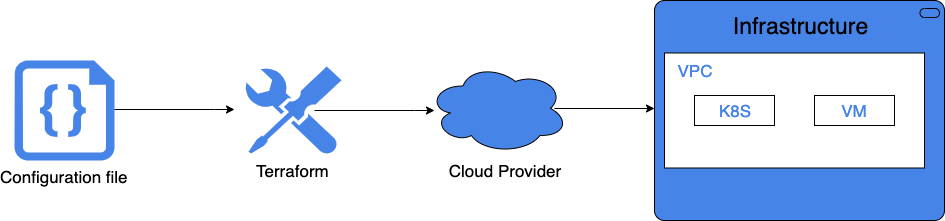
ACTIVIDAD Terraform Como IaC

* Máquina Linux / macOS
* Cuenta de AWS
* Comprensión básica de lo que es un Grupo De Seguridad, Región, Instancia EC2 y AMI en AWS

### 2. Que es Terraform

Terraform es una herramienta open source, desarrollada por HashiCorp que le brinda el poder de automatizar y administrar su infraestructura, y también su plataforma y servicios, utilizando un lenguaje declarativo. Escribe planes y crea [Infraestructura como código.](https://www.adictosaltrabajo.com/2020/06/19/primeros-pasos-con-terraform-crear-instancia-ec2-en-aws/" \l "03)



Cuando trabajamos en Terraform, vamos a utilizar comandos específicos que son como «etapas». Son:

* **init** – este comando instalará todos los binarios del proveedor y preparará el directorio de trabajo para trabajar con él.
* **plan** – lee el código de configuración que has escrito, compara el estado deseado de la infraestructura con lo que realmente existe. Luego crea un plan de qué hacer para que pueda alcanzar su estado deseado y muestra lo que se cambiará.
* **apply** – este comando ejecuta terraform plan al principio y luego procede a crear su infraestructura
* **destroy**– se usa para destruir la infraestructura administrada por Terraform.

### 3. Infrastructure as code

Entonces, la infraestructura como código es básicamente poder usar un cliente o una herramienta para escribir y ejecutar su infraestructura utilizando comandos o lenguaje declarativo en lugar de hacerlo manualmente. Cuando hablamos de automatizar el «proceso de rotación» de nuestra infraestructura, tenemos dos enfoques diferentes que podemos usar.

* **Enfoque imperativo**– este enfoque está relacionado con los comandos de CLI de cableado de un desarrollador para que la infraestructura esté en funcionamiento en algún proveedor. Por ejemplo, si queremos tener una infraestructura ejecutándose con Kubernetes y una máquina virtual en una VPC, decimos:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | client create Kuberenetes      ...more commands  client create Virtual Machine      ...more commands  client wrap Kubernetes and Virtual Machine into a VPC      ...more commands |

* **Enfoque declarativo** – lo que este enfoque le permite hacer es definir el estado final requerido de la infraestructura sin ejecutar un solo comando y luego dejar que el proveedor se encargue del resto.

**Shell**

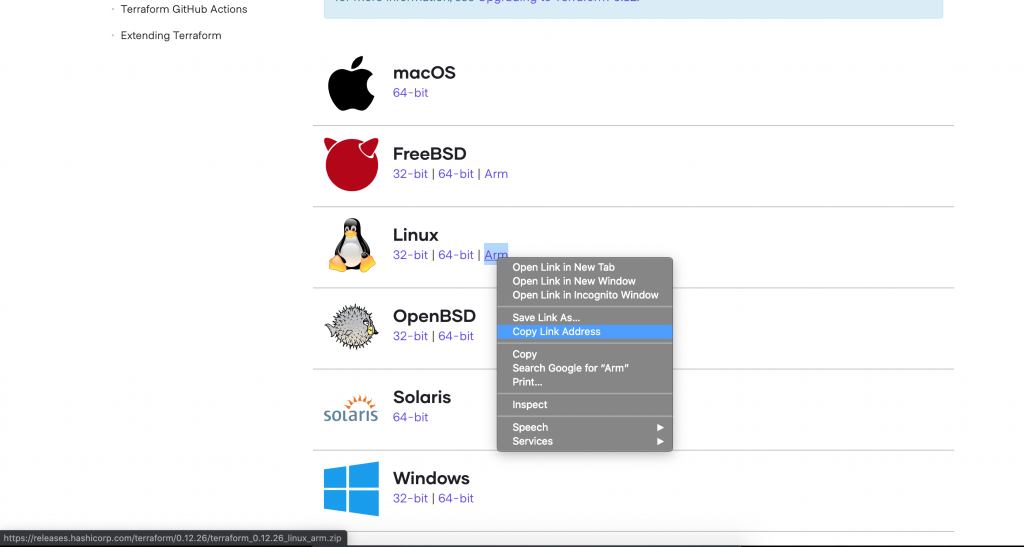
|  |  |
| --- | --- |
| 1  2  3  4  5  6 | I want to have Kubernetes resource running:      - additional config  I want a VM resource too:      - additional config  I want them to be wrapped into a VPC resource:      - additional config |

