



Terraform

TUTORIAL BÁSICO DE TERRAFORM

RAUL CARVAJAL BUSTOS

Contenido

INTRODUCCIÓN	5
Instalación de Terraform.....	6
Instalación en sistema operativo Linux	6
Instalación en sistema operativo Windows	8
Usos básicos de Terraform.....	13
Terraform init.....	13
Terraform-providers: Azure	15
Terraform plan.....	15
Terraform graph	16
Terraform apply.	17
Terraform destroy.....	18
Creación de código.	19
AZ LOGIN.....	21
Terraform init.....	22
Crear nuestro primer grupo de recursos en Azure con teraform.....	24
Ejecutamos por primera vez Terraform plan	26
Ejecución de terraform graph.	27
Ejecución de terraform apply.	30
Usando terraform destroy para deshacer lo creado.....	31
1_NET: Mi primera red virtual.....	32
Vamos a ir rellenando los campos de la red virtual	35
Subnet.	36
Terraform graph:	38
Mis primeras variables.	42
2_NSG: Network Security Group.....	45
Security rule.....	46
3_NIC y Public ip.	50
Public IP:	51

NIC:	52
Comprobación del código:	54
Terraform destroy para finalizar el ejercicio.....	58
4_Virtual_Machine.	58
Compute.tf.....	58
variables.tf	59
Script servidor web: scripts.tf	59
Web.sh.....	60
Terraform plan.....	60
Terraform graph:	64
Resultado final	64
5_Multiples_VMs: Creación de varias VMs mediante una declaración.	66
Main.tf	68
LOOP: count y count.index.....	68
Name:	68
Compute.tf.....	70
Scripts.tf.....	71
Si realizamos un terraform graph:.....	72
Realizamos un terraform apply y aceptamos:	73
6_Metadatos	75
7_SSH: Como proteger nuestras VMs	78
Creación de una clave asimétrica mediante Powershell:.....	78
Opensshutils	78
Iniciamos el servicio de ssh:	79
Confirmamos que con la instalación se ha creado una regla para el servicio SSH en el firewall	79
Creamos una regla en el firewall de Windows:	80
Creación de nuestra clave de usuario.....	81
Permisos para no hacer accesible nuestras claves.	82
Modificación de atributos en la declaración de la máquina virtual:	84

Toca realizar el despliegue.	86
Referencias	87

Erratas:

*Se he modificado el código de la subnet para hacer un recurso aparte y no estar dentro de la declaración de la Vnet. domingo, 24 de mayo de 2020

*Se ha eliminado referencia sobre el script que indicaba que no funcionaba, miércoles, 3 de junio de 2020

*Se ha añadido capítulo 5 con la creación de varias máquinas virtuales mediante la declaración de una sola, miércoles, 3 de junio de 2020

*Se ha añadido el capítulo 6 con el acceso a los metadatos de las máquinas virtuales. Modificación del script para la creación de un index.html que refleje el hostname y la ip pública. miércoles, 3 de junio de 2020

INTRODUCCIÓN

Terraform es una herramienta para construir, cambiar y versionar infraestructura de forma segura y eficiente. Terraform puede gestionar proveedores de servicios existentes y populares, así como soluciones internas personalizadas.

Los archivos de configuración describen a Terraform los componentes necesarios para ejecutar una sola aplicación o su centro de datos completo. Terraform genera un plan de ejecución que describe lo que hará para alcanzar el estado deseado, y luego lo ejecuta para construir la infraestructura descrita. A medida que cambia la configuración, Terraform puede determinar qué cambió y crear planes de ejecución incrementales que se pueden aplicar.

La infraestructura que Terraform puede administrar incluye componentes de bajo nivel como instancias de cómputo, almacenamiento y redes, así como componentes de alto nivel como entradas DNS, funciones SaaS, etc.

Las características clave de Terraform son:

Infraestructura como Código

La infraestructura se describe utilizando una sintaxis de configuración de alto nivel. Esto permite que un plano de su centro de datos sea versionado y tratado como lo haría con cualquier otro código. Además, la infraestructura se puede compartir y reutilizar.

Planes de ejecución

Terraform tiene un paso de "planificación" donde genera un plan de ejecución. El plan de ejecución muestra lo que hará Terraform cuando llame a aplicar. Esto le permite evitar sorpresas cuando Terraform manipula la infraestructura.

Gráfico de recursos

Terraform crea un gráfico de todos sus recursos y paraleliza la creación y modificación de cualquier recurso no dependiente. Debido a esto, Terraform construye infraestructura de la manera más eficiente posible, y los operadores obtienen información sobre las dependencias en su infraestructura.

Cambiar automatización

Los conjuntos de cambios complejos se pueden aplicar a su infraestructura con una interacción humana mínima. Con el plan de ejecución y el gráfico de recursos mencionados anteriormente, usted sabe exactamente qué cambiará Terraform y en qué orden, evitando muchos posibles errores humanos.

[\(Terraform, s.f.\)](#)

Instalación de Terraform

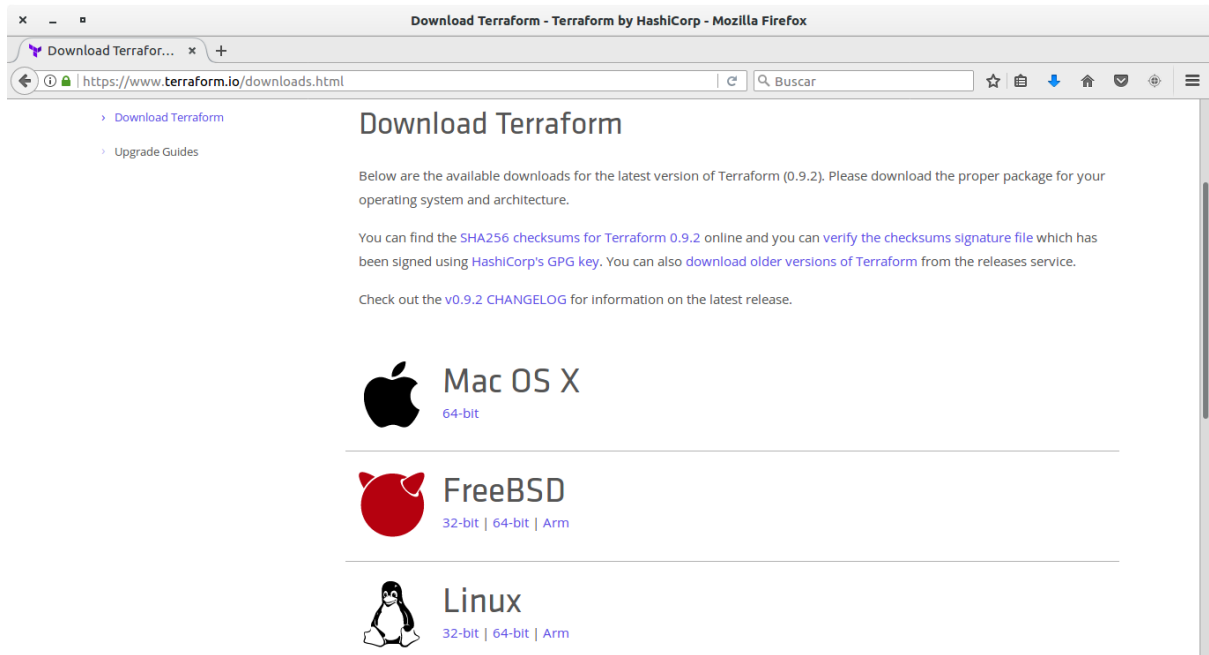
Para instalar Terraform, descargamos el paquete apropiado para nuestro sistema desde la página oficial de Terraform.

Sistemas operativos compatibles con Terraform:

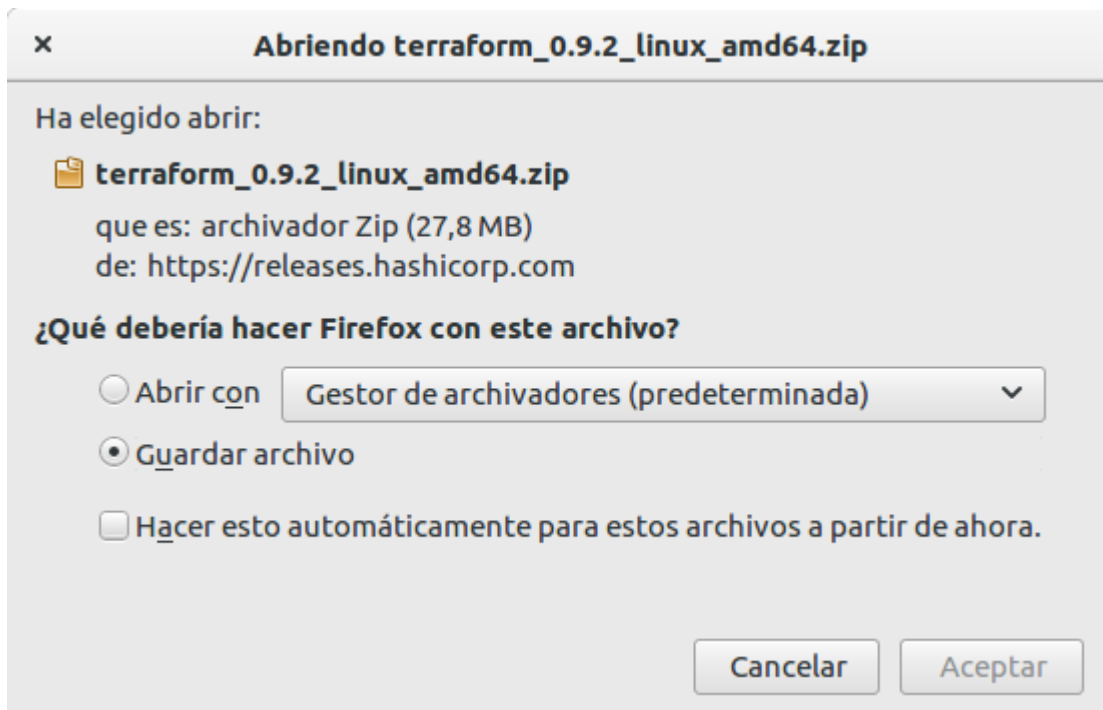
- Linux: 32bit | 64bit | Arm
- Windows: 32bit | 64bit
- Mac OS X: 64bit
- FreeBSD: 32bit | 64bit | Arm
- OpenBSD: 32bit | 64bit
- Solaris: 64bit

Instalación en sistema operativo Linux

Vamos a la página oficial <https://www.terraform.io/downloads.html>. En ella encontramos los binarios.



Seleccionamos el sistema operativo y la arquitectura, en nuestro caso elegiremos Linux 64bit puesto que lo instalaremos en una maquina Debian Jessie.



Creamos un directorio para los binarios de Terraform.

```
root@castillo:/home/jose# mkdir terraform
root@castillo:/home/jose#
```

Moveremos el fichero descargado anteriormente al interior de la carpeta.


```
root@castillo:/home/jose# mv Descargas/terraform_0.9.2_linux_amd64.zip terraform/.
root@castillo:/home/jose#
```

Nos situamos en el directorio terraform.

```
root@castillo:/home/jose# cd terraform/
root@castillo:/home/jose/terraform#
```

Descomprimos los binarios de Terraform con el comando unzip.

```
root@castillo:/home/jose/terraform# unzip terraform_0.9.2_linux_amd64.zip
Archive:  terraform_0.9.2_linux_amd64.zip
  inflating: terraform
root@castillo:/home/jose/terraform#
```

Exportamos las variables de entorno de Terraform.

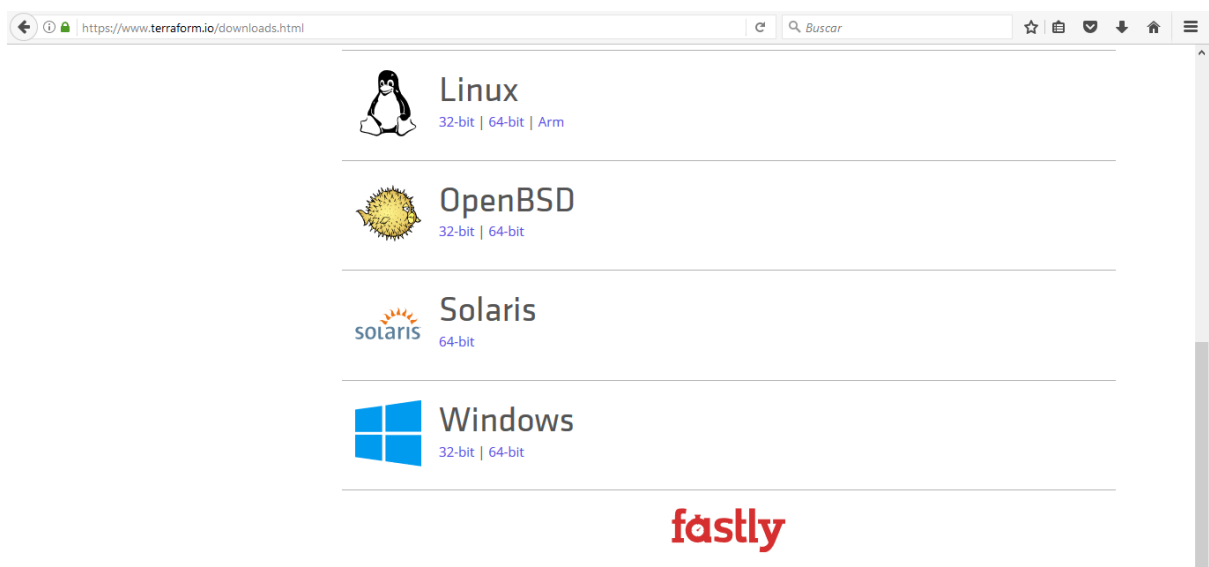
```
root@castillo:/home/jose/terraform# export PATH=/home/jose/terraform:$PATH
root@castillo:/home/jose/terraform#
```

Por último, comprobamos que se ha instalado bien ejecutando el comando siguiente:

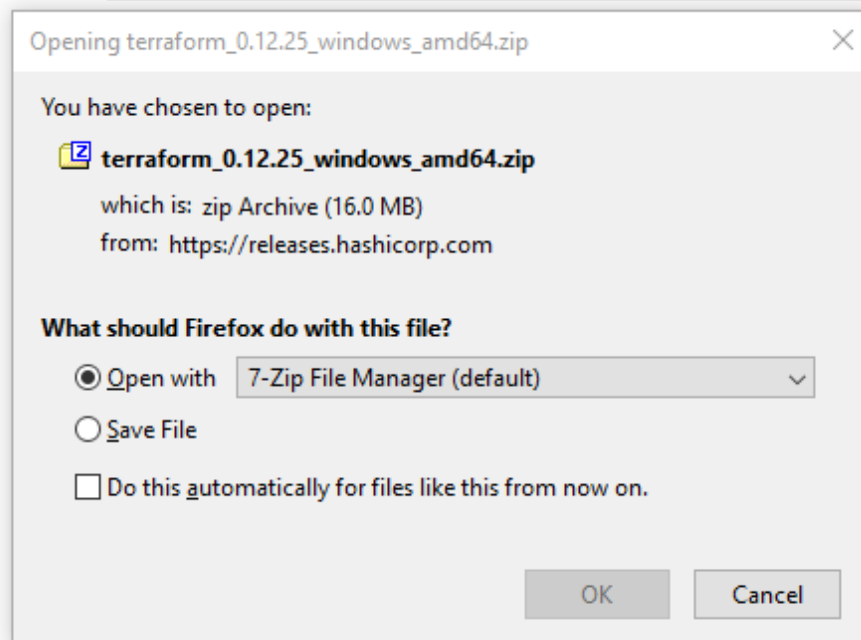
```
root@castillo:/home/jose/terraform# terraform --version
Terraform v0.9.2
root@castillo:/home/jose/terraform#
```

Instalación en sistema operativo Windows

Vamos a la página oficial <https://www.terraform.io/downloads.html>. En ella encontramos el ejecutable.



