DigitalHouse>

Certified Tech Developer
The Ultimate Degree

Infraestructura II

Crear un ambiente de desarrollo con Terraform

¡Te damos la bienvenida! En este espacio vamos a poner en práctica todo lo que aprendimos durante esta semana. ¡Vamos!

Objetivo

El desafío es usar Terraform en nuestra cuenta de AWS para crear dos instancias EC2 dentro de una región específica.

Instrucciones

Durante la última guía que realizamos para Ansible, tuvimos que ingresar en nuestra computadora las credenciales de AWS transformadas en claves cifradas. Esto significa que ya tenemos gran parte del trabajo previo realizado. Pero, ¿qué nos falta?

Cuando hagamos cambios en la infraestructura usando código, vamos a necesitar un método de despliegue. En nuestro caso, necesitamos instalar "Terraform CLI" en nuestro equipo para poder ejecutar los comandos y cumplir nuestro objetivo. Tenemos que elegir la opción más adecuada para el tipo de sistema operativo que tenemos instalado. Podemos seguir la guía en el siguiente enlace:
https://learn.hashicorp.com/tutorials/terraform/install-cli

¡Ahora sí! Manos a la obra. Terraform nos permite utilizar módulos ya definidos para AWS, estos nos agilizan mucho el trabajo porque gran parte de la lógica de automatización ya se encuentra realizada.

Vamos a crear dos servidores en AWS que comparten la misma red utilizando los siguientes servicios:

- EC2
- VPC

¡Iniciemos! Lo primero que tenemos que hacer es usar la VPC por defecto dentro de nuestra cuenta de AWS y luego, crearemos nuestras instancias EC2.

Recordemos que uno de los beneficios de la infraestructura como código es el poder versionar nuestros archivos de configuración, no dudes realizarlo para mantenerlos accesibles y compartirlos con tus equipos de trabajo, sin embargo debemos agregar a nuestra archivo .gitignore la siguiente lista:

```
.terraform

.terraform.lock.hcl

terraform.tfstate

terraform.tfstate.backup
```

Ya que estos archivos pueden contener información sensible como credenciales.

Continuando con nuestra tarea, vamos a crear nuestro primer archivo de configuración para Terraform, tengamos en cuenta las extensiones que son importantes para nuestros desarrollos, ya que indicamos qué herramienta o lenguaje de programación vamos a utilizar. En un directorio o carpeta vacíos vamos a crear los siguientes archivos.

vpc.tf

Creemos el archivo “vpc.tf” dentro de nuestro directorio de trabajo:

```
resource "aws_default_vpc" "shared-vpc" {
  tags = {
    Name = "Default VPC"
  }
}
resource "aws_subnet" "my-subnet" {
  vpc_id     = aws_default_vpc.shared-vpc.id
  cidr_block = "172.31.128.0/20"
  tags = {
    Name = "grupo-x-subnet"
  }
}
```

Aquí configuramos nuestra nube virtual privada y una subred donde que luego usaran nuestras instancias EC2.

ec2.tf

En nuestro segundo archivo de configuración vamos a definir dos instancias EC2 conectadas compartiendo la misma VPC y un grupo de seguridad para definir los accesos a nivel de red.

```
module "ec2_cluster" {
  source                 = "terraform-aws-modules/ec2-instance/aws"
  version                = "~> 2.0"
  name                   = "grupo-x-maquinavirtual"
  instance_count         = 2
  ami                    = "ami-04505e74c0741db8d"
  instance_type          = "t2.micro"
  vpc_security_group_ids =
[module.ssh_security_group.this_security_group_id]
  subnet_ids             = [ aws_subnet.my-subnet.id ]
  tags = {
    Terraform   = "true"
    Environment = "dev"
 }
}

module "ssh_security_group" {
  source      = "terraform-aws-modules/security-group/aws//modules/ssh"
  version     = "~> 3.0"
  name        = "ssh-server"
  description = "Grupo de seguridad para server ssh y puerto ssh
(22)abiertos"
  vpc_id      = aws_default_vpc.shared-vpc.id
  ingress_cidr_blocks = ["172.31.0.0/16"]
}
```

main.tf

Por último, como cualquier proyecto de código, necesitamos nuestro punto de entrada a la aplicación con sus definiciones globales. En nuestro caso este “main.tf” va a contener la dirección de nuestras credenciales de AWS y la región donde las vamos a utilizar.

```
provider "aws" {
 shared_credentials_files = [ "~/.aws/credentials" ]
 region = "us-east-1"
}
```

El parámetro “shared_credentials_files” tendrá como valor el archivo con las credenciales de AWS en la computadora de cada uno, el que usamos en este ejemplo es la ruta por defecto, donde usualmente se almacenan.

