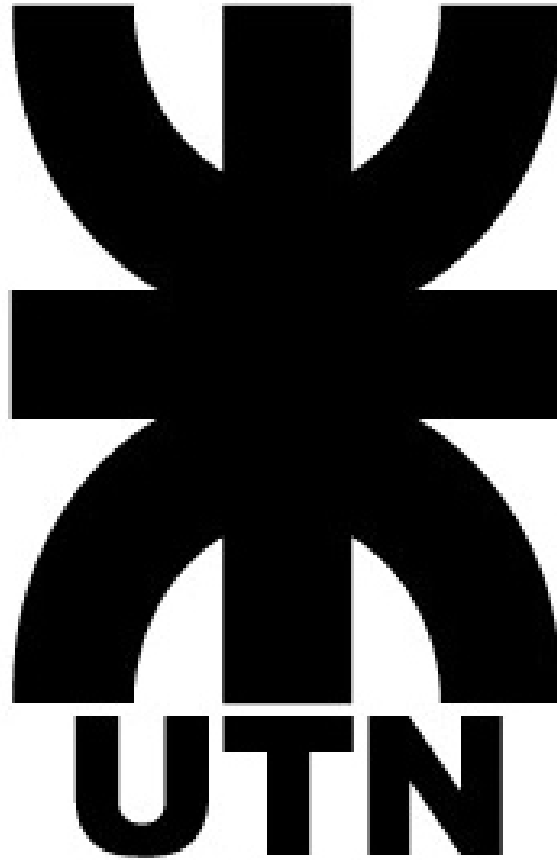


# Redes de información



## **Trabajo práctico N°4 Desafío Terraform**

Integrantes:

Fernandez Juan Carlos 19878

Garcia Emmanuel 22294

Gasparutti Edgardo 21918

Retamozo Agustin 19704

Sabo Carlos Leandro Gabriel 18106

Sanchez Negrette Luis A. 18536

Vidal Raul 16660

Villegas Cesar 18130

Universidad Tecnológica Nacional

22/11/2021

<b>Introducción</b>	<b>3</b>
<b>Desarrollo</b>	<b>3</b>
Creación y configuración del repositorio	3
Archivos necesarios	4
Comparación con el repositorio usado para otro cliente	5
variables.tf	6
key_pair.tf	6
networking.tf	6
security_groups.tf	7
instance.tf	8
provider.tf, output.tf y user-data.sh	9
Ejecución del workflow	10
Diagrama de la infraestructura	10
<b>Conclusión</b>	<b>10</b>

# Introducción

En este trabajo implementaremos un Proof of Concept para una start up, que nuestra empresa tiene como cliente, utilizando una infraestructura como código (IaC), mediante Terraform, que es una infraestructura Open-Source que nos provee un CLI workflow consistente con una configuración declarativa de archivos. El despliegue es de forma automatizada mediante GitHub Actions en los servidores de AWS.

## Desarrollo

### Creación y configuración del repositorio

En primer lugar, creamos un nuevo repositorio de GitHub, con visibilidad privada y un archivo `.gitignore`, usando la plantilla Terraform.

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

**Repository template**  
Start your repository with a template repository's contents.

No template ▾

**Owner \*** **Repository name \***

egarcia1997 ▾ /

Great repository names are short and memorable. Need inspiration? How about [ubiquitous-octo-invention?](#)

**Description (optional)**

☐ **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☒ **Private**  
You choose who can see and commit to this repository.

**Initialize this repository with:**  
Skip this step if you're importing an existing repository.

☐ **Add a README file**  
This is where you can write a long description for your project. [Learn more.](#)

☒ **Add .gitignore**  
Choose which files not to track from a list of templates. [Learn more.](#)