Seleccionamos el sistema operativo y la arquitectura, en nuestro caso elegiremos Windows 64bit puesto que lo instalaremos en una maquina Windows 10.



Creamos un directorio para el ejecutable de Terraform.

```
PS C:\Users\frs> mkdir terraform

Directory: C:\Users\frs

Mode                LastWriteTime         Length Name
----                -
d-----          16/05/2020    20:18             terraform

PS C:\Users\frs>
```

Copiamos el fichero descargado anteriormente al directorio terraform.

```
PS C:\Users\frs> cp D:\descargas\terraform_0.12.25_windows_amd64.zip .\terraform\
PS C:\Users\frs>
```

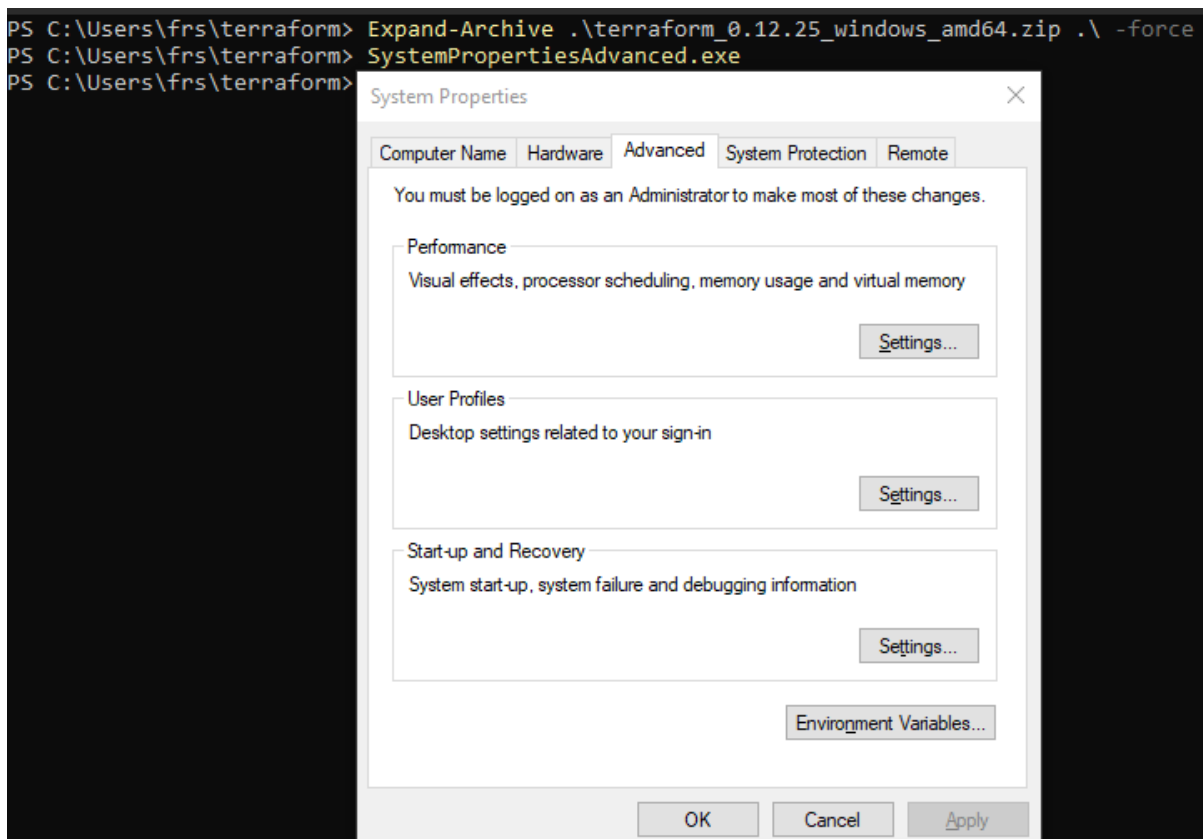
Nos situamos en el directorio terraform.

```
PS C:\Users\frs> cd .\terraform\  
PS C:\Users\frs\terraform>
```

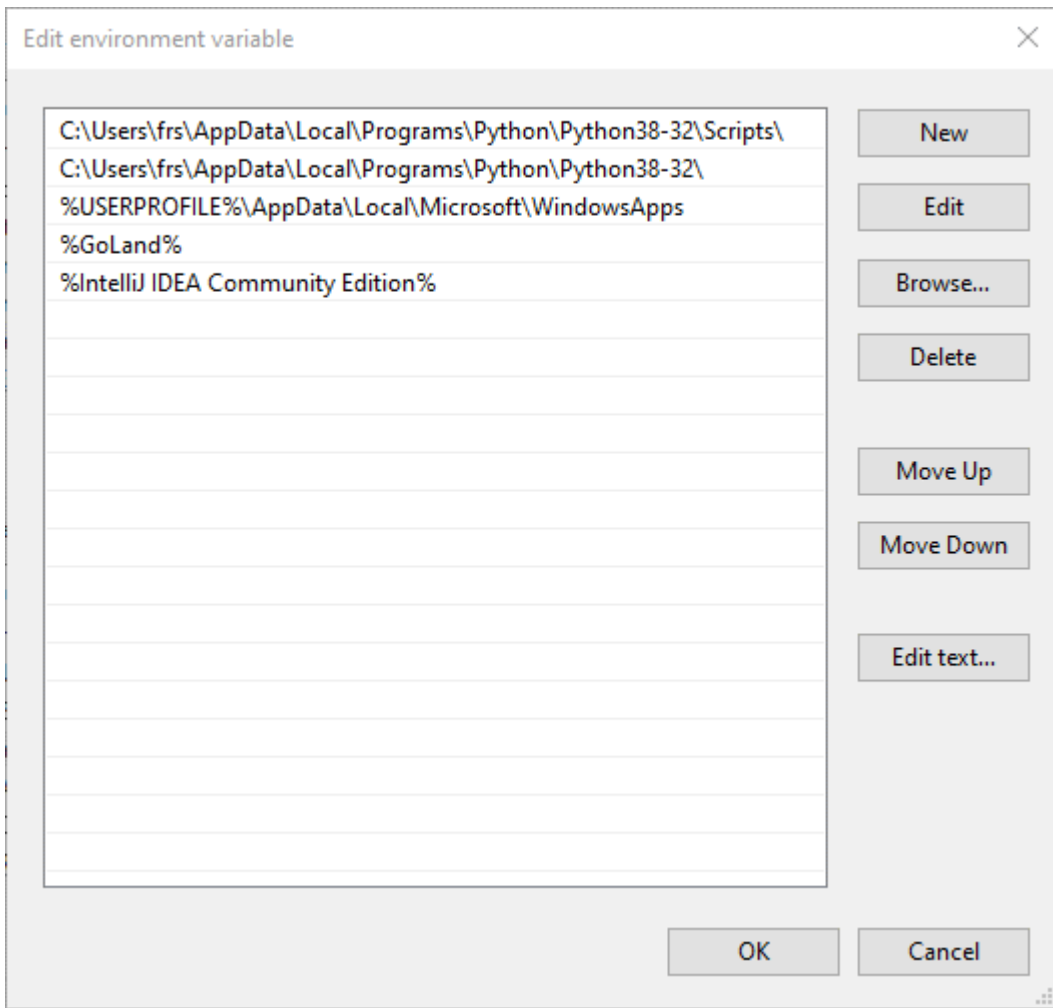
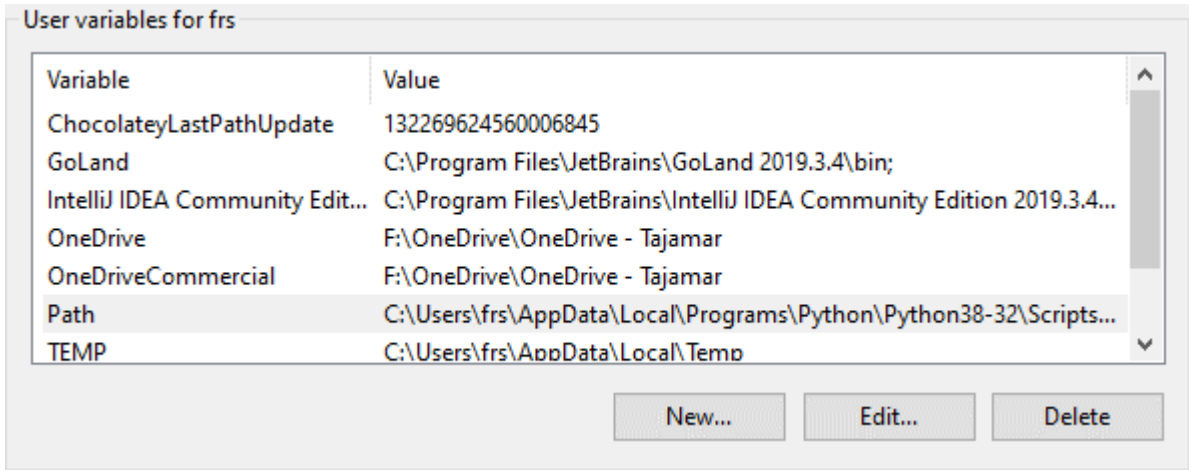
Descomprimos el ejecutable en el interior del directorio creado anteriormente.

```
PS C:\Users\frs\terraform> Expand-Archive .\terraform_0.12.25_windows_amd64.zip .\ -force  
PS C:\Users\frs\terraform>
```

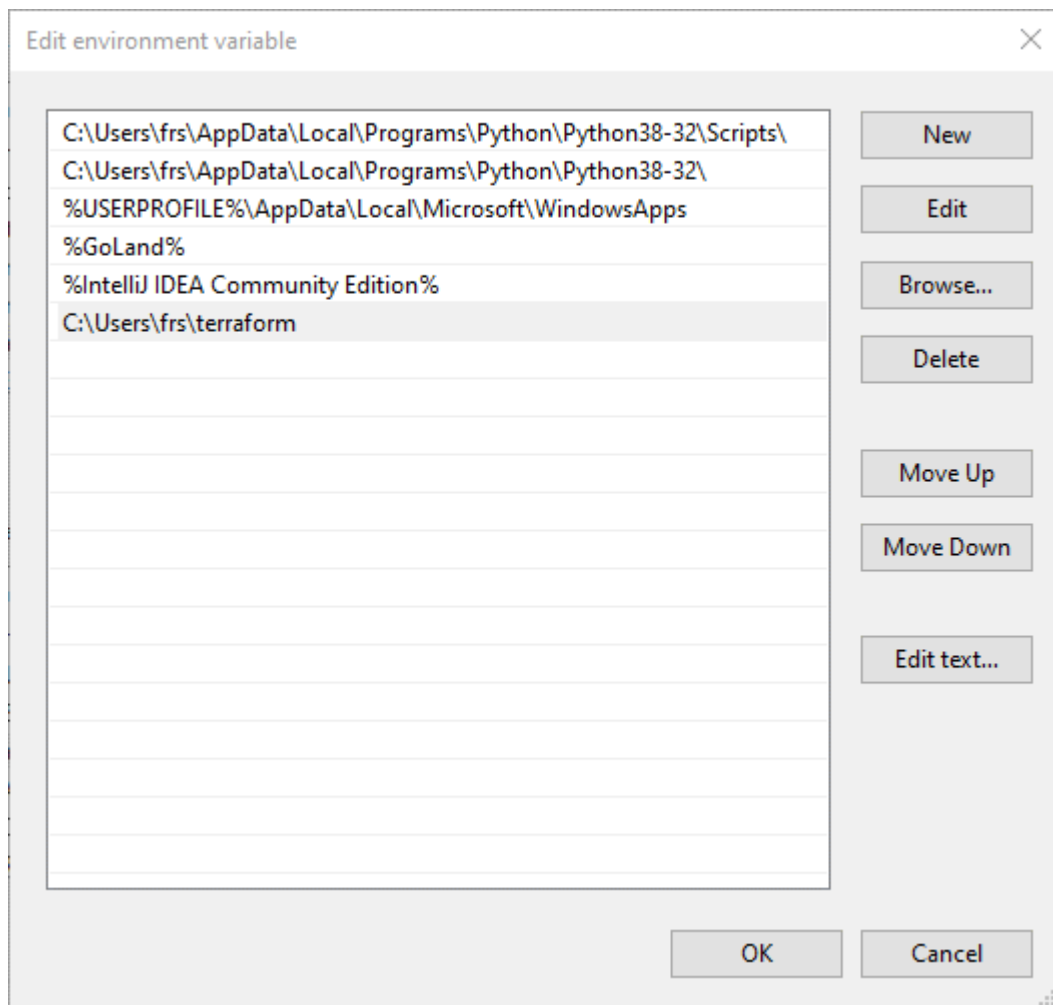
Ahora añadimos la variable de entorno de Terraform para el usuario, para ello escribimos `SystemPropertiesAdvanced.exe`. Hacemos clic en Variables de entorno.



En la siguiente pantalla editaremos la variable path en las variables del sistema.



Pinchamos en New y en una vez aparezca la celda pinchamos en Browse. Accedemos a la ruta del ejecutable:



Aceptamos y cerramos la ventana de símbolo de sistema y de las variables del sistema.

Ahora desde una nueva ventana de símbolo de sistema refrescamos comprobamos que Terraform funciona:

```
PS C:\Users\frs\terraform> terraform --version
Terraform v0.12.25
PS C:\Users\frs\terraform>
```

Usos básicos de Terraform

Ejecutando terraform nos aparecen un listado de comandos comunes.

```
PS C:\Users\frs\terraform> terraform
Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
  apply          Builds or changes infrastructure
  console        Interactive console for Terraform interpolations
  destroy        Destroy Terraform-managed infrastructure
  env            Workspace management
  fmt            Rewrites config files to canonical format
  get            Download and install modules for the configuration
  graph          Create a visual graph of Terraform resources
  import         Import existing infrastructure into Terraform
  init           Initialize a Terraform working directory
  login          Obtain and save credentials for a remote host
  logout         Remove locally-stored credentials for a remote host
  output         Read an output from a state file
  plan           Generate and show an execution plan
  providers      Prints a tree of the providers used in the configuration
  refresh        Update local state file against real resources
  show           Inspect Terraform state or plan
  taint          Manually mark a resource for recreation
  untaint        Manually unmark a resource as tainted
  validate       Validates the Terraform files
  version        Prints the Terraform version
```

Para el nivel de este tutorial, nos quedaremos con:

Init

Plan

Graph

Apply

Destroy

Terraform init

Terraform init es el primer comando que ejecutaremos en una carpeta con los archivos *.tf

Terraform -help init:

```
PS C:\Users\frs\terraform> terraform -help init
Usage: terraform init [options] [DIR]
```

Initialize a new or existing Terraform working directory by creating initial files, loading any remote state, downloading modules, etc.

This is the first command that should be run for any new or existing Terraform configuration per machine. This sets up all the local data necessary to run Terraform that is typically not committed to version control.

This command is always safe to run multiple times. Though subsequent runs may give errors, this command will never delete your configuration or state. Even so, if you have important information, please back it up prior to running this command, just in case.

If no arguments are given, the configuration in this working directory is initialized.