En conclusion. Si ejecutamos el script de enfoque imperativo, terminaremos con múltiples entornos. Si uno de los pasos falla, deberá proporcionar una forma adicional de manejar el error, revertir algunos cambios necesarios o intentar recrear la instancia.

Con el enfoque declarativo, no importa cuántas veces ejecute el script, siempre termina con el mismo resultado que ha declarado porque la herramienta (en nuestro caso Terraform) se encarga de todo por nosotros. Entonces tenemos esa independencia cuando trabajamos con un enfoque declarativo.

### 4. Cómo instalar Terraform

1. Ir a la [página de descarga oficial](https://www.terraform.io/downloads.html) y copiar el enlace de la instalación para tu sistema operativo específico.

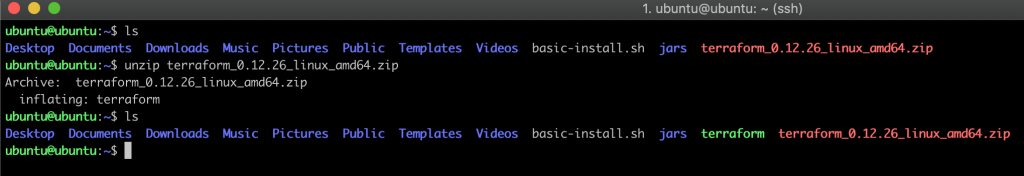
  
2. Ir a la terminal y lanza el comando siguiente para bajar el zip. Reemplazar el enlace con el enlace que acaba de copiar del sitio web para su sistema operativo

|  |  |
| --- | --- |
| 1 | wget https://releases.hashicorp.com/terraform/0.12.26/terraform\_0.12.26\_linux\_arm.zip |

3. Descomprimir el archivo. Si no tienes unzip, puedes instalarlo con **apt-get install unzip**

|  |  |
| --- | --- |
| 1 | unzip terraform\_0.12.26\_linux\_arm.zip |

Verás que se extrajo un archivo binario llamado «terraform».



4. Si intentas ejecutar el comando **$ terraform** en el terminal, dirá **«comando no encontrado»**. Dado que es un archivo binario y no será cómodo ejecutarlo cada vez, establezcamos una ruta hacia él. Ejecuta los comandos:

|  |  |
| --- | --- |
| 1 | echo $"export PATH=\$PATH:$(pwd)" >> ~/.bash\_profile |

|  |  |
| --- | --- |
| 1 | source ~/.bash\_profile |

La parte **\$PATH:$(pwd)** del primer comando es simplemente la ruta actual en la que estamos. Así que asegúrate de estar en el directorio donde descomprimiste Terraform o sustituye esa parte con **"/path/to/terraform/binary/file"**.