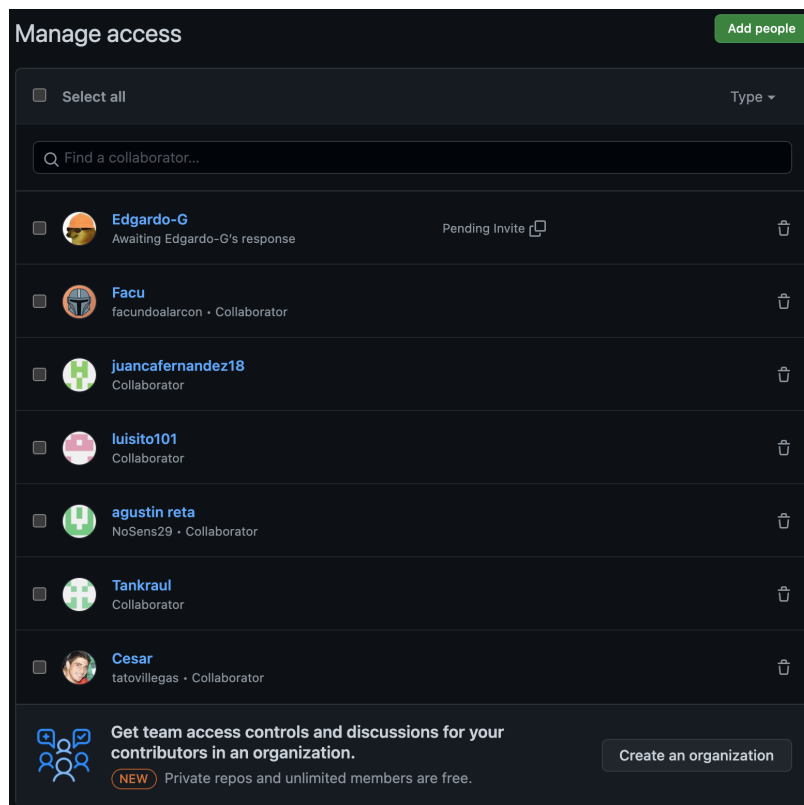
.gitignore template: Terraform ▾

☐ **Choose a license**  
A license tells others what they can and can't do with your code. [Learn more.](#)

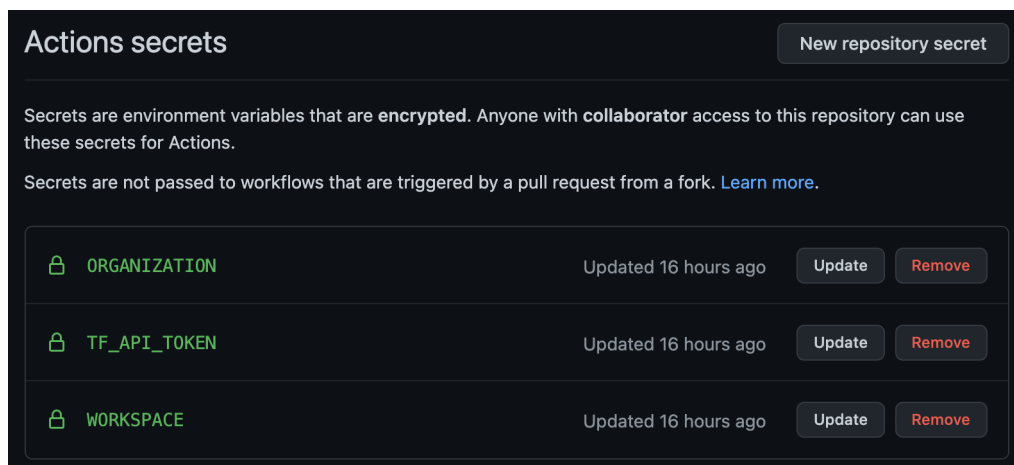
This will set `main` as the default branch. Change the default name in your [settings](#).

Create repository

Luego, agregamos a todos los stakeholders necesarios, es decir, los integrantes de nuestro grupo y el profesor Facundo Alarcón:

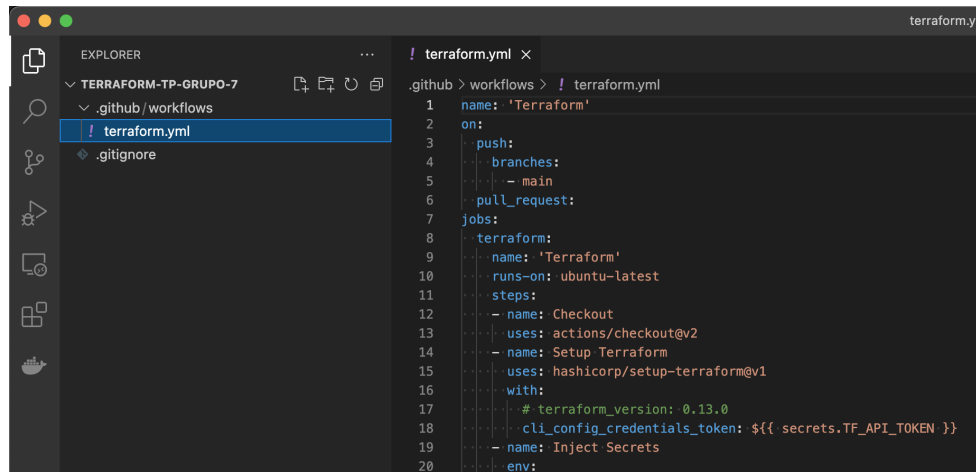


A continuación creamos los tres secretos solicitados: ORGANIZATION, TF\_API\_TOKEN y WORKSPACE.



## Archivos necesarios

En primer lugar, se nos solicitó copiar el archivo `.github/workflows/terraform.yml`, el cual sirve para implementar un workflow en GitHub Actions que hace uso de Terraform Cloud para desplegar la infraestructura en una VPC de AWS.