Options:

-backend=true	Configure the backend for this configuration.
-backend-config=path	This can be either a path to an HCL file with key/value assignments (same format as terraform.tfvars) or a 'key=value' format. This is merged with what is in the configuration file. This can be specified multiple times. The backend type must be in the configuration itself.
-force-copy	Suppress prompts about copying state data. This is equivalent to providing a "yes" to all confirmation prompts.
-from-module=SOURCE	Copy the contents of the given module into the target directory before initialization.
-get=true	Download any modules for this configuration.
-get-plugins=true	Download any missing plugins for this configuration.
-input=true	Ask for input if necessary. If false, will error if input was required.
-lock=true	Lock the state file when locking is supported.
-lock-timeout=0s	Duration to retry a state lock.

Si tuviésemos el primer archivo necesario (provider.tf con los datos de Azure) este paso descargaría los módulos de Azure necesarios para crear un grupo de recursos de Azure.

Terraform-providers: Azure

Para conocer la última versión disponible del provider, hemos de ir al repositorio de github:

<https://github.com/terraform-providers/terraform-provider-azurerm/releases>

Es recomendable visitar de vez en cuando para comprobar los cambios nuevos y modificar el provider.tf en función de los cambios.

Terraform plan.

Terraform plan genera una vista previa de las acciones que va a generar terraform y a su vez sirve para comprobar si el código es correcto.

```
PS C:\Users\frs\terraform> terraform -help plan
Usage: terraform plan [options] [DIR]

Generates an execution plan for Terraform.

This execution plan can be reviewed prior to running apply to get a
sense for what Terraform will do. Optionally, the plan can be saved to
a Terraform plan file, and apply can take this plan file to execute
this plan exactly.
```


Options:

<code>-compact-warnings</code>	If Terraform produces any warnings that are not accompanied by errors, show them in a more compact form that includes only the summary messages.
<code>-destroy</code>	If set, a plan will be generated to destroy all resources managed by the given configuration and state.
<code>-detailed-exitcode</code>	Return detailed exit codes when the command exits. This will change the meaning of exit codes to: 0 - Succeeded, diff is empty (no changes) 1 - Errored 2 - Succeeded, there is a diff
<code>-input=true</code>	Ask for input for variables if not directly set.
<code>-lock=true</code>	Lock the state file when locking is supported.
<code>-lock-timeout=0s</code>	Duration to retry a state lock.
<code>-no-color</code>	If specified, output won't contain any color.
<code>-out=path</code>	Write a plan file to the given path. This can be used as input to the "apply" command.
<code>-parallelism=n</code>	Limit the number of concurrent operations. Defaults to 10.
<code>-refresh=true</code>	Update state prior to checking for differences.

Terraform graph

Genera una representación gráfica en formato *.svg. Para visualizar el gráfico es necesario (en Windows) de un software: graphviz

<https://www.graphviz.org/>

```
PS C:\Users\frs\terraform> terraform -help graph
Usage: terraform graph [options] [DIR]

Outputs the visual execution graph of Terraform resources according to
configuration files in DIR (or the current directory if omitted).

The graph is outputted in DOT format. The typical program that can
read this format is GraphViz, but many web services are also available
to read this format.

The -type flag can be used to control the type of graph shown. Terraform
creates different graphs for different operations. See the options below
for the list of types supported. The default type is "plan" if a
configuration is given, and "apply" if a plan file is passed as an
argument.

Options:

  -draw-cycles      Highlight any cycles in the graph with colored edges.
                    This helps when diagnosing cycle errors.

  -type=plan        Type of graph to output. Can be: plan, plan-destroy, apply,
                    validate, input, refresh.

  -module-depth=n   (deprecated) In prior versions of Terraform, specified the
                    depth of modules to show in the output.

PS C:\Users\frs\terraform>
```

Terraform apply.

El comando aplicará los cambios necesarios y ejecutará las acciones. Podemos realizar ejecuciones específicas o completas. Como no quiero arbumar, usaremos el comando apply solo para que se haga todo la configuración.

```
PS C:\Users\frs\terraform> terraform -help apply
Usage: terraform apply [options] [DIR-OR-PLAN]

Builds or changes infrastructure according to Terraform configuration
files in DIR.

By default, apply scans the current directory for the configuration
and applies the changes appropriately. However, a path to another
configuration or an execution plan can be provided. Execution plans can be
used to only execute a pre-determined set of actions.
```

Options:

<code>-auto-approve</code>	Skip interactive approval of plan before applying.
<code>-backup=path</code>	Path to backup the existing state file before modifying. Defaults to the "-state-out" path with ".backup" extension. Set to "-" to disable backup.
<code>-compact-warnings</code>	If Terraform produces any warnings that are not accompanied by errors, show them in a more compact form that includes only the summary messages.
<code>-lock=true</code>	Lock the state file when locking is supported.
<code>-lock-timeout=0s</code>	Duration to retry a state lock.
<code>-input=true</code>	Ask for input for variables if not directly set.
<code>-no-color</code>	If specified, output won't contain any color.
<code>-parallelism=n</code>	Limit the number of parallel resource operations. Defaults to 10.
<code>-refresh=true</code>	Update state prior to checking for differences. This has no effect if a plan file is given to apply.
<code>-state=path</code>	Path to read and save state (unless state-out is specified). Defaults to "terraform.tfstate".

Terraform destroy.

Con este comando ejecutaremos la destrucción de todos los recursos que hayamos realizado con nuestro archivo de configuración (no borra todo lo que tenemos en la nube).

```
PS C:\Users\frs\terraform> terraform -help destroy
Usage: terraform destroy [options] [DIR]

Destroy Terraform-managed infrastructure.
```

Options:

<code>-backup=path</code>	Path to backup the existing state file before modifying. Defaults to the "-state-out" path with ".backup" extension. Set to "-" to disable backup.
<code>-auto-approve</code>	Skip interactive approval before destroying.
<code>-force</code>	Deprecated: same as auto-approve.
<code>-lock=true</code>	Lock the state file when locking is supported.
<code>-lock-timeout=0s</code>	Duration to retry a state lock.
<code>-no-color</code>	If specified, output won't contain any color.
<code>-parallelism=n</code>	Limit the number of concurrent operations. Defaults to 10.
<code>-refresh=true</code>	Update state prior to checking for differences. This has no effect if a plan file is given to apply.
<code>-state=path</code>	Path to read and save state (unless state-out is specified). Defaults to "terraform.tfstate".
<code>-state-out=path</code>	Path to write state to that is different than "-state". This can be used to preserve the old state.

Creación de código.

Empezamos en nuestra carpeta terraform y creamos un directorio nuevo que se llamará 0_RG.

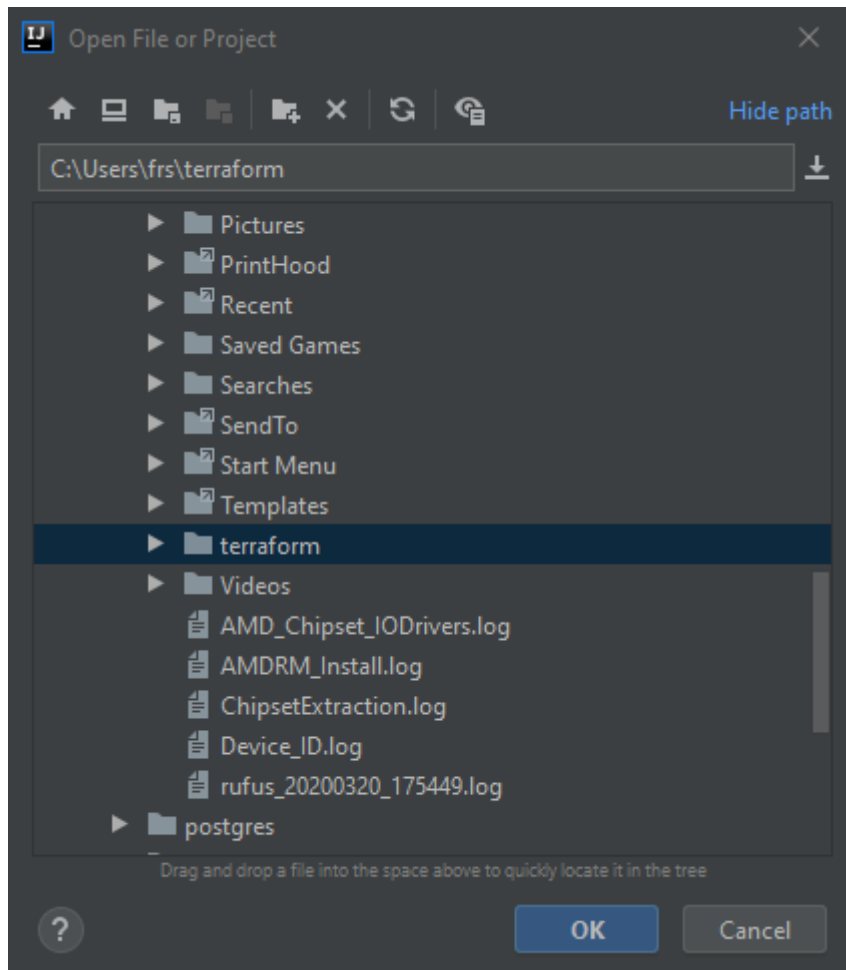
```
PS C:\Users\frs\terraform> mkdir 0_RG

Directory: C:\Users\frs\terraform

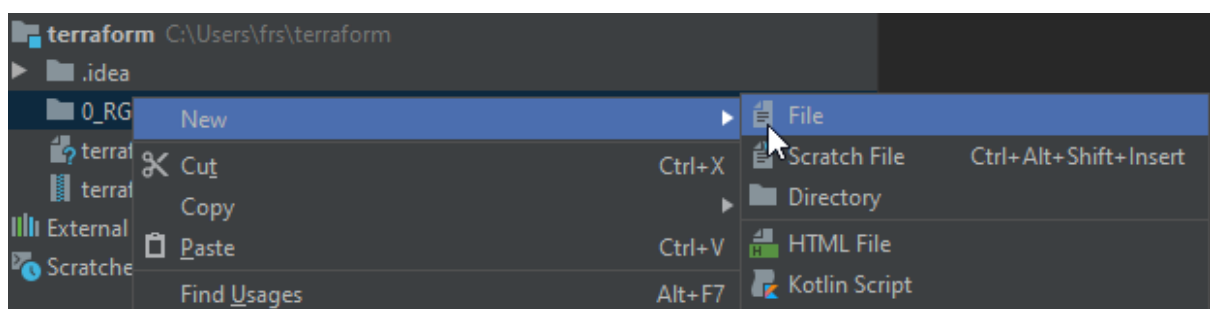
Mode                LastWriteTime         Length Name
----                -
d-----          16/05/2020   23:40             0_RG

PS C:\Users\frs\terraform> cd .\0_RG\
PS C:\Users\frs\terraform\0_RG> ls
PS C:\Users\frs\terraform\0_RG>
```

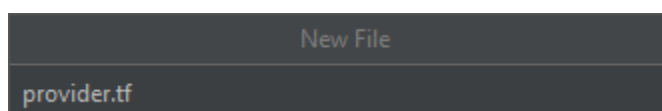
Si tenemos VS Code, abriremos la carpeta o con el editor que tengamos, en mi caso utilizaré JetBrains IntelliJ IDEA Community Edition:



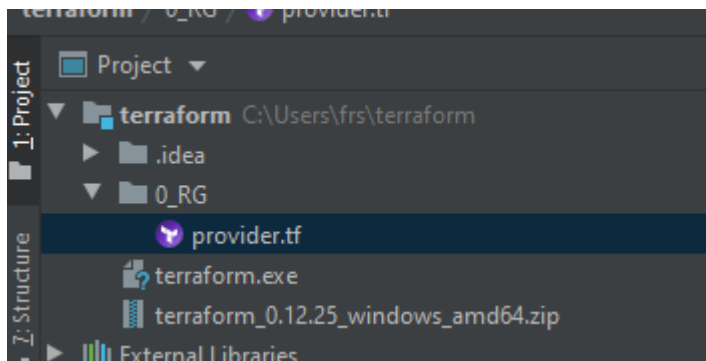
Doble click en 0_RG y con el botón derecho pinchamos en new file:



En el cuadro escribimos provider.tf



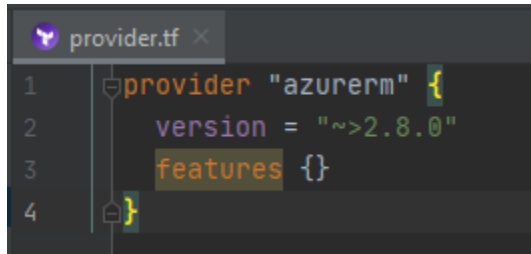
Pulsamos enter y ya tenemos nuestro archivo provider para el tipo de nube que usemos (aws, azure, Cloudflare, etc...):



Si estás usando IntelliJ IDEA al detectar el archivo .tf te pedirá instalar el plugin de HashiCorp Terraform. En caso de usar VS Code, hay que descargar las extensiones de Terraform. Todo el código se puede meter en un solo archivo, pero es mejor diversificar.

En nuestro caso usaremos la nube de Azure (más adelante usaremos AWS si los tutoriales continúan), por ello escribiremos en el archivo **provider.tf**:

```
Provider "azurerm" {  
  Version = "~>2.8.0"  
  features {}  
}
```



AZ LOGIN

Guardamos el archivo y podemos sacar la consola. Lo primero que debemos hacer es hacer **az login** para loguearnos en Azure desde la consola.

```
C:\Users\frs\terraform>az login
You have logged in. Now let us find all the subscriptions to which you have access...
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "ede0fbe3-e7c6-4f96-8a8b-70bbc01b5329",
    "id": " ",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Azure for Students",
    "state": "Enabled",
```

Al ejecutar az login nos saldrá una ventana del navegador para loguearnos en Azure. Es turno de ejecutar nuestro primer comando y es `terraform init`

