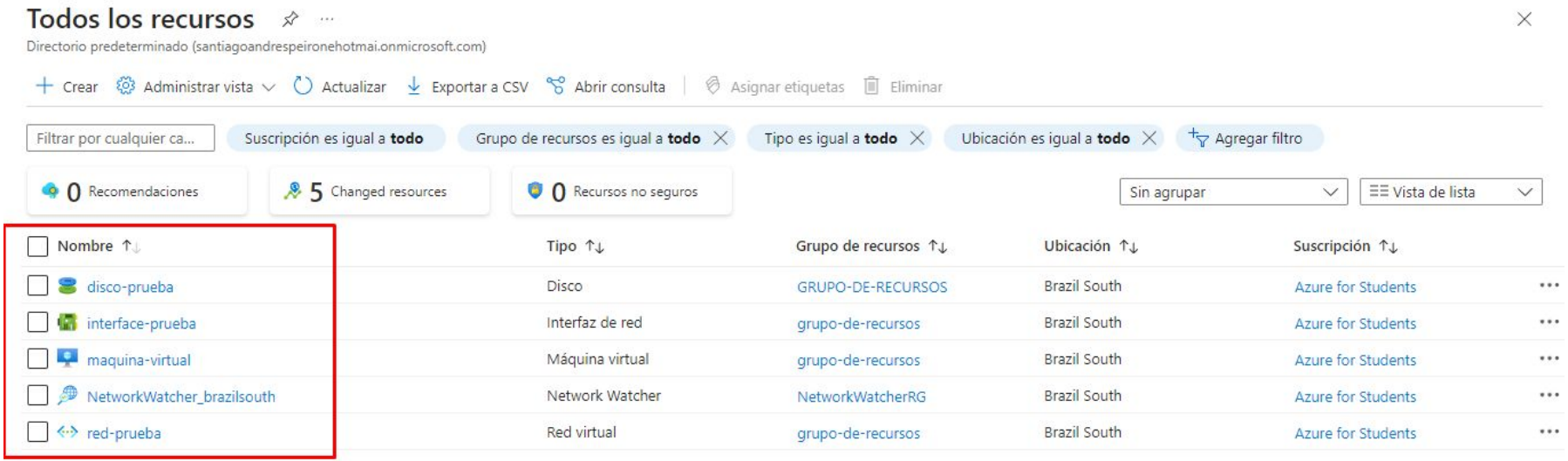
```
PS C:\Users\Santi\Desktop\Proyecto Terraform> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/azurerm...
- Installing hashicorp/azurerm v3.110.0...
- Installed hashicorp/azurerm v3.110.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
```

Posterior a eso utilizamos el siguiente comando **`terraform plan`** para generar y mostrar el plan de ejecución que describe los cambios de infraestructura definido en la configuración anterior

```
PS C:\Users\Santi\Desktop\Proyecto Terraform> terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # azurerm_network_interface.network_interface will be created
  + resource "azurerm_network_interface" "network_interface" {
      + accelerated_networking_enabled = (known after apply)
      + applied_dns_servers            = (known after apply)
      + dns_servers                    = (known after apply)
      + enable_accelerated_networking  = (known after apply)
      + enable_ip_forwarding           = (known after apply)
      + id                             = (known after apply)
      + internal_domain_name_suffix    = (known after apply)
      + ip_forwarding_enabled          = (known after apply)
      + location                       = "brazilsouth"
      + mac_address                    = (known after apply)
      + name                           = "interface-prueba"
      + private_ip_address             = (known after apply)
      + private_ip_addresses           = (known after apply)
      + resource_group_name            = "grupo-de-recursos"
      + virtual_machine_id             = (known after apply)

      + ip_configuration {
          + gateway_load_balancer_frontend_ip_configuration_id = (known after apply)
          + name                                               = "internal"
          + primary                                            = (known after apply)
          + private_ip_address                                 = (known after apply)
          + private_ip_address_allocation                      = "Dynamic"
          + private_ip_address_version                         = "IPv4"
          + subnet_id                                          = (known after apply)
        }
    }

  # azurerm_resource_group.resource_group will be created
  + resource "azurerm_resource_group" "resource_group" {
      + id       = (known after apply)
      + location = "brazilsouth"
      + name     = "grupo-de-recursos"
    }
```

Pruebas de laboratorio

Y finalmente utilizando el comando `terraform apply` toma el plan de ejecución y lo aplica para crear, actualizar o destruir los recursos según sea necesario, para alcanzar el estado deseado definido en tu configuración de Terraform.