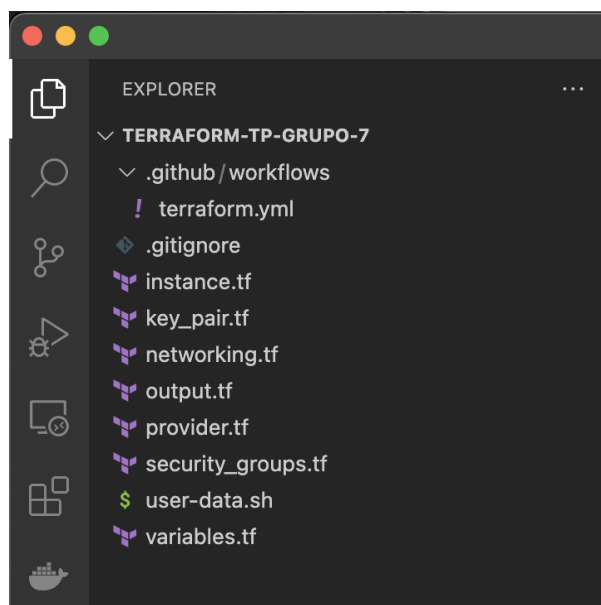


```
1 name: 'Terraform'
2 on:
3   push:
4     branches:
5       - main
6   pull_request:
7 jobs:
8   terraform:
9     name: 'Terraform'
10    runs-on: ubuntu-latest
11    steps:
12      - name: Checkout
13        uses: actions/checkout@v2
14      - name: Setup Terraform
15        uses: hashicorp/setup-terraform@v1
16        with:
17          # terraform_version: 0.13.0
18          cli_config_credentials_token: ${{ secrets.TF_API_TOKEN }}
19      - name: Inject Secrets
20        env:
```

Luego se nos proveyó de un repositorio con un código de ejemplo, el cual fue usado en un proyecto anterior. Dicho repositorio incluía los siguientes archivos:

- instance.tf
- key\_pair.tf
- networking.tf
- output.tf
- provider.tf
- security\_groups.tf
- user-data.sh
- variables.tf

Los cuales fueron copiados en nuestro repositorio.



## Comparación con el repositorio usado para otro cliente

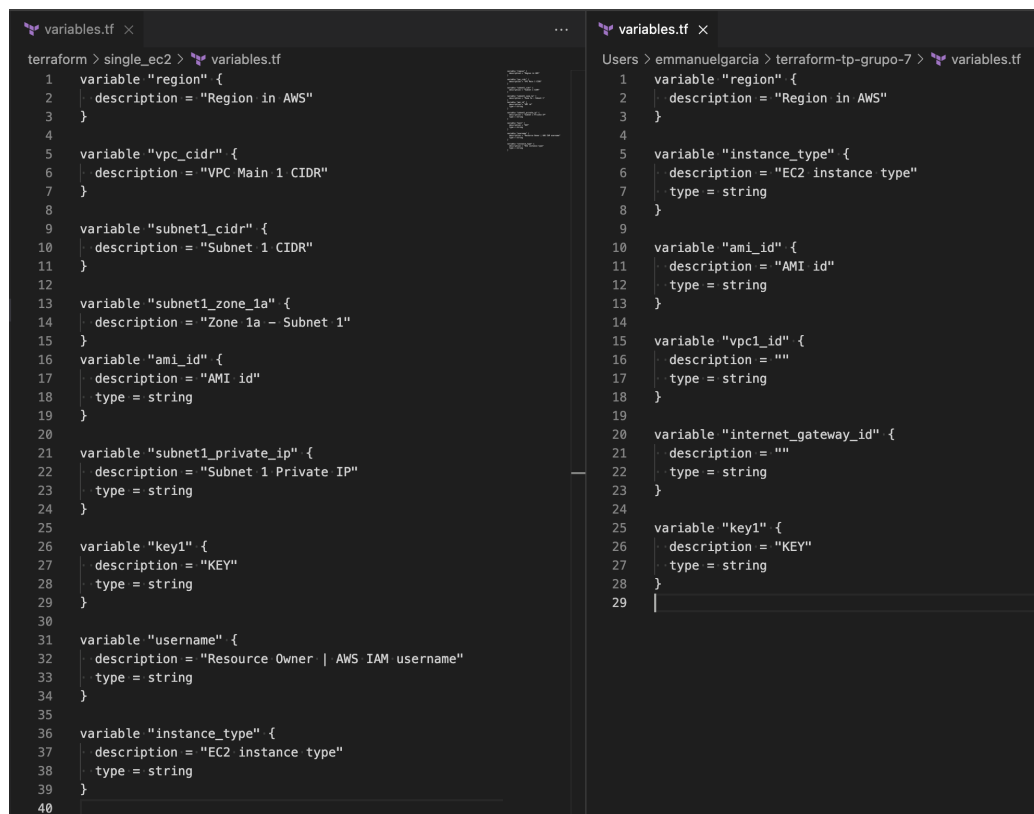
Los cambios que hicimos son los siguientes:

variables.tf

En este archivo se encuentran las variables de Terraform Cloud que serán utilizadas en el workflow. El escenario informa que tenemos acceso a las siguientes variables:

- region
- instance\_type
- ami\_id
- vpc1\_id
- internet\_gateway\_id
- key1

Como dicho archivo difería de lo solicitado, eliminamos las variables `vpc_cidr`, `subnet1_cidr`, `subnet1_zone_1a`, `subnet1_private_ip` y `username`, ya que no eran necesarias, y agregamos `vpc1_id` e `internet_gateway_id`.



key\_pair.tf

Este archivo se eliminó, ya que tenemos disponible la key en la variable `var.key1`.

networking.tf

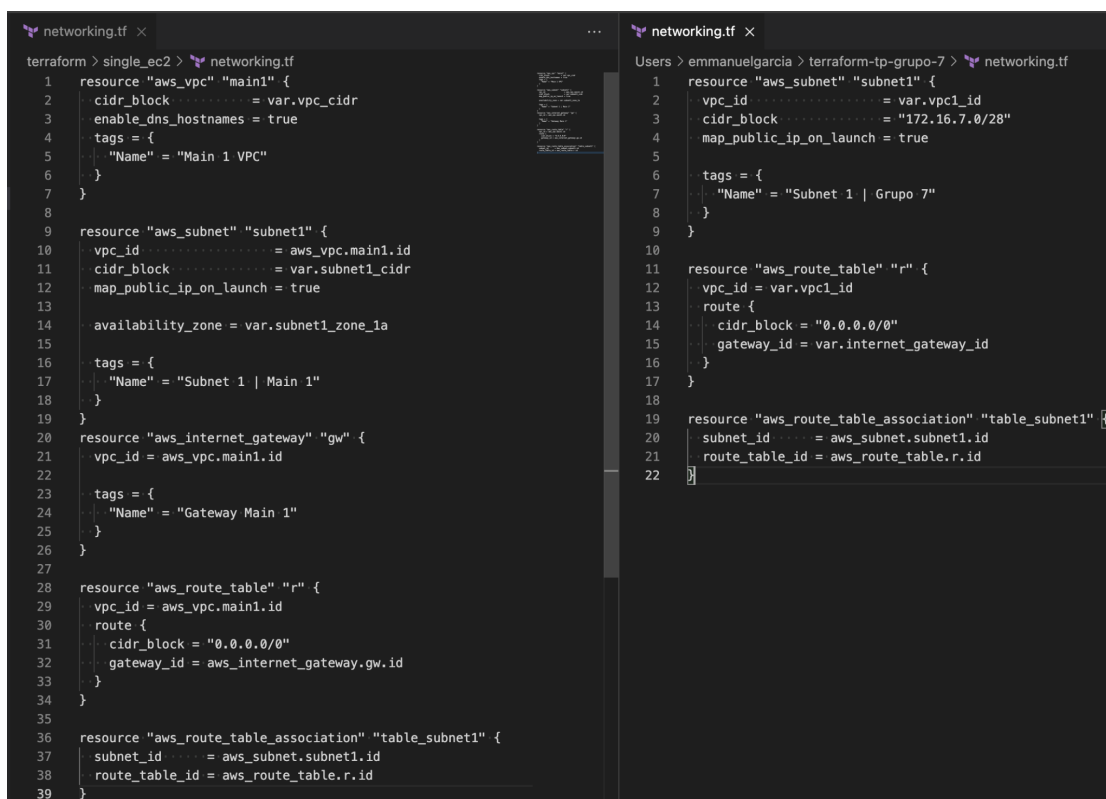
En este archivo se declaran los recursos de red que se crearán. En el archivo original se creaban una VPC, una subnet, un gateway y una tabla de ruteo, y se asociaba la subnet con la tabla de ruteo.

Como la VPC y el gateway ya estaban creados (y teníamos acceso a sus IDs en las variables), se eliminaron los recursos "aws\_vpc" "main1" y "aws\_internet\_gateway" "gw".

En la creación de la subnet modificamos la VPC a la que pertenece mediante el atributo `vpc_id`, asignándole la variable `var.vpc1_id`. También se cambió el rango a utilizar mediante el atributo `cidr_block`, asignándole el valor "172.16.7.0/28", el cual debía estar contenido en el rango de la VPC (172.16.0.0/16). Además, se agregó nuestro número de grupo al tag `Name`.

En la creación de la tabla de ruteo se cambiaron los ID de la VPC y el gateway a los que pertenece, mediante los atributos `vpc_id` y `gateway_id`, a los que se les asignaron las variables `var.vpc1_id` y `var.internet_gateway_id`.