0_RG: Mi primer grupo de recursos

Terraform init

Accedemos dentro de la carpeta `0_RG` y ejecutamos `terraform init`

```
C:\Users\frs\terraform\0_RG>terraform init

Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "azurerm" (hashicorp/azurerm) 2.8.0...

Terraform has been successfully initialized!
```

```
- Downloading plugin for provider "azurerm" (hashicorp/azurerm) 2.8.0...

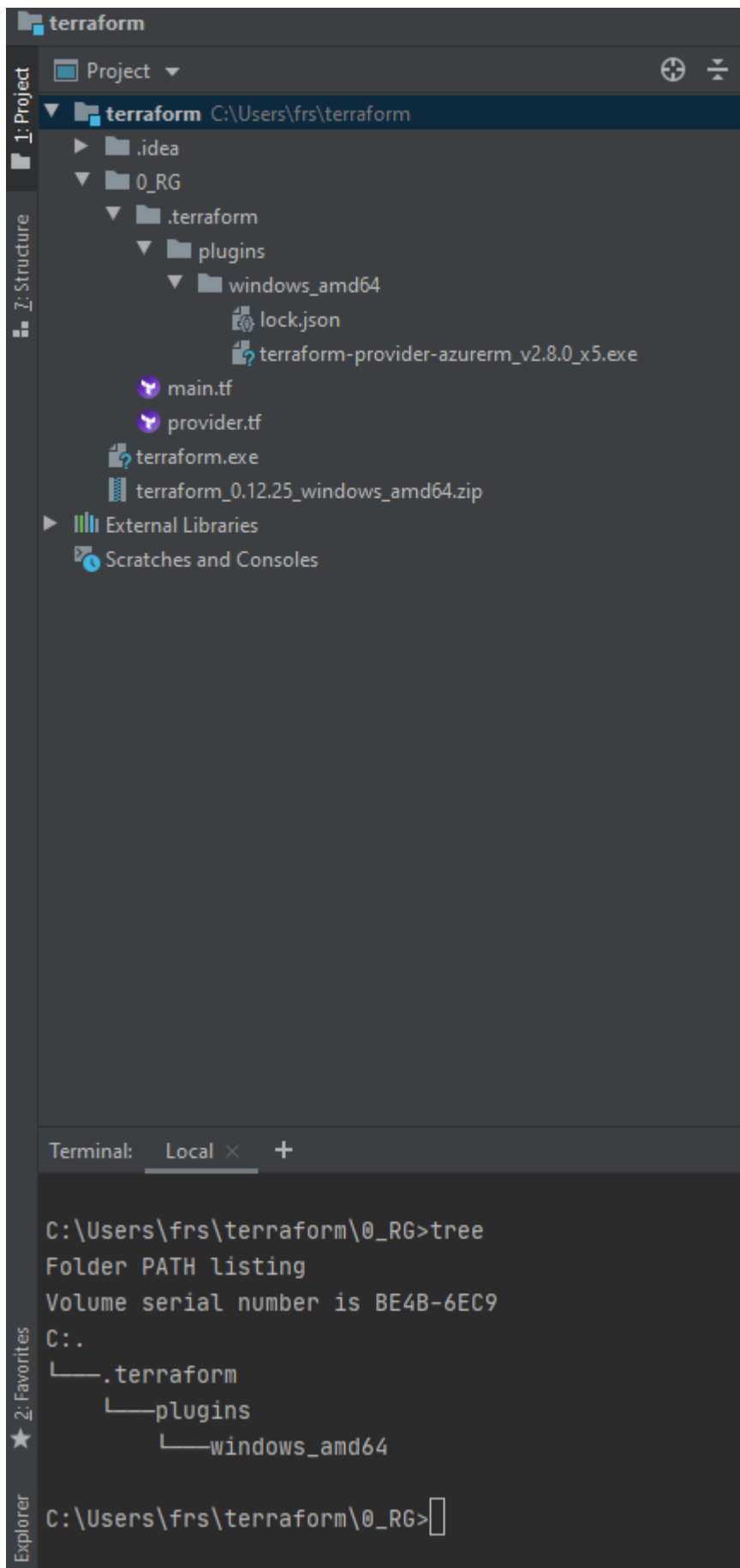
Terraform has been successfully initialized!

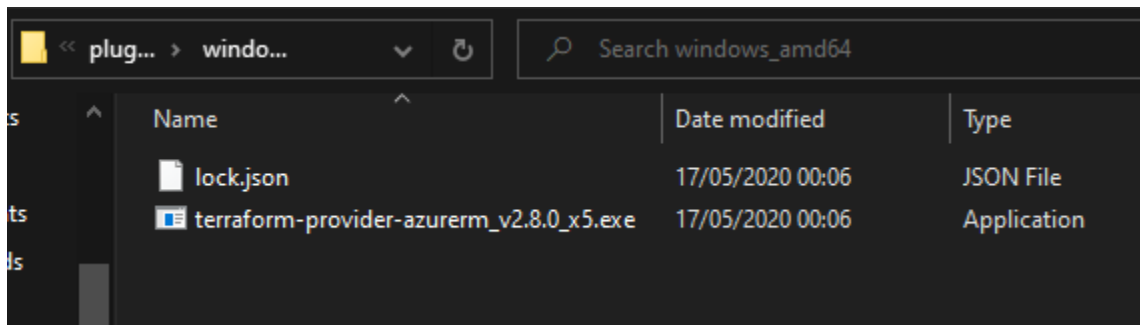
You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

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.

C:\Users\frs\terraform\0_RG>
```

Con la ejecución del comando nos ha descargado varios archivos:

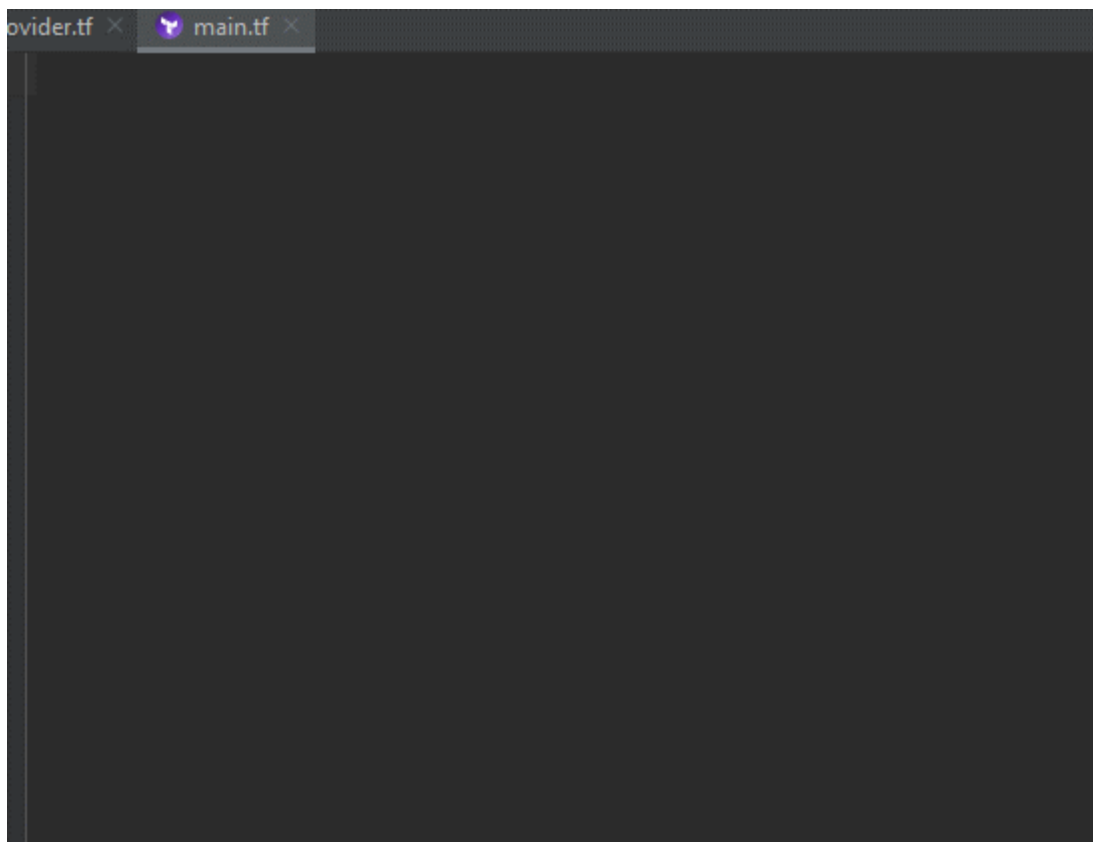
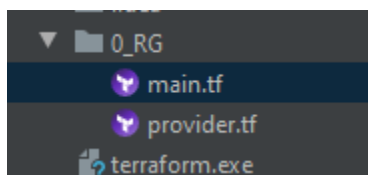




Con esto solo tenemos un archivo que le indica a terraform que proveedor usamos. Vamos a crear un grupo de recursos.

Crear nuestro primer grupo de recursos en Azure con terraform.

Creamos otro archivo que se llame **main.tf**:



Queremos hacer un grupo de recursos que esté en eastus2 y se llame 0_RG:

```
#Queremos un grupo de recursos en el este de estados unidos 2
#y que se llame 0_RG
resource "azurerm_resource_group" "" {
  location = ""
  name = ""
}
```

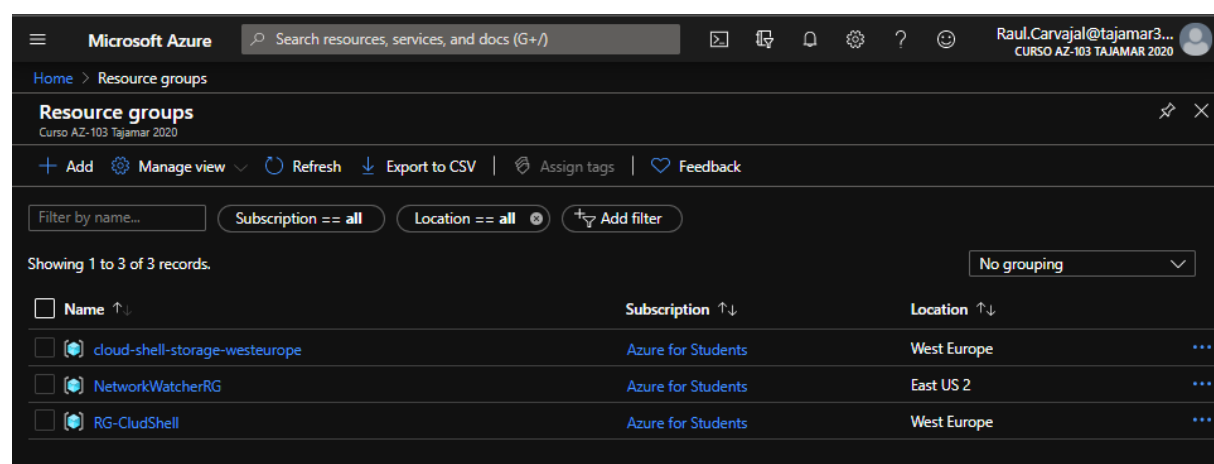
Rellenamos los campos:




```
#Queremos un grupo de recursos en el este de estados unidos 2
#y que se llame 0_RG.
resource "azurerm_resource_group" "rg" {
  #<- rg es el nombre que le damos al recurso azurerm_resource_group.
  location = "eastus2"
  #<- eastus2 en que zona se desplegará el recurso.
  name = "0_RG"
  #<- El nombre como nombraremos el recurso.
}
```

Versión limpia:

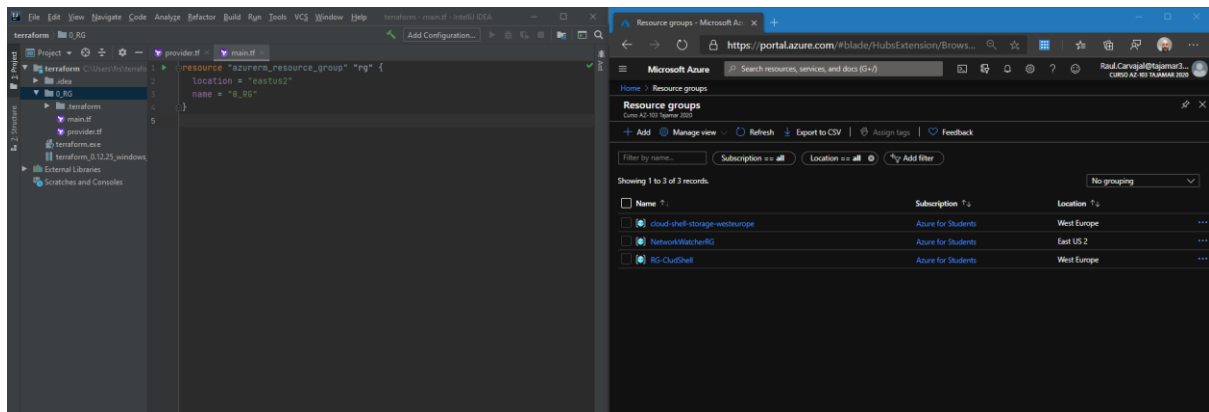
```
provider.tf x main.tf x
1 resource "azurerm_resource_group" "rg" {
2   location = "eastus2"
3   name = "0_RG"
4 }
5
```

Vamos a loguearnos con nuestro usuario en Azure en un navegador y accedemos al Blade de los grupos de recursos:



Resource groups		
Showing 1 to 3 of 3 records.		
Name ↑↓	Subscription ↑↓	Location ↑↓
 cloud-shell-storage-west europe	Azure for Students	West Europe
 NetworkWatcherRG	Azure for Students	East US 2
 RG-CloudShell	Azure for Students	West Europe

Ahora tengo en la misma pantalla el ide y el navegador:



Ejecutamos por primera vez Terraform plan

Vamos a ejecutar terraform plan en el terminal, recordad que el lanzamiento de comando han de estar dentro de la carpeta con los .tf:

```
C:\Users\frs\terraform\0_RG>terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.
```

```
-----
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create
```

```
Terraform will perform the following actions:
```

```
# azurerm_resource_group.rg will be created
+ resource "azurerm_resource_group" "rg" {
  + id          = (known after apply)
  + location    = "eastus2"
  + name        = "0_RG"
}
```

Antes de aplicar el despliegue, repasaremos que nos indica la ejecución del terraform plan:

El símbolo + en verde significa que se van a crear los recursos y que no hay fallo en el código. Conocemos 2 parámetros, pero hasta que no se crea el recurso 0_RG no se sabe el id. Más adelante podremos montar cosas indicando que busque el id y desplegar en consecuencia.

```
Plan: 1 to add, 0 to change, 0 to destroy.

-----

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.

C:\Users\frs\terraform\0_RG>
```

La continuación nos indica que ya podemos hacer un apply. Nos indica el plan que hay 1 que añadir 0 que modificar y 0 que eliminar.

Ejecución de terraform graph.

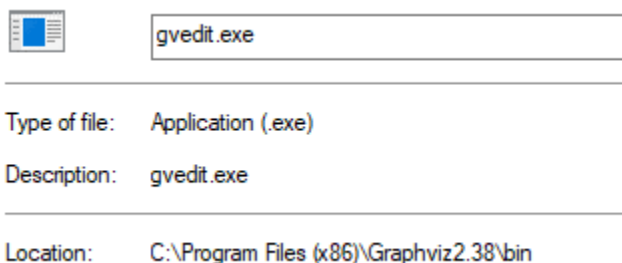
Vamos a hacer una pequeña demostración que va a hacer con terraform apply.