Prueba técnica en video: https://youtu.be/XKsx2BhU-tU

Otras alternativas de Terraform

AWS CloudFormation: Funciona solamente para AWS

Pulumi: Permite usar distintos tipos de lenguaje de programación(JS, python,c#)

Kubernetes Helm: Gestiona paquetes para kubernetes

Google Cloud Deployment Manager: Funciona solamente para GCP

Diferencias entre Terraform y AWS CloudFormation

Características	Terraform	AWS CloudFormation
Proveedores soportados	Multi-cloud(AWS,Azure,GCP)	Solo AWS
Lenguaje de programación	HCL(Hashicorp Configuration Lenguaje)	JSON o YAML
Aprendizaje	Difícil ya que HCL es específico de Terraform	Más ‘fácil’ si estás familiarizado con JSON/YAML
Soporte de estado	Requiere backend para almacenar estado	Manejado automáticamente por AWS

Diferencias entre Terraform y AWS CloudFormation

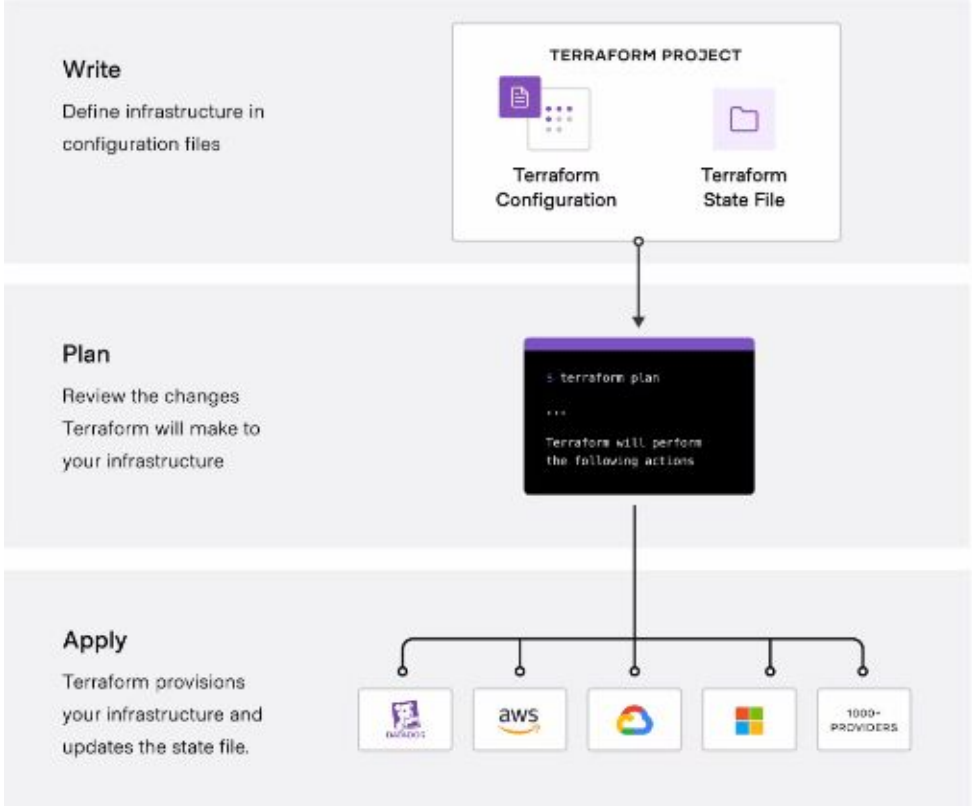
Características	Terraform	AWS CloudFormation
Documentación y comunidad	Amplia y en crecimiento, comunidad activa	Extensa documentación oficial, comunidad AWS
Evolución y soporte	Rápido desarrollo, soporte por HashiCorp y la comunidad	Soporte oficial de AWS
Costos	Open-source, sin costo adicional	Sin costo adicional, pero dentro de AWS
Integración con otras herramientas	Amplia, compatible con muchas herramientas y servicios	Mejor integración con servicios de AWS

Conclusión

Por lo tanto podemos determinar que este software es una herramienta poderosa para gestionar infraestructuras de manera declarativa y automatizada, ofreciendo gran flexibilidad y compatibilidad con diversas plataformas clouds.

Características relevantes:

- **Software open source**
- **Trabaja con 3 pasos (Write, Plan, Apply)**
- **Necesita de un proveedor (Azure, AWS, Google Cloud Platform, etc)**



Bibliografía y referencias

- https://developer.hashicorp.com/terraform/intro
- https://es.wikipedia.org/wiki/Terraform_(software)
- https://chatgpt.com
- https://www.youtube.com/watch?v=a6-G9rbF8yE&ab_channel=ElCaminoDev

Fin.