La asociación de la subnet con la tabla de ruteo se dejó sin modificaciones.



```
networking.tf
terraform > single_ec2 > networking.tf
1 resource "aws_vpc" "main1" {
2   cidr_block = var.vpc_cidr
3   enable_dns_hostnames = true
4   tags = {
5     "Name" = "Main 1 VPC"
6   }
7 }
8
9 resource "aws_subnet" "subnet1" {
10  vpc_id = aws_vpc.main1.id
11  cidr_block = var.subnet1_cidr
12  map_public_ip_on_launch = true
13
14  availability_zone = var.subnet1_zone_1a
15
16  tags = {
17    "Name" = "Subnet 1 | Main 1"
18  }
19 }
20
21 resource "aws_internet_gateway" "gw" {
22  vpc_id = aws_vpc.main1.id
23
24  tags = {
25    "Name" = "Gateway Main 1"
26  }
27 }
28
29 resource "aws_route_table" "r" {
30  vpc_id = aws_vpc.main1.id
31  route {
32    cidr_block = "0.0.0.0/0"
33    gateway_id = aws_internet_gateway.gw.id
34  }
35 }
36
37 resource "aws_route_table_association" "table_subnet1" {
38  subnet_id = aws_subnet.subnet1.id
39  route_table_id = aws_route_table.r.id
40 }
```

```
networking.tf
Users > emmanuelgarcia > terraform-tp-grupo-7 > networking.tf
1 resource "aws_subnet" "subnet1" {
2   vpc_id = var.vpc1_id
3   cidr_block = "172.16.7.0/28"
4   map_public_ip_on_launch = true
5
6   tags = {
7     "Name" = "Subnet 1 | Grupo 7"
8   }
9 }
10
11 resource "aws_route_table" "r" {
12  vpc_id = var.vpc1_id
13  route {
14    cidr_block = "0.0.0.0/0"
15    gateway_id = var.internet_gateway_id
16  }
17 }
18
19 resource "aws_route_table_association" "table_subnet1" {
20  subnet_id = aws_subnet.subnet1.id
21  route_table_id = aws_route_table.r.id
22 }
```

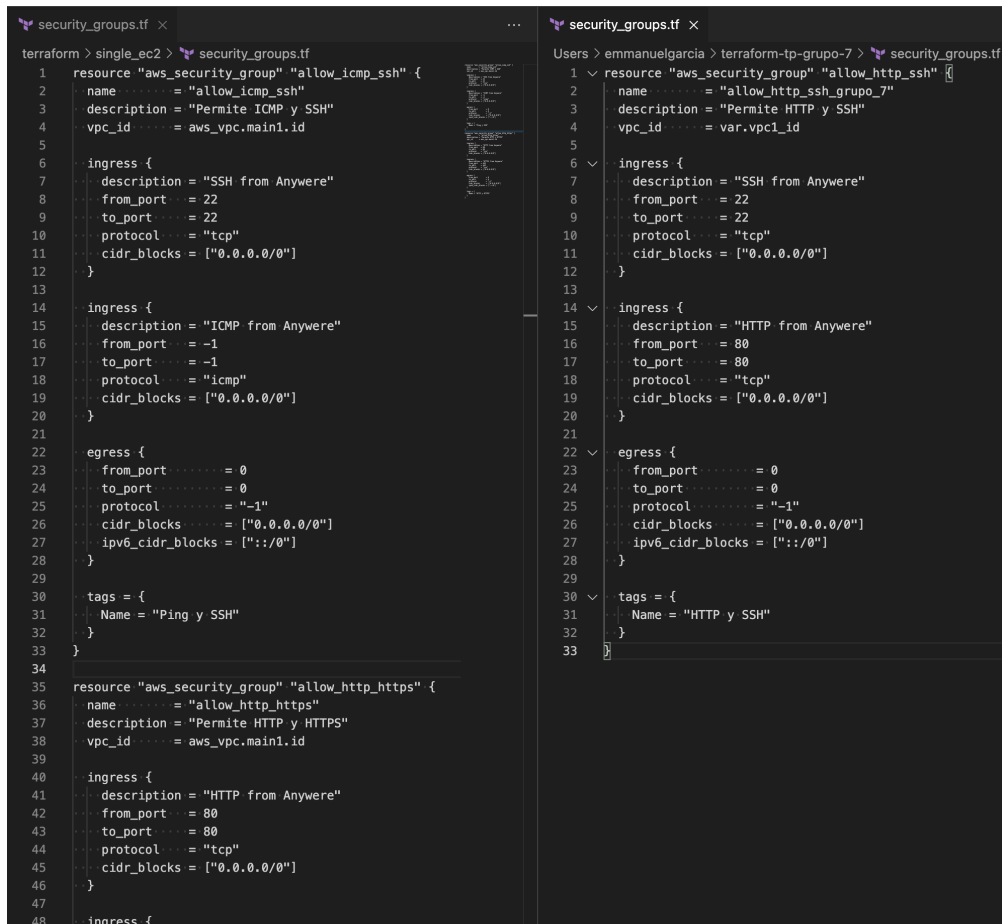
## security\_groups.tf

En este archivo se configura la seguridad que tendrá la instancia.