Ejecutamos: `terraform graph -draw-cycles > 0_RG.svg`

```
C:\Users\frs\terraform\0_RG>terraform graph -draw-cycles > 0_RG.svg

C:\Users\frs\terraform\0_RG>
```

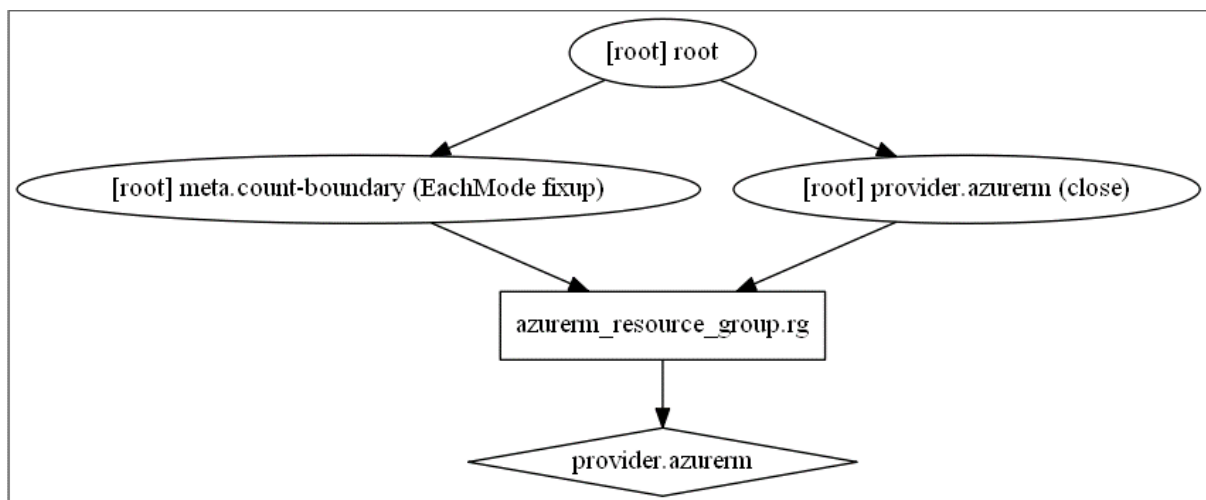
Vemos que no hay output del comando aparente, pero dentro de la carpeta hay un archivo svg con que ejecutaremos con graphviz (la ruta:



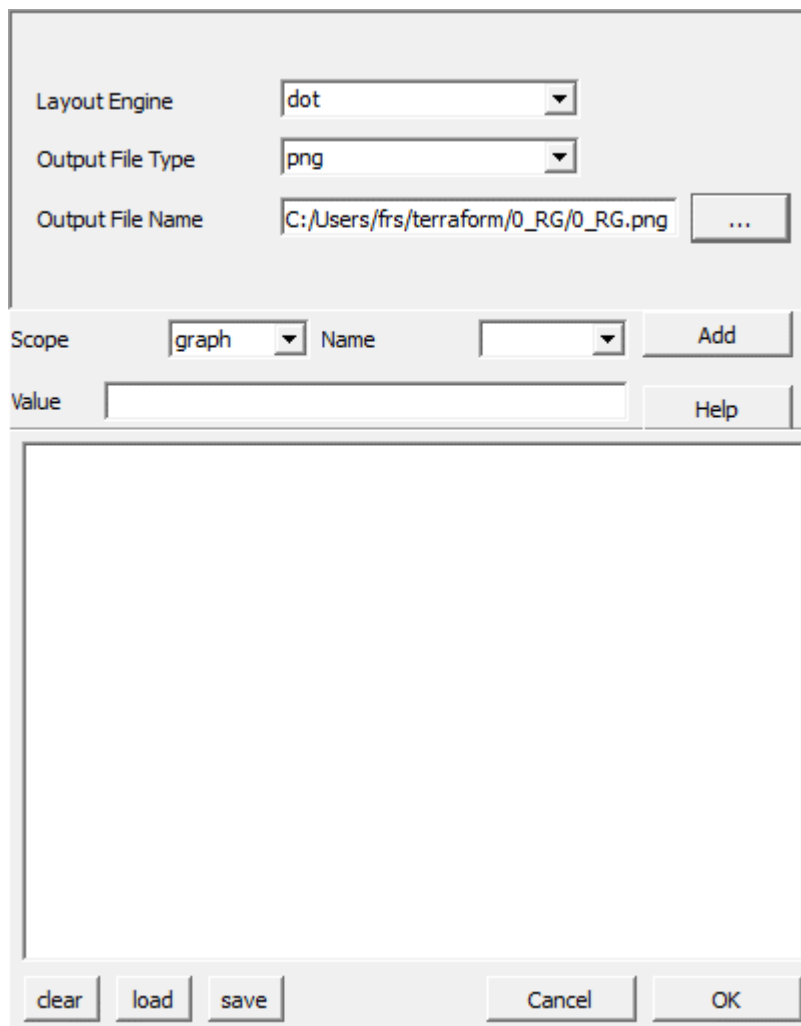
C:\Program Files (x86)\Graphviz2.38\bin\gvedit.exe)

Name	Date modified	Type	Size
.terraform	17/05/2020 00:06	File folder	
0_RG.svg	17/05/2020 00:35	SVG File	
main.tf	17/05/2020 00:21	TF File	
provider.tf	16/05/2020 23:57	TF File	

Abrimos el archive svg con el programa gvedit.exe



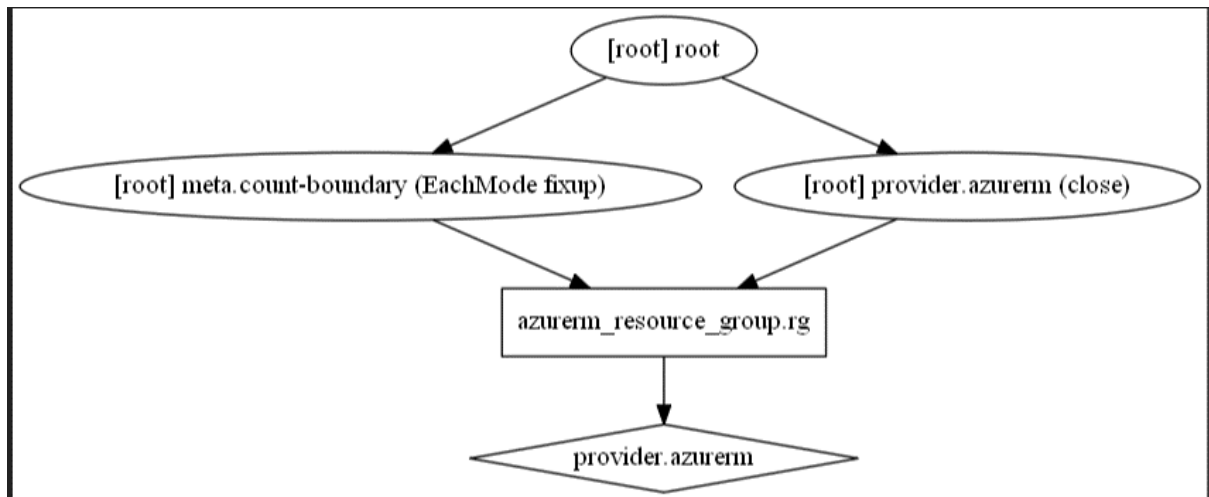
Tenemos nuestra representación grafica de nuestro despliegue. Tenemos nuestro proveedor y nuestro recurso. Ahora queremos sacar un png para que nuestros compañeros nos ayuden con un despliegue, nos dirigimos a graph->settings:



Si pinchamos en ok, por defecto nos generará un png con el nombre igual, pero con la extensión png que podremos abrir con el navegador, etc...

Name	Date modified	Type	Size
.terraform	17/05/2020 00:06	File folder	
0_RG.png	17/05/2020 00:44	PNG File	
0_RG.svg	17/05/2020 00:35	SVG File	
main.tf	17/05/2020 00:21	TF File	
provider.tf	16/05/2020 23:57	TF File	

Podemos cerrar el programa svg y ejecutamos el png.



Ya podemos compartir con el resto de gente facilmente.

Ejecución de terraform apply.

Una vez mostrado que es lo que va hacer terraform, ejecutamos terraform apply:

```
+ resource "azurerm_resource_group" "rg" {
  + id      = (known after apply)
  + location = "eastus2"
  + name    = "0_RG"
}
```

Plan: 1 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:

Al ejecutar, nos revisa el código y nos pide que escribamos yes para que lo ejecute. Tras escribir yes, vemos como empieza a generar el código en azure:

```

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

azurerm_resource_group.rg: Creating...
azurerm_resource_group.rg: Creation complete after 1s [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

C:\Users\frs\terraform\0_RG>

```

Refrescamos el grupo de recursos en azure:

Resource groups
Curso AZ-103 Tajamar 2020

+ Add ⚙️ Manage view ▾ ↻ Refresh ⬇️ Export to CSV | 🏷️ Assign tags | ❤️ Feedback

Filter by name... Subscription == all Location == all ⚙️ + Add filter

Showing 1 to 4 of 4 records. No groups

<input type="checkbox"/> Name ↑↓	Subscription ↑↓	Location ↑↓
<input type="checkbox"/> 0_RG	Azure for Students	East US 2
<input type="checkbox"/> cloud-shell-storage-west europe	Azure for Students	West Europe
<input type="checkbox"/> NetworkWatcherRG	Azure for Students	East US 2
<input type="checkbox"/> RG-CloudShell	Azure for Students	West Europe

Ye tenemos nuestro primer recurso creado en Azure usando terraform.

Usando terraform destroy para deshacer lo creado.

Una vez creado el recurso queremos ahorrar y borramos el recurso:


```

- destroy

Terraform will perform the following actions:

# azurerm_resource_group.rg will be destroyed
- resource "azurerm_resource_group" "rg" {
  - id      = "/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG" -> null
  - location = "eastus2" -> null
  - name     = "0_RG" -> null
  - tags     = {} -> null
}

Plan: 0 to add, 0 to change, 1 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: 

```

Vemos como ahora nos indica con un – en color rojo que va a eliminar ese recurso. Escribimos yes.

```

There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes

azurerm_resource_group.rg: Destroying... [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG]
azurerm_resource_group.rg: Still destroying... [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG, 10s elapsed]
azurerm_resource_group.rg: Still destroying... [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG, 20s elapsed]
azurerm_resource_group.rg: Still destroying... [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG, 30s elapsed]
azurerm_resource_group.rg: Still destroying... [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG, 40s elapsed]
azurerm_resource_group.rg: Destruction complete after 48s

Destroy complete! Resources: 1 destroyed.

C:\Users\frs\terraform\0_RG>

```

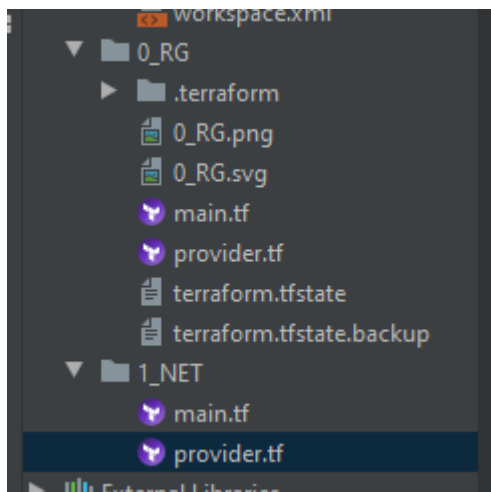
Vemos que ha tardado bastante más que en crear el recurso.

Ya tenemos lo mínimo para comprender el funcionamiento de terraform con recursos creados desde cero. Con recursos existente hay que importar y de momento no trabajaremos con recursos existentes.

1_NET: Mi primera red virtual

Partiendo del paso 0 con nuestro grupo de recursos creado, vamos a crear nuestra primera Virtual network y una subred.

Copiamos los 2 archivos tf del ejercicio 0 en la carpeta 1:



Una vez los tenemos, abrimos la terminal del ide y nos loqueamos en azure con `az login`

You have logged into Microsoft Azure!

You can close this window, or we will redirect you to the [Azure CLI documents](#) in 10 seconds.

Una vez que nos aparece así ya podemos cerrar la pestaña y volver al ide.

Ejecutamos: `terraform init`

```
- Downloading plugin for provider "azurerm" (hashicorp/azurerm) 2.8.0...

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.

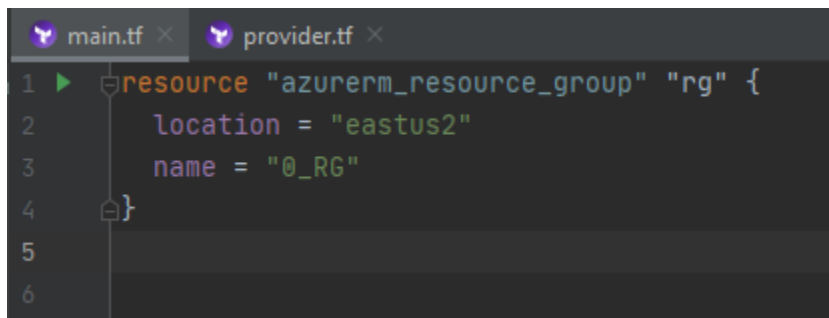
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

C:\Users\frs\Desktop\terraform\1_NET>
```

Esto nos vuelve a bajar el ejecutable que teníamos en la carpeta 0, como son ejercicios que vamos de menos a más, podemos eliminar la carpeta 0 tranquilamente.

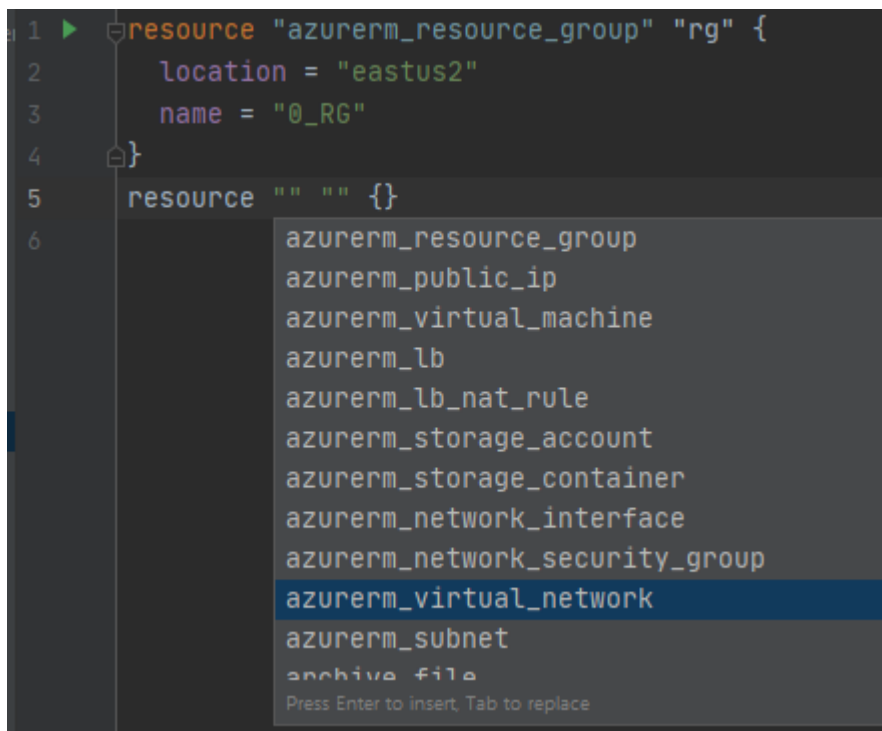
Vamos a ir traspasando del ejercicio 0 hasta que al final consigamos hacer una maquina virtual con todos los pasos en carpetas consecutivamente.

Partimos del archivo main.tf que contiene solamente la configuración de un grupo de recursos que se llama 0_RG y que estará ubicado en el este de estados unidos 2.



```
1 resource "azurerm_resource_group" "rg" {
2   location = "eastus2"
3   name     = "0_RG"
4 }
5
6
```

Escribimos `resource` y el ide nos tiene que mostrar las opciones disponibles, seleccionamos virtual network o escribimos: `azurerm_virtual_network`

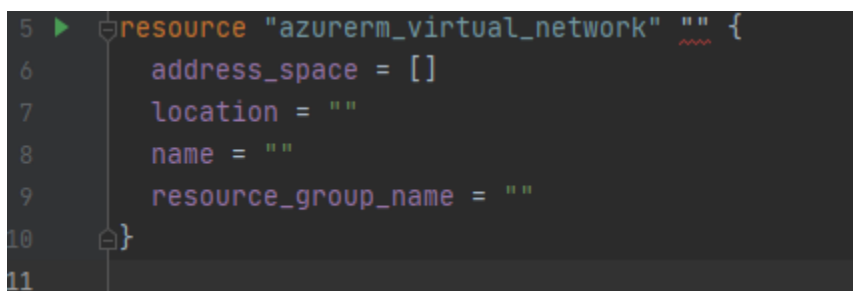


```
1 resource "azurerm_resource_group" "rg" {
2   location = "eastus2"
3   name     = "0_RG"
4 }
5 resource "" "" {}
6
```

- azurerm_resource_group
- azurerm_public_ip
- azurerm_virtual_machine
- azurerm_lb
- azurerm_lb_nat_rule
- azurerm_storage_account
- azurerm_storage_container
- azurerm_network_interface
- azurerm_network_security_group
- azurerm_virtual_network**
- azurerm_subnet
- archive_file

Press Enter to insert, Tab to replace

Al seleccionar virtual_network nos generará un código básico:



```
5 resource "azurerm_virtual_network" "" {
6   address_space = []
7   location      = ""
8   name          = ""
9   resource_group_name = ""
10 }
11
```

- El primer par de `""`: Es el identificador del recurso.
- Address_space: Espacio de direcciones que usaremos [`"CDIR"`]

- Location: Lugar donde se ubicará la virtual network.
- Name: El nombre que le pondremos a la virtual network.
- Resource_group_name: Grupo de recursos asociado.