¡Pasamos a la acción!

Los comandos de Terraform

terraform init

Cuando tenemos un proyecto nuevo en Terraform tenemos que inicializarlo, esto se realiza con el comando “terraform init”. La salida del comando debe ser algo similiar a esto:

```
[enuel@enuel digitalhouse]$ terraform init
Initializing modules...
Downloading terraform-aws-modules/ec2-instance/aws 2.19.0 for ec2_cluster...
- ec2_cluster in .terraform/modules/ec2_cluster
Downloading terraform-aws-modules/security-group/aws 3.18.0 for ssh_security_group...
- ssh_security_group in .terraform/modules/ssh_security_group/modules/ssh
- ssh_security_group.sg in .terraform/modules/ssh_security_group
Downloading terraform-aws-modules/vpc/aws 3.2.0 for vpc...
- vpc in .terraform/modules/vpc

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching ">= 2.42.0, >= 3.15.0, >= 3.24.0"...
- Installing hashicorp/aws v3.51.0...
- Installed hashicorp/aws v3.51.0 (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.
```

El texto final en color verde nos indica que fue inicializado correctamente.

terraform validate

Luego de inicializar podemos verificar la sintaxis y demás configuración locales con “terraform validate”

terraform plan

Nos permite previsualizar los cambios que vamos a hacer en la infraestructura, en particular resalta los recursos a crear y destruir sin ejecutar ningún cambio real.

```
[user@vm]$ 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:

  # aws_default_vpc.shared-vpc will be created
  + resource "aws_default_vpc" "shared-vpc" {
      + arn                                  = (known after apply)
      + cidr_block                           = (known after apply)
      + default_network_acl_id               = (known after apply)
      + default_route_table_id               = (known after apply)
      + default_security_group_id            = (known after apply)
      + dhcp_options_id                      = (known after apply)
      + enable_classiclink                   = (known after apply)
      + enable_classiclink_dns_support       = (known after apply)
      + enable_dns_hostnames                 = true
      + enable_dns_support                   = true
      + existing_default_vpc                 = (known after apply)
      + force_destroy                        = false
      + id                                   = (known after apply)
      + instance_tenancy                     = (known after apply)
      + ipv6_association_id                  = (known after apply)
      + ipv6_cidr_block                      = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + tags                                 = {
          + "Name" = "Default VPC"
        }
      + tags_all                             = {
          + "Name" = "Default VPC"
        }
    }

...

Plan: 8 to add, 0 to change, 0 to destroy.
```

terraform apply

El siguiente paso es aplicar la configuración terraform que hemos creado usando el comando “terraform apply”, este se encargará de aplicar los cambios en nuestra cuenta de AWS, de forma remota y desde la comodidad de nuestro PC. Este proceso puede tardar. La demora varía según el tiempo que necesite AWS para crear los recursos y según nuestra conexión de Internet.

En la salida del comando hay un dato a destacar: la cantidad de recursos que va a crear (add), lo que va a modificar (change) y lo que tiene que borrar (destroy):

```
Plan: 26 to add, 0 to change, 0 to destroy.
```