Al recurso "allow\_icmp\_ssh" se lo renombró por "allow\_http\_ssh", ya que se pide acceso mediante HTTP y SSH, y no ICMP. Lo mismo se reflejó en el nombre y la descripción. Al ID de la VPC se le asignó la variable `var.vpc1_id`.

El ingress con descripción "SSH from Anywere" se dejó sin cambios. El de descripción "ICMP from Anywere" se reemplazó por otro "HTTP from Anywere", que abre el puerto 80 y usa protocolo TCP. El egress se dejó sin cambios y el tag `Name` se cambió a "HTTP y SSH".

El recurso "allow\_http\_https" se eliminó, ya que se habilitó el acceso mediante HTTP en el recurso anterior, y no se solicitó mediante HTTPS.



The image shows two side-by-side Terraform configuration files for AWS security groups. The left file, named 'security\_groups.tf', defines three security groups: 'allow\_icmp\_ssh', 'allow\_icmp', and 'allow\_http\_https'. The 'allow\_icmp\_ssh' group allows SSH (port 22) and ICMP (port -1) from anywhere. The 'allow\_icmp' group allows ICMP from anywhere. The 'allow\_http\_https' group allows HTTP (port 80) from anywhere. The right file, also named 'security\_groups.tf', shows the updated configuration where the 'allow\_http\_https' group has been replaced by 'allow\_http\_ssh\_grupo\_7', which allows SSH (port 22) and HTTP (port 80) from anywhere. The 'allow\_icmp' group remains unchanged.

```
1 resource "aws_security_group" "allow_icmp_ssh" {
2   name = "allow_icmp_ssh"
3   description = "Permite ICMP y SSH"
4   vpc_id = aws_vpc.main1.id
5
6   ingress {
7     description = "SSH from Anywhere"
8     from_port = 22
9     to_port = 22
10    protocol = "tcp"
11    cidr_blocks = ["0.0.0.0/0"]
12  }
13
14  ingress {
15    description = "ICMP from Anywhere"
16    from_port = -1
17    to_port = -1
18    protocol = "icmp"
19    cidr_blocks = ["0.0.0.0/0"]
20  }
21
22  egress {
23    from_port = 0
24    to_port = 0
25    protocol = "-1"
26    cidr_blocks = ["0.0.0.0/0"]
27    ipv6_cidr_blocks = [ "::/0" ]
28  }
29
30  tags = {
31    Name = "Ping y SSH"
32  }
33 }
34
35 resource "aws_security_group" "allow_http_https" {
36   name = "allow_http_https"
37   description = "Permite HTTP y HTTPS"
38   vpc_id = aws_vpc.main1.id
39
40   ingress {
41     description = "HTTP from Anywhere"
42     from_port = 80
43     to_port = 80
44     protocol = "tcp"
45     cidr_blocks = ["0.0.0.0/0"]
46   }
47
48   ingress {
```

```
1 resource "aws_security_group" "allow_http_ssh_grupo_7" {
2   name = "allow_http_ssh_grupo_7"
3   description = "Permite HTTP y SSH"
4   vpc_id = var.vpc1_id
5
6   ingress {
7     description = "SSH from Anywhere"
8     from_port = 22
9     to_port = 22
10    protocol = "tcp"
11    cidr_blocks = ["0.0.0.0/0"]
12  }
13
14  ingress {
15    description = "HTTP from Anywhere"
16    from_port = 80
17    to_port = 80
18    protocol = "tcp"
19    cidr_blocks = ["0.0.0.0/0"]
20  }
21
22  egress {
23    from_port = 0
24    to_port = 0
25    protocol = "-1"
26    cidr_blocks = ["0.0.0.0/0"]
27    ipv6_cidr_blocks = [ "::/0" ]
28  }
29
30  tags = {
31    Name = "HTTP y SSH"
32  }
33 }
```

instance.tf

En este archivo se encuentran los recursos y la configuración que tendrá la nueva instancia de EC2.

En el atributo `vpc_security_group_ids` se reflejaron los cambios en el archivo anterior, eliminando el security group `allow_http_https`, y renombrando el restante a `allow_http_ssh`. En el atributo `private_ip` se cargó "172.16.7.6", y en `key_name` se usó la variable `var.key1`.