Vamos a ir rellenando los campos de la red virtual

Para seguir con los ejercicios en el campo name, añadiremos: 1_Vnet (No siempre dejará añadir un guion bajo en el nombre).

```
name = "1_VNet"
```

Ahora, ¿A qué grupo de recursos lo queremos unir? Al de 0_RG, ¿Y cómo lo hacemos? Remplazaremos las comillas por:

```
azurerm_resource_group.rg.name
```

¿Qué significa esto? Estamos aprovechando el código para rellenar campos. Llamamos a: azurerm_resource_group.rg.name, que contiene como valor 0_RG.

Hacemos lo mismo con el campo location:

```
azurerm_resource_group.rg.location
```

Hemos vuelto a reutilizar el código del grupo de recursos añadiendo al final location. Esto hace que coja la localización del recurso.

Nos toca el campo address_space:

```
address_space = ["10.0.0.0/16"]
```

Añadimos el espacio de nombres en formato CIDR entre comillas.

Nos queda el último campo. El identificador del recurso, escribiremos: vnet

```
resource "azurerm_virtual_network" "vnet" {
  address_space = ["10.0.0.0/16"]
  location = azurerm_resource_group.rg.location
  name = "1_VNet"
  resource_group_name = azurerm_resource_group.rg.name
}
```

Ya tenemos nuestra primera red virtual básica.

Terraform plan

Resource actions are indicated with the following symbols:

+ create

Terraform will perform the following actions:

```
# azurerm_resource_group.rg will be created
+ resource "azurerm_resource_group" "rg" {
  + id          = (known after apply)
  + location    = "eastus2"
  + name        = "0_RG"
}

# azurerm_virtual_network.vnet will be created
+ resource "azurerm_virtual_network" "vnet" {
  + address_space = [
    + "10.0.0.0/16",
  ]
  + guid          = (known after apply)
  + id            = (known after apply)
  + location      = "eastus2"
  + name          = "1_VNet"
  + resource_group_name = "0_RG"
  + subnet        = (known after apply)
}
```

Los cambios que realizará si realizamos un terraform apply. Vemos como coge los campos correctamente. Existen las variables, pero más adelante realizaremos los mismos pasos, pero con variables.

Nos queda un detalle, no tiene subred la red virtual, vamos a añadir la subred.

Subnet.

Escribimos resource "azurerm_subnet" "":

```
resource "azurerm_subnet" "subnet" {
  address_prefix = "10.0.1.0/24"
  name           = "subnet"
  resource_group_name = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.vnet.name
}
```

```

resource "azurerm_subnet" "subnet" {
  address_prefix = "10.0.1.0/24"
  name = "subnet"
  resource_group_name = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.vnet.name
}

```

```

resource "azurerm_virtual_network" "vnet" {
  address_space = ["10.0.0.0/16"]
  location = var.location
  name = "1_VNet"
  resource_group_name = azurerm_resource_group.rg.name
}

resource "azurerm_subnet" "subnet" {
  address_prefix = "10.0.1.0/24"
  name = "subnet"
  resource_group_name = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.vnet.name
}

```

Ejecutamos `terraform plan`:

```

+ resource "azurerm_subnet" "subnet" {
  + address_prefix           = "10.0.1.0/24"
  + address_prefixes         = (known after apply)
  + enforce_private_link_endpoint_network_policies = false
  + enforce_private_link_service_network_policies = false
  + id                       = (known after apply)
  + name                     = "subnet"
  + resource_group_name      = "0_RG"
  + virtual_network_name     = "1_VNet"
}

```

Vamos a visualizar el despliegue mediante `terraform graph`:

`Terraform graph -draw-cycles > vnet.svg`

```

C:\Users\frs\Desktop\terraform\1_NET>terraform graph -draw-cycles > vnet.svg

C:\Users\frs\Desktop\terraform\1_NET>

```

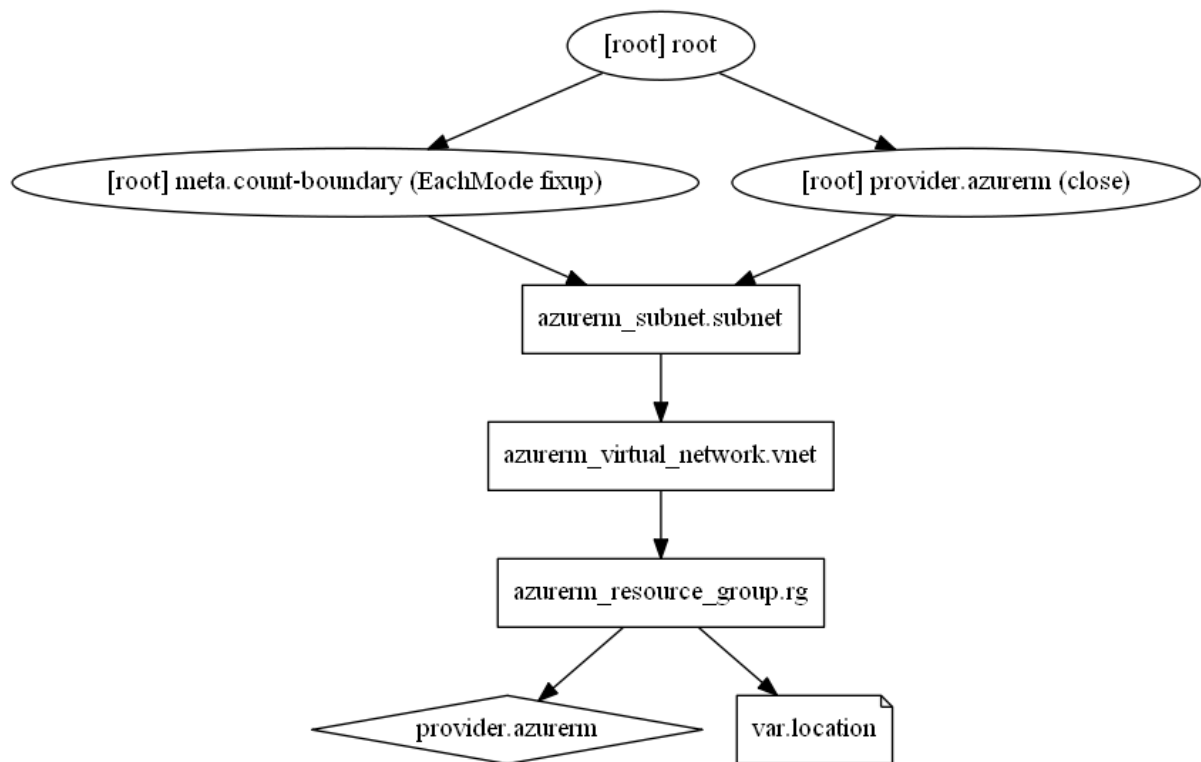
Accedemos a la carpeta y ejecutamos el `svg->graph settings` o pinchamos el

icono: 

Dejamos el formato por defecto en png (se puede cambiar el formato de salida a jpg, pdf, jpeg, etc...) y pinchamos en ok.

Terraform graph:

Abrimos el png y observamos el despliegue. Realizará un grupo de recursos y de él crearemos una red virtual.



Realizamos `terraform apply` y escribimos yes cuando lo solicite.

```
# azurerm_virtual_network.vnet will be created
+ resource "azurerm_virtual_network" "vnet" {
  + address_space      = [
    + "10.0.0.0/16",
  ]
  + guid               = (known after apply)
  + id                 = (known after apply)
  + location            = "eastus2"
  + name               = "1_VNet"
  + resource_group_name = "0_RG"
  + subnet              = [
    + {
      + address_prefix = "10.0.1.0/24"
      + id              = (known after apply)
      + name             = "Subnet"
      + security_group  = ""
    },
  ]
}
```

Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?

Plan: 2 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

Accedemos a la página web de azure y comprobamos los cambios.

```
azurerm_resource_group.rg: Creating...
azurerm_resource_group.rg: Creation complete after 2s [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG]
azurerm_virtual_network.vnet: Creating...
azurerm_virtual_network.vnet: Still creating... [10s elapsed]
azurerm_virtual_network.vnet: Creation complete after 17s [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG/providers/Microsoft.Network/virtualNetworks/1_VNet]
```

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

C:\Users\frs\Desktop\terraform\1_NET>

Home > Resource groups > 0_RG

0_RG
Resource group

Search (Ctrl+/) <<

+ Add Edit columns Delete resource group Refresh Move Export to CSV ...

Overview

Activity log

Access control (IAM)

Tags

Events

Settings

Quickstart

Deployments

Policies

Properties

Subscription (change)
Azure for Students

Deployments
No deployments

Subscription ID
63b1d439-1468-4cc7-95cc-8ee4cf0fea0f

Tags (change)
Click here to add tags

Filter by name... Type == all Location == all Add filter

Showing 1 to 1 of 1 records. Show hidden types No grouping

Name	Type	Location
1_VNet	Virtual network	East US 2

Si pinchamos en la virtual network y comprobamos la subred:

Home > Resource groups > 0_RG > 1_VNet

1_VNet
Virtual network

Search (Ctrl+/) <<

Refresh Move Delete

Overview

Activity log

Access control (IAM)

Tags

Resource group (change)
0_RG

Address space
10.0.0.0/16

Location
East US 2

DNS servers
Azure provided DNS service

Subscription (change)
Azure for Students

Home > Resource groups > 0_RG > 1_VNet | Subnets

1_VNet | Subnets
Virtual network

Search (Ctrl+/) <<

+ Subnet + Gateway subnet Refresh

Search subnets

Name	Address range	IPv4 available ad...	Delegated to
Subnet	10.0.1.0/24	251	-

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Settings

Address space

Connected devices

Subnets

DDoS protection

Vemos que nos ha realizado: Un grupo de recursos, una virtual network y una subred.

Realizamos un `terraform destroy` para deshacer el despliegue (es importante que con nuestro plan de estudiante eliminemos todo aquello que consume crédito).

```
# azurerm_virtual_network.vnet will be destroyed
- resource "azurerm_virtual_network" "vnet" {
  - address_space      = [
    - "10.0.0.0/16",
  ] -> null
  - dns_servers        = [] -> null
  - guid               = "ab03a3e2-a229-4a18-ab4e-cb4ed13661a0" -> null
  - id                 = "/subscriptions/63b1d439-1468-4cc7-95cc-8ee4c
Microsoft.Network/virtualNetworks/1_VNet" -> null
  - location           = "eastus2" -> null
  - name               = "1_VNet" -> null
  - resource_group_name = "0_RG" -> null
  - subnet              = [
    - {
      - address_prefix = "10.0.1.0/24"
      - id              = "/subscriptions/63b1d439-1468-4cc7-95cc-8e
rs/Microsoft.Network/virtualNetworks/1_VNet/subnets/Subnet"
      - name            = "Subnet"
      - security_group  = ""
    },
  ] -> null
  - tags               = {} -> null
}
```

Escribimos yes y ejecutamos.

```
Enter a value: yes

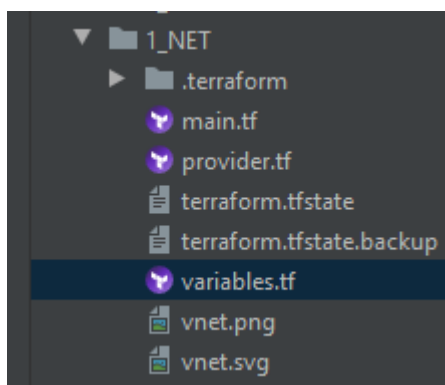
azurerm_virtual_network.vnet: Destroying... [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/0_RG/providers/Microsoft.Network/virtualNetworks/1_VNet]
azurerm_virtual_network.vnet: Destruction complete after 2s
azurerm_resource_group.rg: Destroying... [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/0_RG]
azurerm_resource_group.rg: Still destroying... [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/0_RG, 10s elapsed]
azurerm_resource_group.rg: Still destroying... [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/0_RG, 20s elapsed]
azurerm_resource_group.rg: Still destroying... [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/0_RG, 30s elapsed]
azurerm_resource_group.rg: Still destroying... [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/0_RG, 40s elapsed]
azurerm_resource_group.rg: Destruction complete after 48s

Destroy complete! Resources: 2 destroyed.

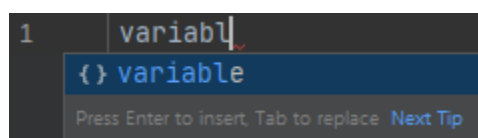
C:\Users\frs\Desktop\terraform\1_NET>
```

Mis primeras variables.

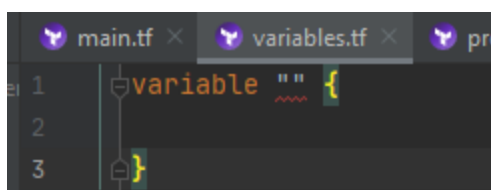
Vamos a crear un archivo `variables.tf` en el directorio `1_NET`:



Vamos a crear nuestra primera variable de tipo string para el campo location.



Escribiremos variable y nos genera el código:



Vamos a rellenar la variable. Primer par de "" el nombre de la variable: location

```
variable "location" {  
}
```

Dentro de las llaves escribiremos `type = string`

```
variable "location" {  
  type = string  
}
```

Tenemos que darle un valor a esa variable, escribiremos:

Default = "eastus2"

```
variable "location" {  
  type = string  
  default = "eastus2"  
}
```

Tenemos nuestra primera variable para que cada vez que nos pida el valor de location no tengamos que reusar código y hacerlo de manera más simple.

Vamos a hacerlo. Volvamos a `main.tf` y remplazaremos todo lo que contenga `location = "azurerm..."` por `var.location`:

```
resource "azurerm_resource_group" "rg" {  
  location = "eastus2"  
  name = "0_RG"  
}  
  
resource "azurerm_virtual_network" "vnet" {  
  address_space = ["10.0.0.0/16"]  
  location = azurerm_resource_group.rg.location  
  name = "1_VNet"  
  resource_group_name = azurerm_resource_group.rg.name  
  subnet {  
    address_prefix = "10.0.1.0/24"  
    name = "Subnet"  
  }  
}
```

Después

```

main.tf x variables.tf x provider.tf x
> resource "azurerm_resource_group" "rg" {
  location = var.location
  name     = "0_RG"
}
> resource "azurerm_virtual_network" "vnet" {
  address_space = ["10.0.0.0/16"]
  location      = var.location
  name          = "1_VNet"
  resource_group_name = azurerm_resource_group.rg.name
  subnet {
    address_prefix = "10.0.1.0/24"
    name           = "Subnet"
  }
}

```

Volvemos a ejecutar `terraform plan`:

```

# azurerm_resource_group.rg will be created
+ resource "azurerm_resource_group" "rg" {
  + id          = (known after apply)
  + location    = "eastus2"
  + name        = "0_RG"
}

# azurerm_virtual_network.vnet will be created
+ resource "azurerm_virtual_network" "vnet" {
  + address_space = [
    + "10.0.0.0/16",
  ]
  + guid          = (known after apply)
  + id            = (known after apply)
  + location      = "eastus2"
  + name          = "1_VNet"
  + resource_group_name = "0_RG"
  + subnet        = [
    + {
      + address_prefix = "10.0.1.0/24"
      + id              = (known after apply)
      + name            = "Subnet"
    }
  ]
}

```

Vemos como nos coge la variable que declaremos en el archivo `variables.tf`.