Una vez que nos detalle en una gran lista todo lo que va a realizar, nos pide una confirmación. Vamos a ingresar “yes”.

```
Plan: 26 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

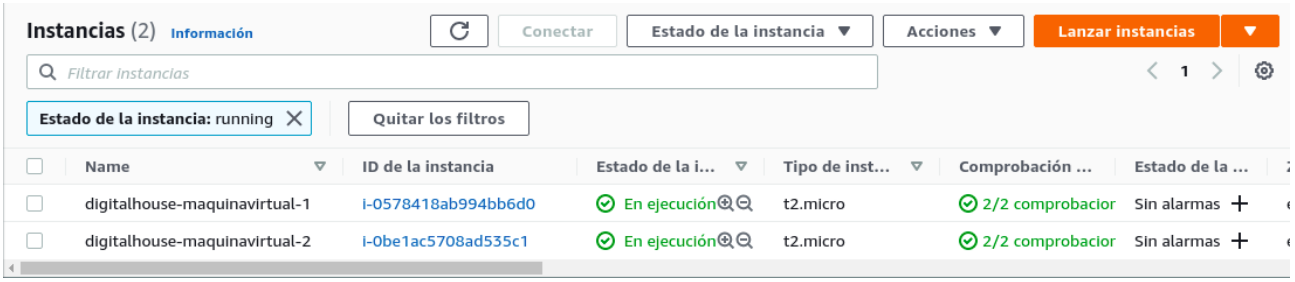
Luego, vemos cómo la salida del comando nos muestra en tiempo real la creación de los recursos:

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

module.vpc.aws_eip.nat[1]: Creating...
module.vpc.aws_vpc.this[0]: Creating...
module.vpc.aws_eip.nat[0]: Creating...
module.vpc.aws_eip.nat[0]: Creation complete after 3s [id=eipalloc-0c6ea6668e8410eae]
module.vpc.aws_eip.nat[1]: Creation complete after 3s [id=eipalloc-01ae76653708644f5]
module.vpc.aws_vpc.this[0]: Still creating... [10s elapsed]
module.vpc.aws_vpc.this[0]: Creation complete after 13s [id=vpc-0043735802e80c005]
module.vpc.aws_internet_gateway.this[0]: Creating...
module.vpc.aws_subnet.public[1]: Creating...
module.vpc.aws_subnet.private[1]: Creating...
module.vpc.aws_route_table.private[0]: Creating...
module.vpc.aws_route_table.public[0]: Creating...
module.vpc.aws_route_table.private[1]: Creating...
module.vpc.aws_subnet.private[0]: Creating...
module.vpc.aws_subnet.public[0]: Creating...
```

Al terminar, nos indica el estado final de nuestra infraestructura con un resumen que debe coincidir con el informado antes de ingresar “yes”:

```
module.vpc.aws_nat_gateway.this[0]: Still creating... [1m10s elapsed]
module.vpc.aws_nat_gateway.this[1]: Still creating... [1m10s elapsed]
module.vpc.aws_nat_gateway.this[0]: Still creating... [1m20s elapsed]
module.vpc.aws_nat_gateway.this[1]: Still creating... [1m20s elapsed]
module.vpc.aws_nat_gateway.this[1]: Still creating... [1m30s elapsed]
module.vpc.aws_nat_gateway.this[0]: Still creating... [1m30s elapsed]
module.vpc.aws_nat_gateway.this[0]: Still creating... [1m40s elapsed]
module.vpc.aws_nat_gateway.this[1]: Still creating... [1m40s elapsed]
module.vpc.aws_nat_gateway.this[0]: Still creating... [1m50s elapsed]
module.vpc.aws_nat_gateway.this[1]: Still creating... [1m50s elapsed]
module.vpc.aws_nat_gateway.this[1]: Creation complete after 1m52s [id=nat-0473da0778adfb6b3]
module.vpc.aws_nat_gateway.this[0]: Creation complete after 1m52s [id=nat-0900ca8a6dd115399]
module.vpc.aws_route.private_nat_gateway[1]: Creating...
module.vpc.aws_route.private_nat_gateway[0]: Creating...
module.vpc.aws_route.private_nat_gateway[1]: Creation complete after 5s [id=r-rtb-00524ad475278b2041080289494]
module.vpc.aws_route.private_nat_gateway[0]: Creation complete after 5s [id=r-rtb-07f739de5607760701080289494]

Apply complete! Resources: 26 added, 0 changed, 0 destroyed.
```