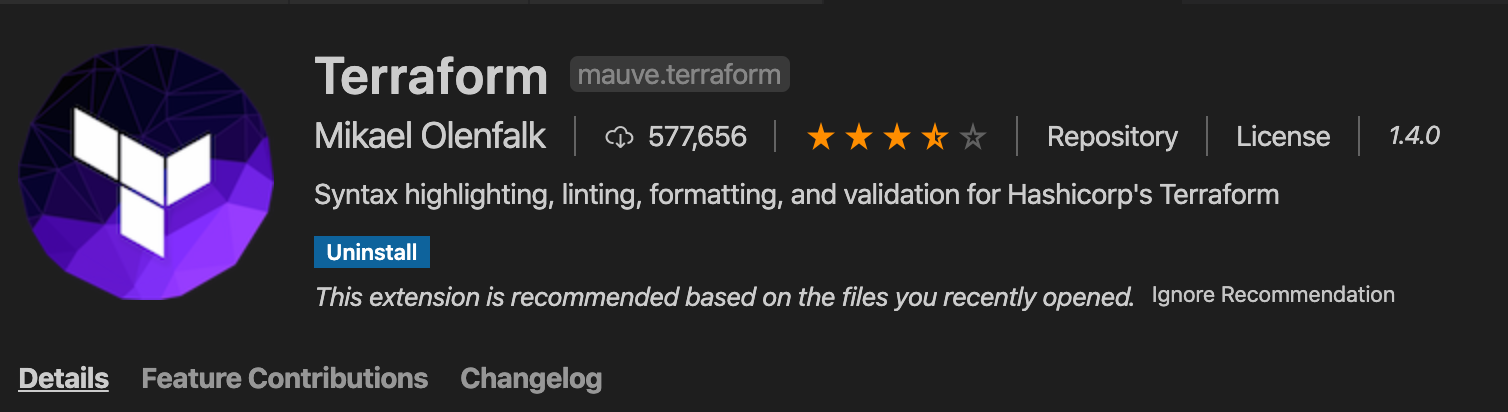
Ahora, cuando ejecutes **terraform** en el terminal, deberías ver el siguiente resultado que indica que lo hemos instalado con éxito y está listo para usar.



Te pedí intencionalmente que ejecutaras el comando **$ terraform** en lugar de **$ terraform -version** para que pidieras echar un vistazo y ver la lista de comandos, ¡siempre es bueno hacerlo!

### 5. Crear una EC2 instancia simple

Si usas VS Code para un editor de texto, ten en cuenta que hay un plugin de Terraform que ayuda a escribir la configuración. Se parece a esto:



Vamos a crear una instancia EC2 simple usando Terraform y AWS como proveedor de la nube.

Antes de eso, navega a tu carpeta de practica de Terraform y crea un archivo con la extensión «.tf». Este será tu archivo de configuración principal. Lo he llamado «main.tf».

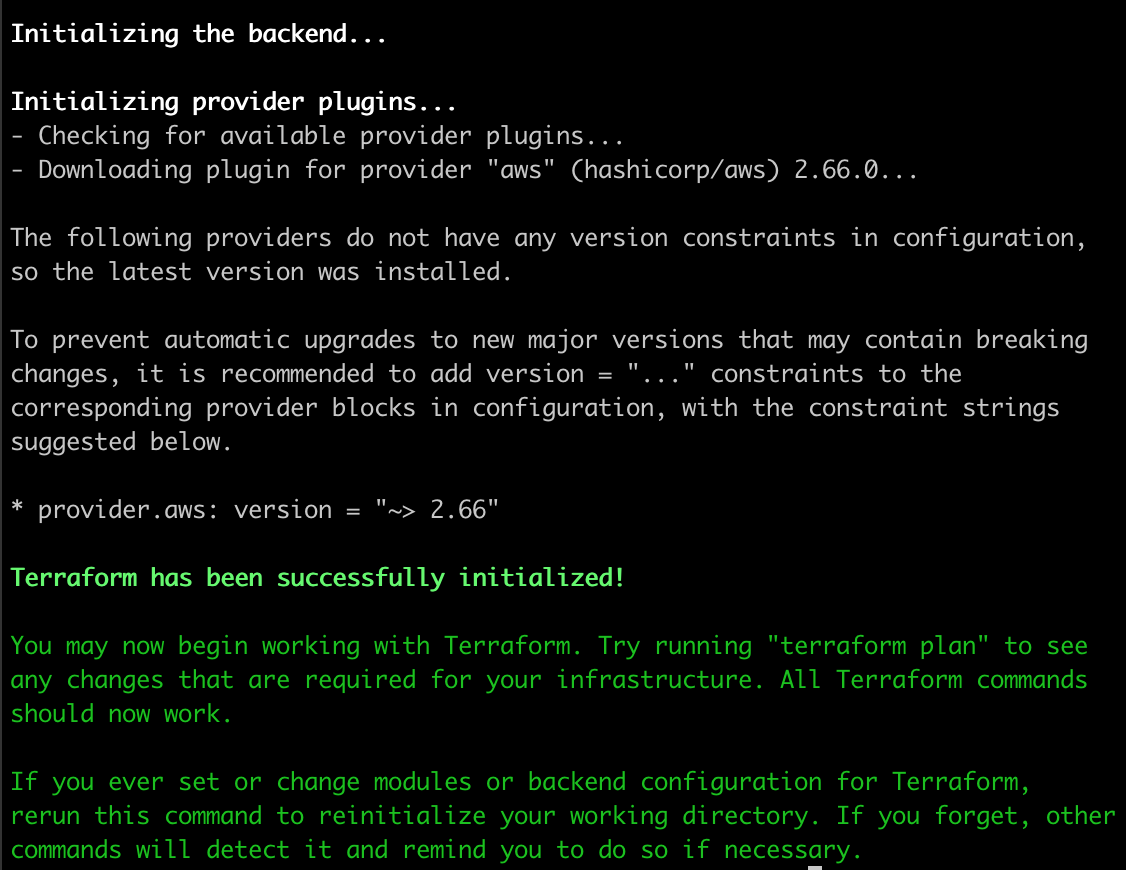
Entonces, el primer paso es preparar el directorio de trabajo para usarlo con AWS. Para hacerlo, en nuestro archivo main.tf, escribe la siguiente configuración de proveedor:

|  |  |
| --- | --- |
| 1  2  3  4  5 | provider "aws" {  region = "eu-west-3"      access\_key = "aws\_user\_access\_key"  secret\_key = "aws\_user\_secret\_key"  } |

En los campos de clave de acceso y clave secreta, debemos proporcionar las credenciales para nuestro usuario específico de AWS.

**No se recomienda usar la cuenta raíz para las tareas diarias**, por lo que es mejor proporcionar credenciales para el usuario de IAM que tiene los permisos de administrador. Para obtener las credenciales de usuario, ve a la [página del documentación oficial](https://aws.amazon.com/es/blogs/security/wheres-my-secret-access-key/) y sigue los pasos.

Guarda el fichero y ejecuta el comando **terraform init** desde el terminal. Este comando instalará todos los binarios del proveedor y preparará el directorio de trabajo para trabajar con él. Deberías ver el siguiente resultado que indica que la inicialización fue exitosa:



Dado que el archivo principal de terraform se va a cargar en el repositorio de código con todo el proyecto, claramente no podemos almacenar ningún dato sensible. Así que pongamos la clave secreta y la clave de acceso en un archivo diferente, y coloquemos este archivo en nuestro gitignore.

Para separar los datos sensibles, crea un archivo en el mismo directorio llamado **terraform.tfvars**. Y decláralos así:

|  |  |
| --- | --- |
| 1  2 | my\_access\_key = "RANDOM\_TEST\_ACCESS\_KEY"  my\_secret\_key = "RANDOM\_TEST\_SECRET\_KEY" |

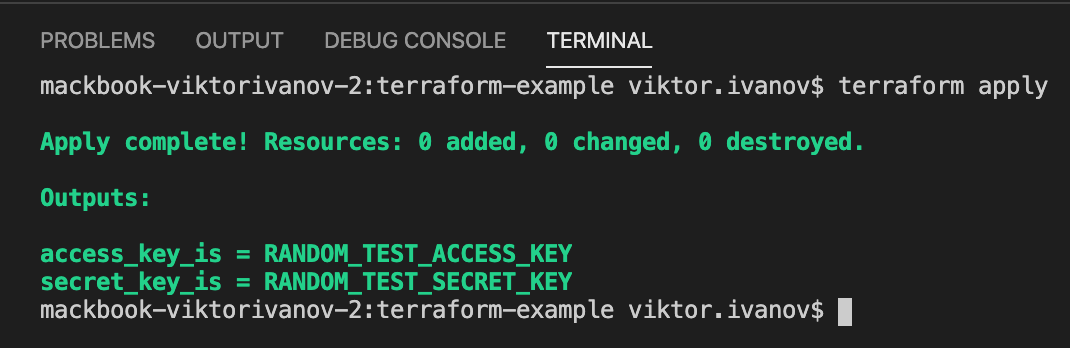
Los archivos de valores variables con nombres que no coinciden con **terraform.tfvars** o **\*.auto.tfvars** se pueden especificar con la opción **-var-file**

Para usar las variables en el archivo Terraform principal, necesitamos declararlas con **el mismo nombre** que les dimos en el archivo «terraform.tfvars» y luego acceder a cada una así  **var.varName**. Para probar que está funcionando, incluyamos también la salida de estas variables para que podamos verlas en la consola.