Desplegamos y comprobamos que se ha realizado lo mismo que declarando el código entero o introduciendo eastus2 en cada campo location.

```
azurerm_resource_group.rg: Creating...
azurerm_resource_group.rg: Creation complete after 1s [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG]
azurerm_virtual_network.vnet: Creating...
azurerm_virtual_network.vnet: Still creating... [10s elapsed]
azurerm_virtual_network.vnet: Still creating... [20s elapsed]
azurerm_virtual_network.vnet: Creation complete after 28s [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG/providers/Microsoft.Network/virtualNetworks/1_VNet]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

C:\Users\frs\Desktop\terraform\1_NET>
```

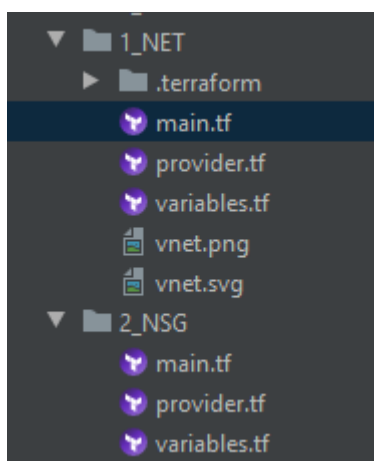
Vemos que se ha realizado el despliegue igual que antes, pero sin necesidad de introducir el lugar donde se desplegarán los recursos cada vez. Tenemos una variable que hemos declarado una vez y podemos reutilizar cada vez y sin miedo a equivocarnos.

`Terraform destroy` para finalizar el ejercicio 1.

2_NSG: Network Security Group.

En este ejercicio vamos a realizar un NSG para poder conectarnos en remoto y el puerto 80 para el servidor web.

Partimos del ejercicio anterior:



Reutilizaré los archivos para la nueva carpeta, para así tener diferenciados los archivos de cada ejercicio.

Comenzamos creando el nsg tecleando `resource:`

```
resource "azurerm_network_security_group" "" {
  location = ""
  name = ""
  resource_group_name = ""
}
```

Empezamos modificando el identificador del nsg: nsg

```
resource "azurerm_network_security_group" "nsg" {
  location = var.location
  name = "NSG"
  resource_group_name = azurerm_resource_group.rg.name
}
```

Añadimos el nombre, la variable de la región y la ruta del grupo de recursos.

Security rule.

Ahora justo después de name y antes de la ultima llave, pulsamos enter para declarar nuestra primera regla en el nsg:

```
resource "azurerm_network_security_group" "nsg" {
  location = var.location
  name = "NSG"
  resource_group_name = azurerm_resource_group.rg.name
  |
}
```

Escribimos security rule:

```
security_rule {
  access = ""
  direction = ""
  name = ""
  priority = 0
  protocol = ""
}
```

Rellenamos los campos con la información siguiente:

```
security_rule {
  access = "Allow"
  direction = "Inbound"
```

```
name = "ssh"
priority = 110
protocol = "Tcp"
source_port_range      = "*"
destination_port_range = "22"
source_address_prefix  = "*"
destination_address_prefix = "*"
}
```

Ahora añadimos la siguiente regla:

```
security_rule {
    access = "Allow"
    direction = "Inbound"
    name = "web"
    priority = 120
    protocol = "Tcp"
    source_port_range      = "*"
    destination_port_range = "80"
    source_address_prefix  = "*"
    destination_address_prefix = "*"
}
```

Ya tenemos nuestras 2 reglas del NSG básicas. El resultado del NSG quedaría tal que así:


```

resource "azurerm_network_security_group" "nsg" {
  location = var.location
  name     = "NSG"
  resource_group_name = azurerm_resource_group.rg.name

  security_rule {
    access = "Allow"
    direction = "Inbound"
    name = "ssh"
    priority = 110
    protocol = "Tcp"
    source_port_range = "*"
    destination_port_range = "22"
    source_address_prefix = "*"
    destination_address_prefix = "*"
  }

  security_rule {
    access = "Allow"
    direction = "Inbound"
    name = "web"
    priority = 120
    protocol = "Tcp"
    source_port_range = "*"
    destination_port_range = "80"
    source_address_prefix = "*"
    destination_address_prefix = "*"
  }
}

```

Las reglas por defecto no hay que crearlas, automáticamente se crean.

Realizamos un `terraform plan` o `terraform validate` a ver si hay algún error en el código:

```
# azurerm_network_security_group.nsg will be created
+ resource "azurerm_network_security_group" "nsg" {
  + id                  = (known after apply)
  + location            = "eastus2"
  + name               = "NSG"
  + resource_group_name = "0_RG"
  + security_rule       = [
    + {
      + access                = "Allow"
      + description           = ""
      + destination_address_prefix = "*"
      + destination_address_prefixes = []
      + destination_application_security_group_ids = []
      + destination_port_range = "22"
      + destination_port_ranges = []
      + direction             = "Inbound"
      + name                  = "ssh"
      + priority              = 110
      + protocol              = "Tcp"
      + source_address_prefix = "*"
      + source_address_prefixes = []
      + source_application_security_group_ids = []
      + source_port_range     = "*"
      + source_port_ranges    = []
    },
    + {
      + access                = "Allow"
      + description           = ""
      + destination_address_prefix = "*"
      + destination_address_prefixes = []
      + destination_application_security_group_ids = []
      + destination_port_range = "80"

```

```
      + destination_port_ranges = []
      + direction             = "Inbound"
      + name                  = "web"
      + priority              = 120
      + protocol              = "Tcp"
      + source_address_prefix = "*"
      + source_address_prefixes = []
      + source_application_security_group_ids = []
      + source_port_range     = "*"
      + source_port_ranges    = []
    },
  ]
}
```

Vemos que no nos ha salido ningún error. Ahora nos falta

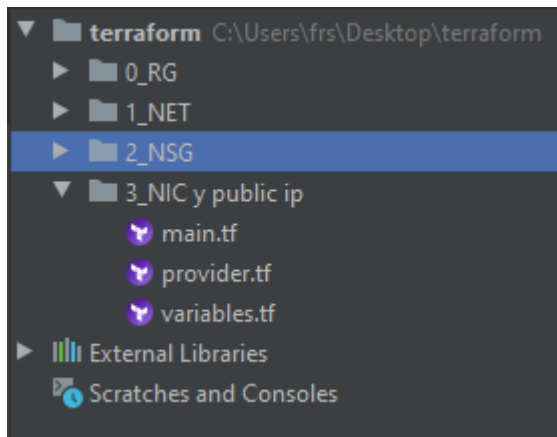
También se puede realizar por separado:

```
resource "azurerm_network_security_rule" "nsg-ssh" {
  access            = "Allow"
  direction         = "Inbound"
  name              = "SSH"
  network_security_group_name = azurerm_network_security_group.nsg.name
  priority          = 110
  protocol          = "Tcp"
  destination_port_range = "22"
  source_address_prefix = "*"
  destination_address_prefix = "*"
  resource_group_name   = azurerm_resource_group.rg.name
  source_port_range     = "*"
}
resource "azurerm_network_security_rule" "nsg-web" {
  access            = "Allow"
  direction         = "Inbound"
  name              = "WEB"
  network_security_group_name = azurerm_network_security_group.nsg.name
  priority          = 111
  protocol          = "Tcp"
  destination_port_range = "80"
  source_address_prefix = "*"
  destination_address_prefix = "*"
  resource_group_name   = azurerm_resource_group.rg.name
  source_port_range     = "*"
}
```

3_NIC y Public ip.

*Es importante hacer az login antes de lanzar el comando terraform apply si has ido haciendo los ejercicios en diferentes días. También es necesario eliminar los *.tfstate y *.tfstate.backup de las carpetas anteriores puesto que puede dar problemas al importar recursos.*

Como siempre, partimos copiando los archivos *.tf del anterior ejercicio y copiándolos en la carpeta del nuevo ejercicio:



Vamos a crear una NIC y una public ip.

Public IP:

Escribimos `resource` e introducimos `azurerm_public_ip`, el intellisense introducirá lo básico a continuación:

```
4 resource "azurerm_public_ip" "" {  
5     location = ""  
6     name = ""  
7     resource_group_name = ""  
8 }
```

Ponemos nombre al identificador: `pip`

Region: `var.location`

Nombre que recibirá el objeto ip publica: `pip`

Grupo de recursos: `azurerm_resource_group.rg.name`

Introduciremos 2 líneas más: tipo de asignación y sku.

Tipo de asignación lo dejamos en `dinamico` y el sku en `básico`.

Para conocer las diferencias de sku en las ips públicas dejo el enlace para su visionado:

<https://docs.microsoft.com/es-es/azure/virtual-network/virtual-network-ip-addresses-overview-arm>

En resumen. Como no vamos a utilizar availability set ni balanceador de carga nos quedaremos con el sku básico.

Tendremos un resultado tal que así:

```
resource "azurerm_public_ip" "pip" {  
  location = var.location  
  name = "pip"  
  resource_group_name = azurerm_resource_group.rg.name  
  allocation_method = "Dynamic"  
  sku = "Basic"  
}
```

NIC:

Introducimos resource y completamos con azurerm_network_interface:

```
resource "azurerm_network_interface" "" {  
  location = ""  
  name = ""  
  resource_group_name = ""  
  ip_configuration {  
    name = ""  
    private_ip_address_allocation = ""  
  }  
}
```

Rellenamos los campos con el identificador: lo llamaremos nic a secas.

Región: variable var.location

Nombre: NIC

Resource_...: azurerm_resource_group.rg.name

Dentro de ip configuration irá:

Name: ipprivada

Private_ip_*_allocation: Dynamic para que sea automática la asignación de las ips

```

resource "azurerm_network_interface" "nic" {
  location = var.location
  name = "NIC"
  resource_group_name = azurerm_resource_group.rg.name
  ip_configuration {
    name = "ipprivada"
    private_ip_address_allocation = "Dynamic"
  }
}

```

Insertamos una línea en blanco después del tipo de asignación de la ip privada para añadir el identificador de la ip pública. Escribimos: `public_ip_address_id = ""`

```

resource "azurerm_network_interface" "nic" {
  location = var.location
  name = "NIC"
  resource_group_name = azurerm_resource_group.rg.name
  ip_configuration {
    name = "ipprivada"
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id = ""
  }
}

```

Lo rellenaremos con el id de la public ip:

```
public_ip_address_id = azurerm_public_ip.pip.id
```

```

resource "azurerm_network_interface" "nic" {
  location = var.location
  name = "NIC"
  resource_group_name = azurerm_resource_group.rg.name
  ip_configuration {
    name = "ipprivada"
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id = azurerm_public_ip.pip.id
  }
}

resource "azurerm_public_ip" "pip" {
  location = var.location
  name = "pip"
  resource_group_name = azurerm_resource_group.rg.name
  allocation_method = "Dynamic"
  sku = "Basic"
}

```

El *.id sirve para asignar la ip publica a la nic mediante el id. Como no sabemos la id hasta que se crea un objeto en Azure, terraform se esperará a que se construya el objeto para asignar la ip publica a la nic.

Añadimos la subnet_id:

```
subnet_id = azurerm_subnet.subnet.id
```

```
+ resource "azurerm_subnet" "subnet" {
  + address_prefix           = "10.0.1.0/24"
  + address_prefixes         = (known after apply)
  + enforce_private_link_endpoint_network_policies = false
  + enforce_private_link_service_network_policies = false
  + id                       = (known after apply)
  + name                     = "subnet"
  + resource_group_name      = "0_RG"
  + virtual_network_name     = "1_VNet"
}
```

Comprobación del código:

Lanzamos terraform validate:

```
C:\Users\frs\Desktop\terraform\3_NIC y public ip>terraform validate
Success! The configuration is valid.
```

Acto seguido si no hay problemas de código lanzamos un terraform plan:

```
# azurerm_public_ip.pip will be created
+ resource "azurerm_public_ip" "pip" {
  + allocation_method      = "Dynamic"
  + fqdn                   = (known after apply)
  + id                     = (known after apply)
  + idle_timeout_in_minutes = 4
  + ip_address              = (known after apply)
  + ip_version              = "IPv4"
  + location                = "eastus2"
  + name                   = "pip"
  + resource_group_name     = "0_RG"
  + sku                     = "Basic"
}
```

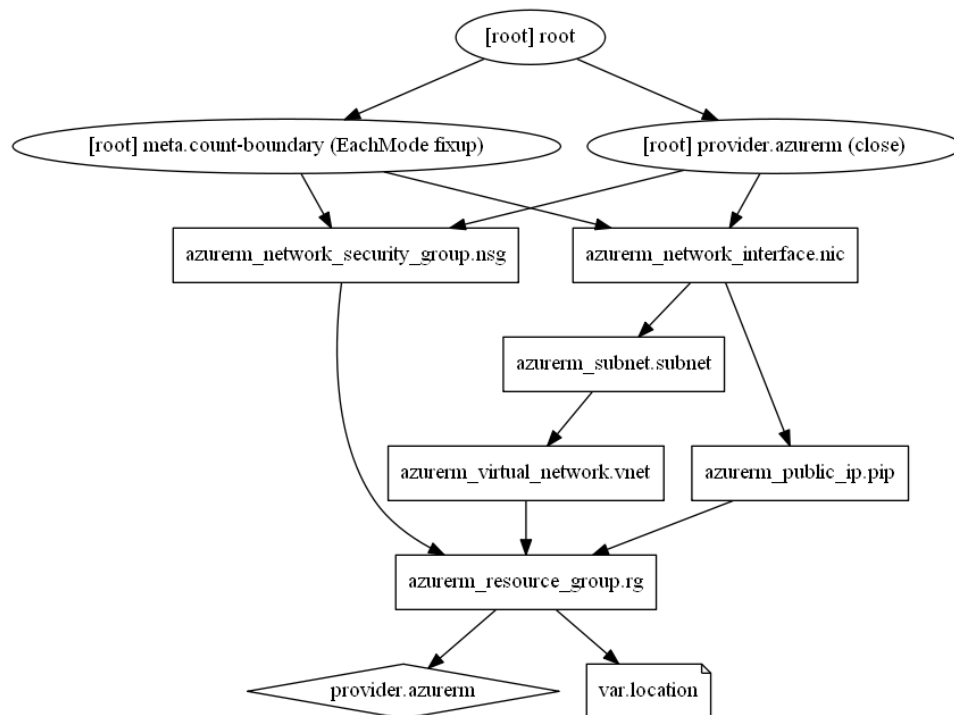
```
# azurerm_network_interface.nic will be created
+ resource "azurerm_network_interface" "nic" {
  + applied_dns_servers      = (known after apply)
  + dns_servers              = (known after apply)
  + enable_accelerated_networking = false
  + enable_ip_forwarding     = false
  + id                      = (known after apply)
  + internal_dns_name_label   = (known after apply)
  + internal_domain_name_suffix = (known after apply)
  + location                 = "eastus2"
  + mac_address              = (known after apply)
  + name                     = "NIC"
  + private_ip_address        = (known after apply)
  + private_ip_addresses      = (known after apply)
  + resource_group_name       = "0_RG"
  + virtual_machine_id        = (known after apply)

  + ip_configuration {
    + name                = "ipprivada"
    + primary              = (known after apply)
    + private_ip_address   = (known after apply)
    + private_ip_address_allocation = "dynamic"
    + private_ip_address_version  = "IPv4"
    + public_ip_address_id  = (known after apply)
  }
}
```

Vamos a visualizar gráficamente el despliegue:

Lanzamos: `terraform graph -draw-cycles > despliegue.svg`

Abrimos el SVG y lo exportamos a png para visualizarlo.