¡Fue un éxito! Nuestras dos instancias fueron creadas con todos los recursos en su contexto para que funcionen correctamente:



terraform destroy

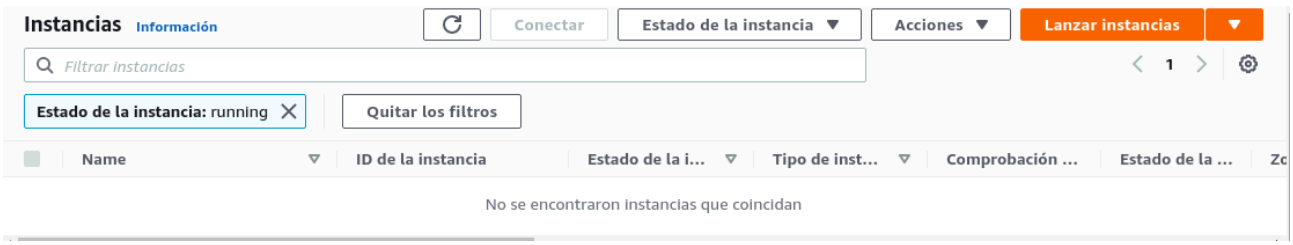
En el caso que dejemos de utilizar nuestra infraestructura, solo basta con ejecutar el comando para borrar todo: “terraform destroy”.

Como vemos en la salida, el resumen del plan se modifica informando que todo está en condiciones de borrarse:

```
Plan: 0 to add, 0 to change, 26 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

module.vpc.aws_route_table_association.private[1]: Destroying... [id=rtbassoc-0f3e3a2d33db505a1]
module.vpc.aws_route_table_association.public[1]: Destroying... [id=rtbassoc-0dfb426c335a5fc16]
module.vpc.aws_route_table_association.public[0]: Destroying... [id=rtbassoc-0ec1964145ab2ff46]
module.ssh_security_group.module.sg.aws_security_group_rule.ingress_rules[0]: Destroying... [id=sgrule-2140067553]
module.vpc.aws_route.private_nat_gateway[0]: Destroying... [id=r-rtb-07f739de5607760701080289494]
module.vpc.aws_route_table_association.private[0]: Destroying... [id=rtbassoc-0c55689a57025b1d9]
module.ssh_security_group.module.sg.aws_security_group_rule.egress_rules[0]: Destroying... [id=sgrule-718074897]
module.vpc.aws_route.private_nat_gateway[1]: Destroying... [id=r-rtb-00524ad475278b2041080289494]
module.vpc.aws_route.public_internet_gateway[0]: Destroying... [id=r-rtb-06a63e4f76701b0bc1080289494]
module.ec2_cluster.aws_instance.this[0]: Destroying... [id=i-0578418ab994bb6d0]
module.vpc.aws_route_table_association.private[1]: Destruction complete after 2s
```

Al finalizar, vemos que nuestros recursos fueron eliminados con éxito. Lo podemos verificar en la salida del comando y en nuestra cuenta de AWS.

```
module.vpc.aws_internet_gateway.this[0]: Destroying... [id=igw-0755e83b90b764dc8]
module.vpc.aws_subnet.public[1]: Destroying... [id=subnet-08a98721e52718d95]
module.vpc.aws_eip.nat[1]: Destroying... [id=eipalloc-01ae76653708644f5]
module.vpc.aws_eip.nat[0]: Destroying... [id=eipalloc-0c6ea6668e8410eae]
module.vpc.aws_subnet.public[0]: Destroying... [id=subnet-0e35369e8d34fa577]
module.vpc.aws_subnet.public[1]: Destruction complete after 2s
module.vpc.aws_eip.nat[1]: Destruction complete after 2s
module.vpc.aws_eip.nat[0]: Destruction complete after 2s
module.vpc.aws_subnet.public[0]: Destruction complete after 2s
module.vpc.aws_internet_gateway.this[0]: Still destroying... [id=igw-0755e83b90b764dc8, 10s elapsed]
module.vpc.aws_internet_gateway.this[0]: Destruction complete after 12s
module.vpc.aws_vpc.this[0]: Destroying... [id=vpc-0043735802e80c005]
module.vpc.aws_vpc.this[0]: Destruction complete after 1s

Destroy complete! Resources: 26 destroyed.
```



Conclusión

Durante el ejercicio, aprendimos que Terraform nos permite agilizar la creación de nuestra infraestructura. Nuestro código no es extenso, hicimos uso de módulos para que nuestro código sea declarativo y legible para nuestros compañeros de equipo.

¿Podrías crear una base de datos RDS que esté conectada a las instancias EC2? Tenemos a nuestra disposición la registry de Terraform para buscar el módulo más adecuado para realizar esa tarea.

En el siguiente enlace podemos encontrar los módulos que utilizamos para este ejercicio guiado, así como más módulos para AWS:

https://registry.terraform.io/providers/hashicorp/aws/latest