Se definieron los tags `Name` y `Owner` como "server-grupo-7" y "student\_7", respectivamente. Los demás atributos (`ami`, `instance_type`, `subnet_id`, etc.) se dejaron sin cambios.



```
Instance.tf x
terraform > single_ge2 > Instance.tf
1 resource "aws_instance" "server1" {
2   ami = var.ami_id
3   instance_type = var.instance_type
4   subnet_id = aws_subnet.subnet1.id
5   associate_public_ip_address = true
6   vpc_security_group_ids = [aws_security_group.allow_icmp_ssh.id,
7     aws_security_group.allow_http_https.id]
8   private_ip = var.subnet1_private_ip
9   key_name = aws_key_pair.key1.key_name
10  user_data = file("user-data.sh")
11
12  tags = {
13    Name = "server-1"
14    Owner = var.username
15    Environment = "develop"
16    OS = "amazon-linux"
17  }
18 }
```

```
Instance.tf x
Users > emmanuelgarcia > terraform-tp-grupo-7 > Instance.tf
1 resource "aws_instance" "server1" {
2   ami = var.ami_id
3   instance_type = var.instance_type
4   subnet_id = aws_subnet.subnet1.id
5   associate_public_ip_address = true
6   vpc_security_group_ids = [aws_security_group.allow_http_ssh.id]
7   private_ip = "172.16.7.6"
8   key_name = var.key1
9   user_data = file("user-data.sh")
10
11  tags = {
12    Name = "server-grupo-7"
13    Owner = "student_7"
14    Environment = "develop"
15    OS = "amazon-linux"
16  }
17 }
```

provider.tf, output.tf y user-data.sh

Consideramos que no es necesario que se deban realizar modificaciones a estos archivos.

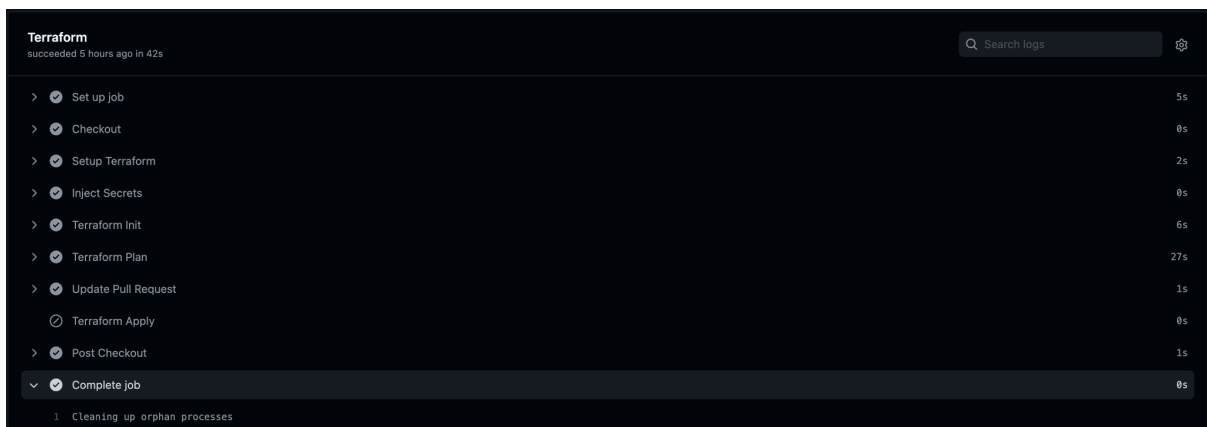
```
provider.tf x
provider.tf
1 provider "aws" {
2   region = var.region
3 }
4
5 terraform {
6   required_providers {
7     aws = {
8       source = "hashicorp/aws"
9     }
10    random = {
11      source = "hashicorp/random"
12    }
13  }
14
15  backend "remote" {
16    organization = "_organization_"
17
18    workspaces {
19      name = "_workspace_"
20    }
21  }
22 }
23 }
```

```
output.tf x
output.tf
1 output "public_ips" {
2   value = aws_instance.server1.*.public_ip
3 }
```

```
$ user-data.sh x
$ user-data.sh
1  #!/bin/bash
2
3  # Create mount volume for logs
4  sudo su - root
5  mkfs.ext4 /dev/sdf
6  mount -t ext4 /dev/sdf /var/log
7
8  # Install & Start nginx server
9  yum search nginx
10 amazon-linux-extras install nginx1 -y
11 systemctl start nginx
12 systemctl enable nginx
13
14 # Print the hostname which includes instance details on nginx homepage
15 sudo mkdir /usr/share/nginx/html/img
16 sudo wget -P /usr/share/nginx/html/img https://github.com/facundoalcaron/computernetworks/raw/main/webfiles/img/This-is-the-way-Mandalorian.jpeg
17 export HOSTNAME=$(hostname -f)
18 sudo echo '<!DOCTYPE html><html>\
19 <body></body>\
20 <script type="text/javascript">console.log("Hello from '$HOSTNAME'");</script></html>\
21 > /usr/share/nginx/html/index.html
22
```