Su archivo principal de Terraform debería verse asi

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | variable "my\_access\_key" {    description = "Access-key-for-AWS"    default = "no\_access\_key\_value\_found"  }    variable "my\_secret\_key" {    description = "Secret-key-for-AWS"    default = "no\_secret\_key\_value\_found"  }    output "access\_key\_is" {    value = var.my\_access\_key  }    output "secret\_key\_is" {    value = var.my\_secret\_key  }    provider "aws" {  region = "eu-west-3"  access\_key = var.my\_access\_key  secret\_key = var.my\_secret\_key  } |

Ejecuta **terraform apply** y deberías recibir un mensaje de éxito que muestra las variables de salida con sus valores:



Ahora reemplaza los valores de prueba de la clave secreta y de acceso con los reales de AWS y elimina los output variables del archivo principal.

El último paso para iniciar la instancia ec2 es agregar la configuración de recursos. Estamos especificando el ID de AMI que tomamos del sitio web de AWS y el tipo de la instancia.

Shell

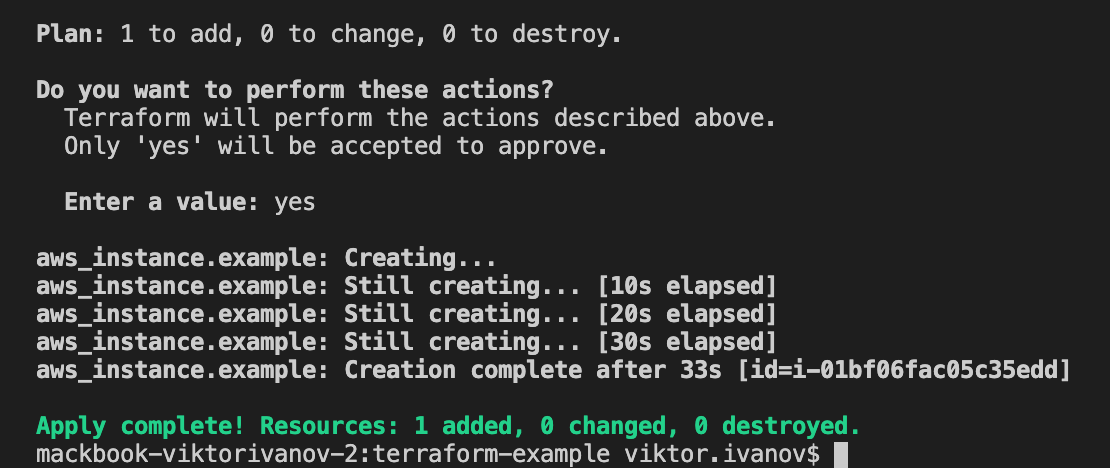
|  |  |
| --- | --- |
| 1  2  3  4 | resource "aws\_instance" "example" {  ami = "ami-0357d42faf6fa582f"  instance\_type = "t2.micro"  } |

Ejecuta **terraform plan** y verás que indica que se va a crear una instancia de EC2 y todos los cambios que implica.