## Ejecución del workflow

El job de Terraform a través del GitHub Actions se ejecuta cada vez que se realiza un push a una rama que contenga un archivo `terraform.yml`.



```
Terraform
succeeded 5 hours ago in 42s

> Set up job 5s
> Checkout 0s
> Setup Terraform 2s
> Inject Secrets 0s
> Terraform Init 6s
> Terraform Plan 27s
> Update Pull Request 1s
> Terraform Apply 0s
> Post Checkout 1s
> Complete job 0s
1 Cleaning up orphan processes
```

Observamos como los servicios de AWS nos brinda una IP a través de las declaraciones en Terraform

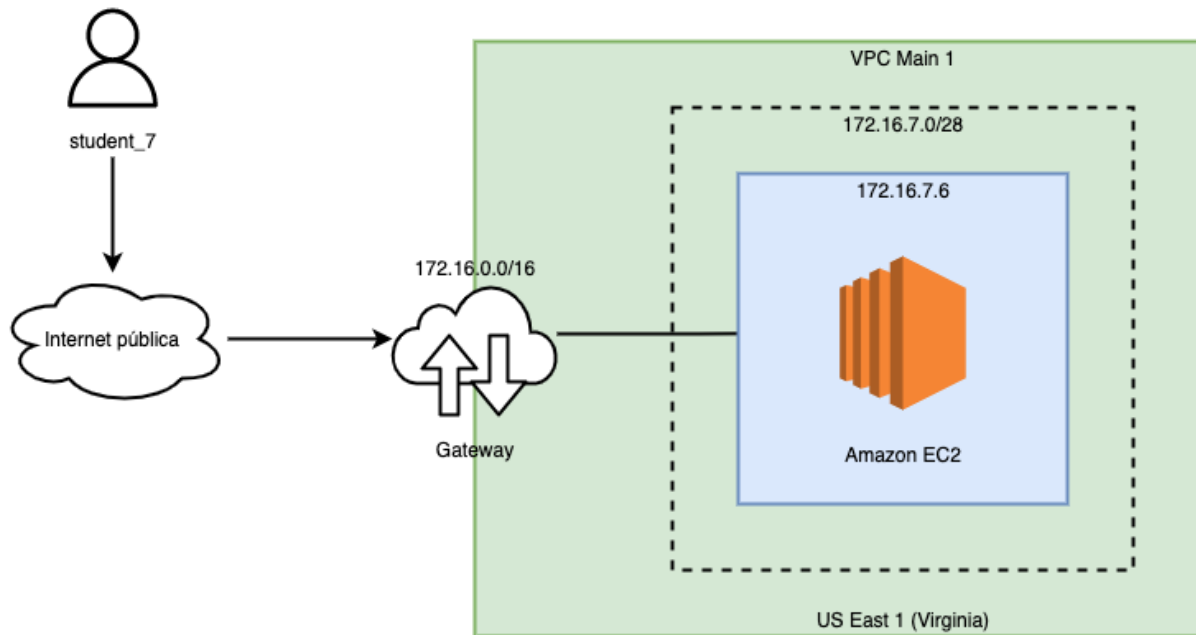
```

53 Terraform used the selected providers to generate the following execution
54 plan. Resource actions are indicated with the following symbols:
55   + create
56   - destroy
57
58 Terraform will perform the following actions:
59
60   # aws_instance.server1 will be created
61   + resource "aws_instance" "server1" {
62     + ami                               = "ami-02e136e904f3da870"
63     + arn                               = (known after apply)
64     + associate_public_ip_address      = true
65     + availability_zone                = (known after apply)
66     + cpu_core_count                   = (known after apply)
67     + cpu_threads_per_core             = (known after apply)
68     + disable_api_termination          = (known after apply)
69     + ebs_optimized                    = (known after apply)
70     + get_password_data                = false
71     + host_id                          = (known after apply)
72     + id                               = (known after apply)
73     + instance_initiated_shutdown_behavior = (known after apply)
74     + instance_state                   = (known after apply)
75     + instance_type                    = "t2.micro"
76     + ipv6_address_count                = (known after apply)
77     + ipv6_addresses                   = (known after apply)
78     + key_name                          = "key1"
79     + monitoring                       = (known after apply)
80     + outpost_arn                      = (known after apply)
81     + password_data                    = (known after apply)
82     + placement_group                  = (known after apply)
83     + placement_partition_number        = (known after apply)
84     + primary_network_interface_id      = (known after apply)
85     + private_dns                      = (known after apply)
86     + private_ip                       = "172.16.7.6"
87     + public_dns                       = (known after apply)
88     + public_ip                        = (known after apply)

```

## Diagrama de la infraestructura

La infraestructura del trabajo es la siguiente:



## Conclusión

Gracias a la realización de este trabajo pudimos comprender los conceptos básicos de cómo es el uso del código declarativo de Terraform para poder desplegar una IaC usando AWS. Además, aprendimos que GitHub Actions es una herramienta potente que puede ser utilizada para automatizar despliegues, no solo en el área de Redes, sino también en otros campos.