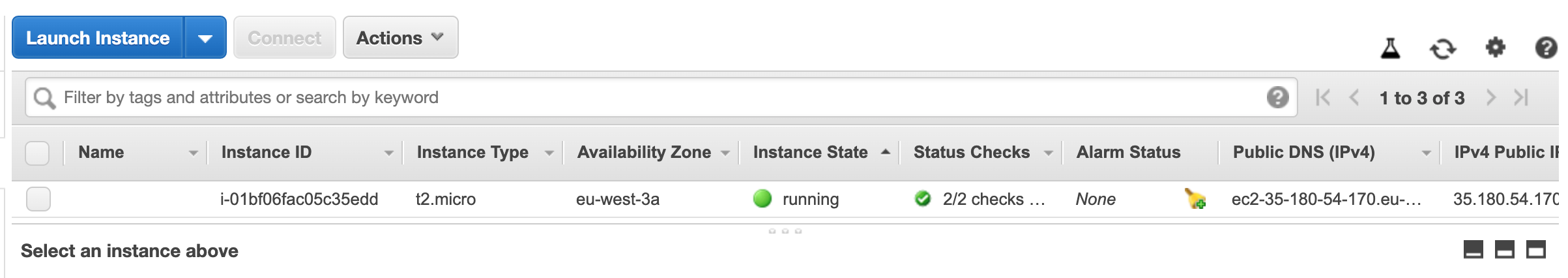
Luego ejecute **terraform apply** y verá que le da una vez más todas las cosas que van a cambiar. Entonces, ¿por qué ejecutar **terraform plan** antes?

**El plan Terraform se usa cuando desea hacer una revisión de lo que has configurado, es realmente útil durante las revisiones de código y nos ayuda a comprender mejor los cambios que hemos realizado.**

Te preguntará si deseas continuar, escribe **yes** y presiona enter. Luego debes esperar hasta que Terraform cree y lance la instancia, y reciba el siguiente resultado:



Si vamos a la consola de administración de aws y elegimos la ventana de instancia EC2, veremos nuestra instancia ec2 en funcionamiento.



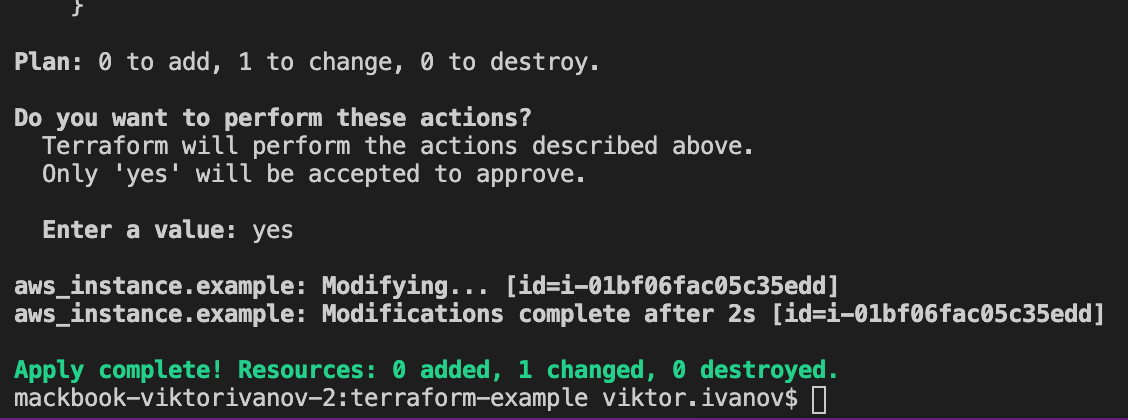
Pero observa que la instancia no tiene nombre. Vamos a cambiar nuestra configuración de Terraform para incluir una etiqueta de nombre en la instancia y ver cómo se comporta al cambiar.

Agrega una configuración de objeto de etiquetas a nuestra parte de recursos EC2 como esta y ejecuta **terraform apply** :

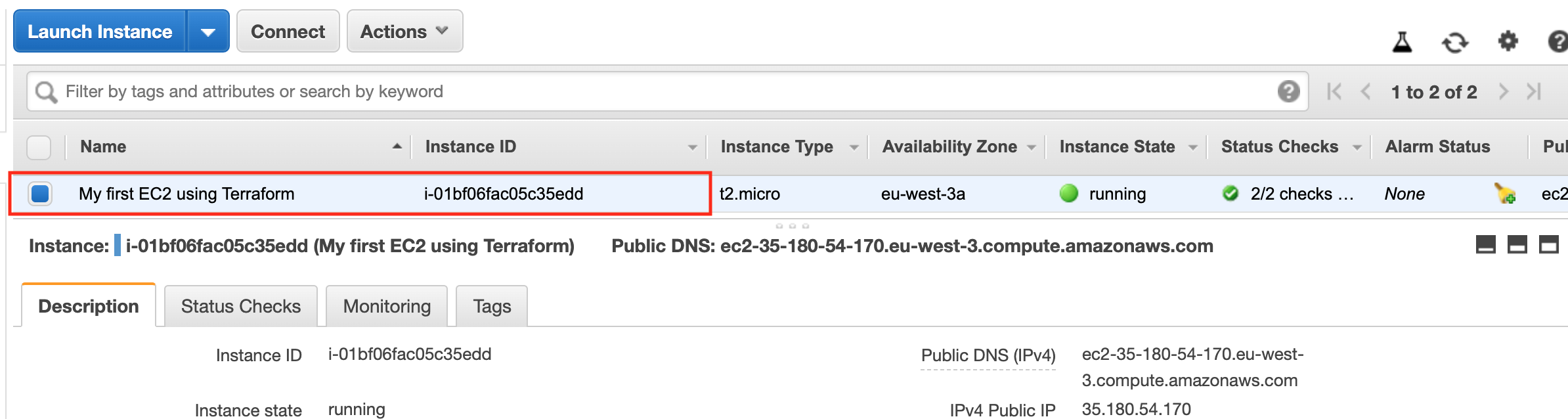
Shell

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | resource "aws\_instance" "example" {  ami = "ami-0357d42faf6fa582f"  instance\_type = "t2.micro"  tags = {  Name = "My first EC2 using Terraform"  }  } |

Terraform indica que hay un cambio detectado.



Podemos ver que tenemos la misma instancia, pero con una etiqueta de nombre agregada. Por lo tanto, Terraform fue lo suficientemente inteligente como para descubrir que no es necesario crear una nueva instancia con ese nombre, sino actualizar la existente.



### 6. Añadir User Data y Security Group

En esta sección, incluyamos un script de User Data que instalará Apache Http Server y mostrará una página web simple. Luego creamos un Security Group con rules de seguridad que abren el puerto 80 para conexiones desde fuera.

Entonces, para crear un grupo de seguridad para que podamos permitir las conexiones a través de HTTP en el puerto 80, debemos agregarlo como un recurso en nuestro archivo **main.tf**. En la parte de **«ingress»** estamos escribiendo la configuración de la regla (inbound rules). El rango de puertos será de 80 a 80, el protocolo es TCP. Y en los bloques CIDR estamos diciendo que permitimos el acceso desde cualquier IP. El bloque de **«egress»** define la regla para el tráfico saliente (outbound rules) y aquí la hemos definido para permitir todo. La combinación de **from\_port = 0**, **to\_port = 0** y el **protocol = «-1»** significa **«permitir el tráfico saliente desde todos los puertos y todos los protocolos».**

Ejecuta **terraform apply** y espera hasta que cree todo para nosotros. Luego, ingresa al panel de control EC2 y verás que tenemos nuestra instancia EC2 actualizada y el grupo de seguridad creado con sus reglas de entrada y salida.

El codigo completo final, es el siguiente :

provider "aws" {

region = "eu-west-3"

access\_key = ""

secret\_key = ""

}

resource "aws\_instance" "example" {

ami = "ami-066663db63b3aa675"

instance\_type = "t2.micro"

key\_name = "terraform-key"

security\_groups = ["${aws\_security\_group.allow\_rdp.name}"]

}

resource "aws\_security\_group" "allow\_rdp" {

name = "allow\_rdp"

description = "Allow ssh traffic"

ingress {

from\_port = 3389 # By default, the windows server listens on TCP port 3389 for RDP

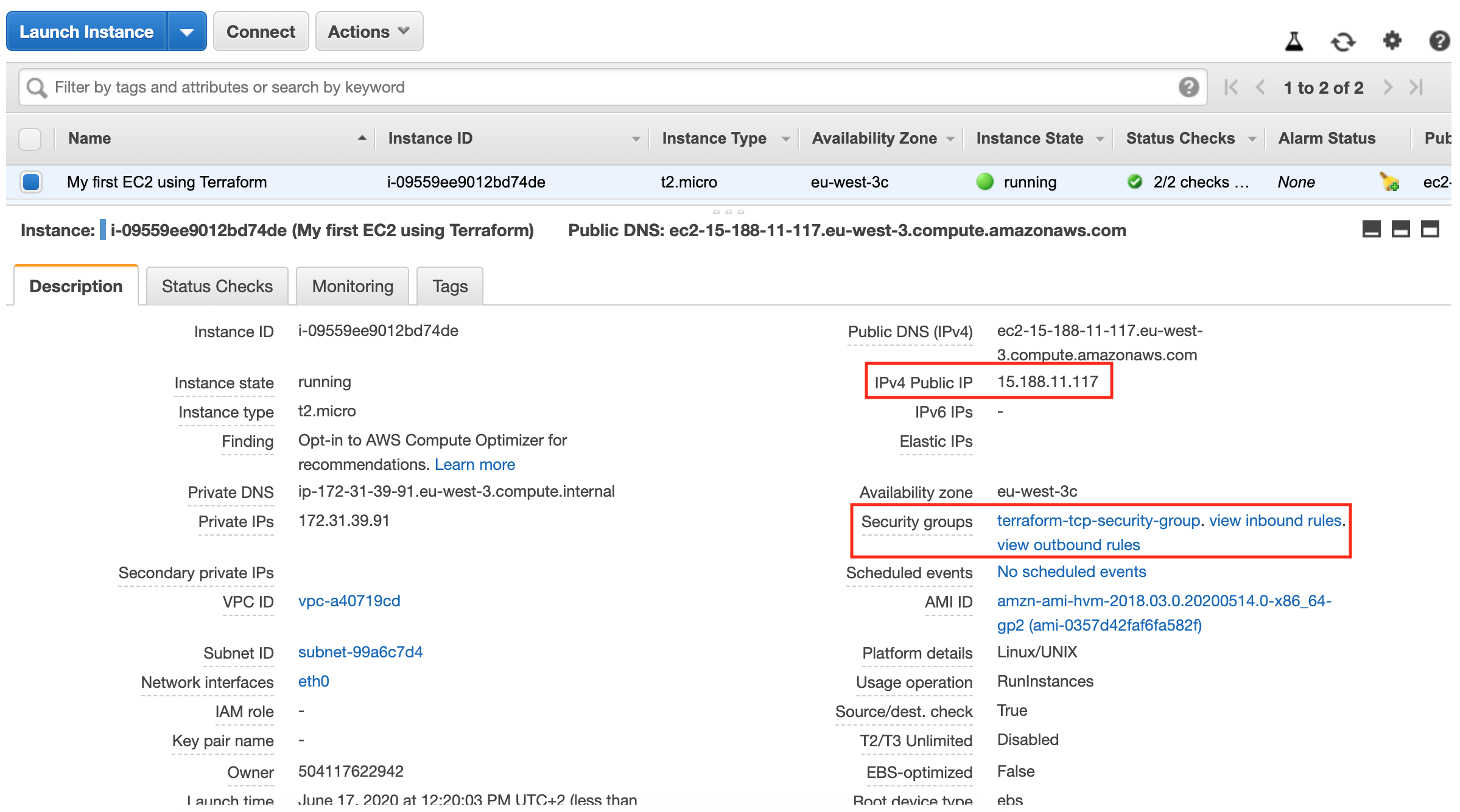
to\_port = 3389

protocol = "tcp"

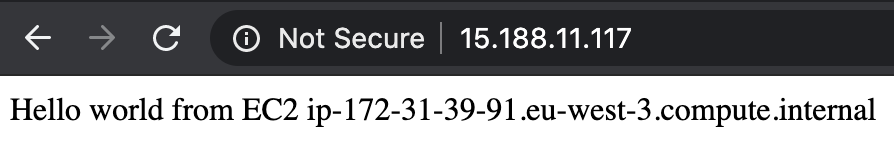
cidr\_blocks = ["0.0.0.0/0"]

}

}



Ahora intente acceder a la máquina a través de su IP pública y verá la siguiente respuesta



### 7. Conclusión

Al final, tenemos una instancia EC2 en funcionamiento que muestra una página web simple a través del acceso HTTP, creada usando Terraform. Utilizamos variables de Terraform y buenas prácticas para ocultar datos sensibles. Aprendimos a crear y vincular un grupo de seguridad simple e insertar un Script de User Data.