Tenemos nuestro despliegue visualmente estructurado. Evidentemente, nos falta la máquina virtual para terminar el despliegue completo, pero eso lo haremos en el próximo ejercicio.

Lanzamos el terraform apply y visualizamos el resultado en Azure:

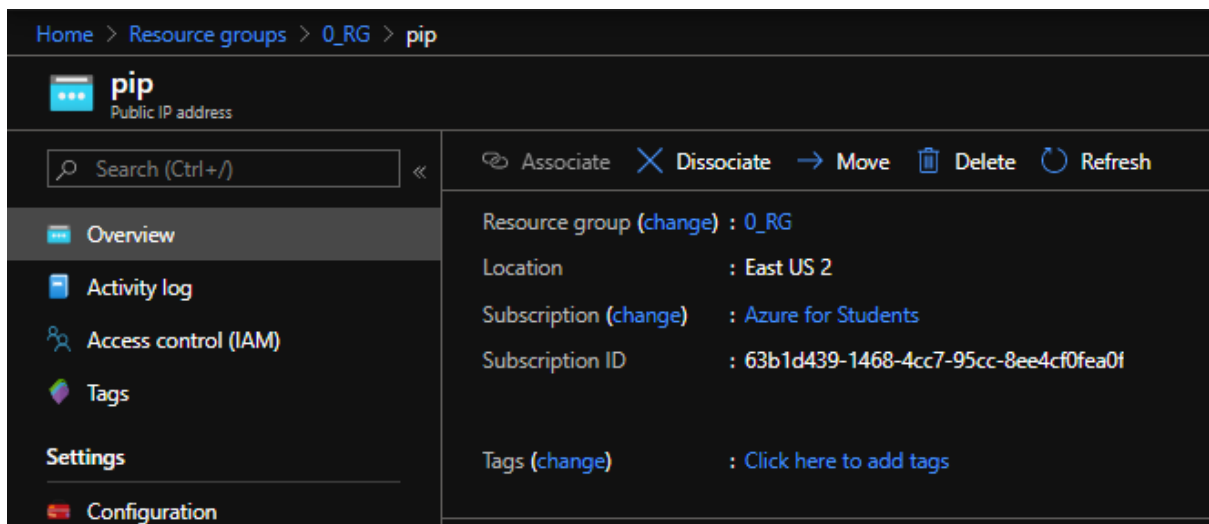
```

azurem_resource_group.rg: Creating...
azurem_resource_group.rg: Creation complete after 1s [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG]
azurem_virtual_network.vnet: Creating...
azurem_public_ip.pip: Creating...
azurem_network_security_group.nsg: Creating...
azurem_network_security_group.nsg: Creation complete after 5s [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG/providers/Microsoft.Network/networkSecurityGroups/NSG]
azurem_public_ip.pip: Creation complete after 5s [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG/providers/Microsoft.Network/publicIPAddresses/pip]
azurem_virtual_network.vnet: Still creating... [10s elapsed]
azurem_virtual_network.vnet: Creation complete after 15s [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG/providers/Microsoft.Network/virtualNetworks/1_VNet]
azurem_subnet.subnet: Creating...
azurem_subnet.subnet: Creation complete after 2s [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG/providers/Microsoft.Network/virtualNetworks/1_VNet/subnets/subnet]
azurem_network_interface.nic: Creating...
azurem_network_interface.nic: Creation complete after 5s [id=/subscriptions/63b1d439-1468-4cc7-95cc-8ee4cf0fea0f/resourceGroups/0_RG/providers/Microsoft.Network/networkInterfaces/NIC]

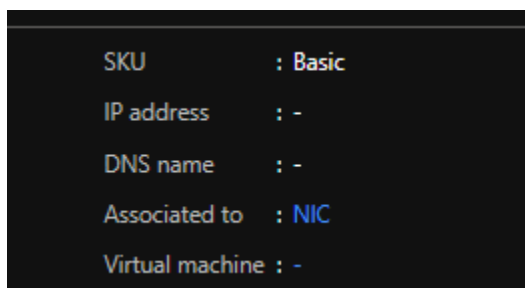
Apply complete! Resources: 6 added, 0 changed, 0 destroyed.
C:\Users\frs\Desktop\terraform\3_NIC y public ip>
  
```

Showing 1 to 4 of 4 records. ☐ Show hidden types

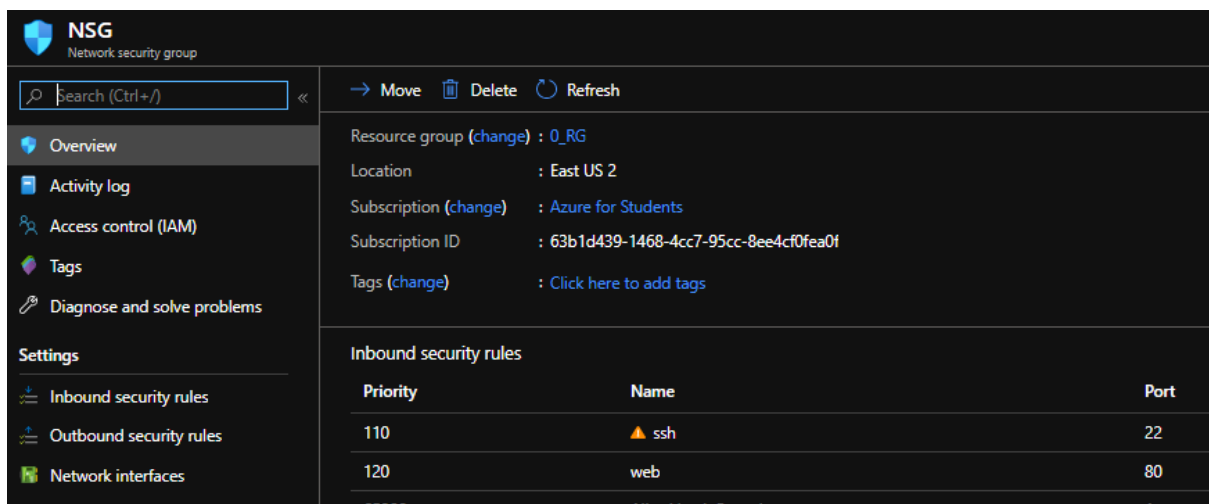
Name ↑↓	Type ↑↓	Location ↑↓
1_VNet	Virtual network	East US 2
NIC	Network interface	East US 2
NSG	Network security group	East US 2
pip	Public IP address	East US 2



Sku basic y sin ip puesto que no hay máquina virtual para asignar un ip:



Un NSG con las reglas:



Las reglas por defecto se crean igualmente, aunque no se declaren.

Una NIC con ip asignada y asociada las subnet:


```

resource "azurerm_virtual_machine" "VM" {
  location                = var.location
  name                    = "VM"
  network_interface_ids = [azurerm_network_interface.nic.id]
  resource_group_name     = azurerm_resource_group.rg.name
  vm_size                  = "Standard_B1s"
  storage_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "18.04-LTS"
    version   = "latest"
  }
  storage_os_disk {
    create_option = "FromImage"
    name          = "VM-OS"
    managed_disk_type = "Standard_LRS"
    caching        = "ReadWrite"
  }
  os_profile_linux_config {
    disable_password_authentication = false
  }
  os_profile {
    admin_username = var.usuario
    admin_password = var.password
    computer_name  = "VM"
  }
  boot_diagnostics {
    enabled      = false
    storage_uri = ""
  }
}

```

variables.tf

Para la declaración de la máquina, necesitamos pasarle el usuario y la contraseña, así que vamos a añadir 2 variables con el usuario y la contraseña:

```

variable "usuario" {
  type = string
  default = "usuario"
}
variable "password" {
  type = string
  default = "P4$$w0rd12345"
}

```

Script servidor web: scripts.tf

El siguiente archivo llama al script web.sh que hemos utilizado en el curso de Azure con Paco para la actualización e instalación de Apache y su correspondiente modificación de la página de inicio.

```
resource "azurerm_virtual_machine_extension" "installweb" {
  name = "installweb"
  publisher = "Microsoft.Azure.Extensions"
  type = "CustomScript"
  type_handler_version = "2.0"
  virtual_machine_id = azurerm_virtual_machine.VM.id

  settings = <<SETTINGS
  {
    "script": "${filebase64("web.sh")}"
  }
  SETTINGS
}
```

Web.sh

```
#!/bin/bash
```

```
sudo apt update
```

```
sudo apt-get -y install apache2
```

```
echo "<H1>Pagina creada mediante script</H1>" > /var/www/html/index.html
```

Terraform plan

A continuación, os pongo la salida de la ejecución del plan de despliegue:

```
+ create

Terraform will perform the following actions:

# azurerm_network_interface.nic will be created
+ resource "azurerm_network_interface" "nic" {
  + applied_dns_servers      = (known after apply)
  + dns_servers              = (known after apply)
  + enable_accelerated_networking = false
  + enable_ip_forwarding     = false
  + id                      = (known after apply)
  + internal_dns_name_label  = (known after apply)
  + internal_domain_name_suffix = (known after apply)
  + location                 = "eastus2"
  + mac_address              = (known after apply)
  + name                     = "NIC"
  + private_ip_address       = (known after apply)
  + private_ip_addresses     = (known after apply)
  + resource_group_name      = "0_RG"
  + virtual_machine_id       = (known after apply)

  + ip_configuration {
    + name                = "ipprivada"
    + primary              = (known after apply)
    + private_ip_address   = (known after apply)
    + private_ip_address_allocation = "dynamic"
    + private_ip_address_version  = "IPv4"
    + public_ip_address_id  = (known after apply)
  }
}
```

```

    + subnet_id = (known after apply)
  }
}

# azurerm_network_security_group.nsg will be created
+ resource "azurerm_network_security_group" "nsg" {
  + id = (known after apply)
  + location = "eastus2"
  + name = "NSG"
  + resource_group_name = "0_RG"
  + security_rule = [
    + {
      + access = "Allow"
      + description = ""
      + destination_address_prefix = "*"
      + destination_address_prefixes = []
      + destination_application_security_group_ids = []
      + destination_port_range = "22"
      + destination_port_ranges = []
      + direction = "Inbound"
      + name = "ssh"
      + priority = 110
      + protocol = "Tcp"
      + source_address_prefix = "*"
      + source_address_prefixes = []
      + source_application_security_group_ids = []
      + source_port_range = "*"
      + source_port_ranges = []
    },
    + {
      + access = "Allow"
      + description = ""
      + destination_address_prefix = "*"
      + destination_address_prefixes = []
      + destination_application_security_group_ids = []
      + destination_port_range = "80"
      + destination_port_ranges = []
      + direction = "Inbound"
      + name = "web"
      + priority = 120
      + protocol = "Tcp"
      + source_address_prefix = "*"
      + source_address_prefixes = []
      + source_application_security_group_ids = []
      + source_port_range = "*"
      + source_port_ranges = []
    },
  ]
}

# azurerm_public_ip.pip will be created
+ resource "azurerm_public_ip" "pip" {
  + allocation_method = "Dynamic"
  + fqdn = (known after apply)
  + id = (known after apply)
  + idle_timeout_in_minutes = 4
  + ip_address = (known after apply)
  + ip_version = "IPv4"
  + location = "eastus2"

```

```

+ name                = "pip"
+ resource_group_name = "0_RG"
+ sku                 = "Basic"
}

# azure_rm_resource_group.0 will be created
+ resource "azure_rm_resource_group" "0" {
+ id          = (known after apply)
+ location    = "eastus2"
+ name        = "0_RG"
}

# azure_rm_subnet.subnet will be created
+ resource "azure_rm_subnet" "subnet" {
+ address_prefix          = "10.0.1.0/24"
+ address_prefixes       = (known after apply)
+ enforce_private_link_endpoint_network_policies = false
+ enforce_private_link_service_network_policies = false
+ id                     = (known after apply)
+ name                   = "subnet"
+ resource_group_name    = "0_RG"
+ virtual_network_name   = "1_VNet"
}

# azure_rm_virtual_machine.VM will be created
+ resource "azure_rm_virtual_machine" "VM" {
+ availability_set_id      = (known after apply)
+ delete_data_disks_on_termination = false
+ delete_os_disk_on_termination = false
+ id                      = (known after apply)
+ license_type             = (known after apply)
+ location                 = "eastus2"
+ name                     = "VM"
+ network_interface_ids    = (known after apply)
+ resource_group_name      = "0_RG"
+ vm_size                  = "Standard_B1s"

+ boot_diagnostics {
+   enabled = false
}

+ identity {
+   identity_ids = (known after apply)
+   principal_id = (known after apply)
+   type         = (known after apply)
}

+ os_profile {
+   admin_password = (sensitive value)
+   admin_username = "usuario"
+   computer_name  = "VM"
+   custom_data    = (known after apply)
}

+ os_profile_linux_config {
+   disable_password_authentication = false
}

+ storage_data_disk {

```

```

+ caching = (known after apply)
+ create_option = (known after apply)
+ disk_size_gb = (known after apply)
+ lun = (known after apply)
+ managed_disk_id = (known after apply)
+ managed_disk_type = (known after apply)
+ name = (known after apply)
+ vhd_uri = (known after apply)
+ write_accelerator_enabled = (known after apply)
}

+ storage_image_reference {
+ offer = "UbuntuServer"
+ publisher = "Canonical"
+ sku = "18.04-LTS"
+ version = "latest"
}

+ storage_os_disk {
+ caching = "ReadWrite"
+ create_option = "FromImage"
+ disk_size_gb = (known after apply)
+ managed_disk_id = (known after apply)
+ managed_disk_type = "Standard_LRS"
+ name = "VM-OS"
+ os_type = (known after apply)
+ write_accelerator_enabled = false
}
}

# azurerm_virtual_machine_extension.installweb will be created
+ resource "azurerm_virtual_machine_extension" "installweb" {
+ id = (known after apply)
+ name = "installweb"
+ publisher = "Microsoft.Azure.Extensions"
+ settings = jsonencode(
+ {
+ + script =
"IyEvYmluL2Jhc2gNCnN1ZG8gYXB0IHVwZGF0ZQ0Kc3VkbyBhcHQQtZ2V0IC15IGluc3RhbgwGwYXBhY2h1Mg0KZWNoYAiPEgxpLBhZ2luYSBjcjcmVhZGEgbWVkaWFudGUgc2NyaXB0PC9IMT4iID4gL3Zhci93d3cvaHRtbC9pbmRleC5odG1sDQo="
+ }
+ )
+ type = "CustomScript"
+ type_handler_version = "2.0"
+ virtual_machine_id = (known after apply)
}

# azurerm_virtual_network.vnet will be created
+ resource "azurerm_virtual_network" "vnet" {
+ address_space = [
+ + "10.0.0.0/16",
+ ]
+ guid = (known after apply)
+ id = (known after apply)
+ location = "eastus2"
+ name = "1_VNet"
+ resource_group_name = "0_RG"
+ subnet = (known after apply)
}

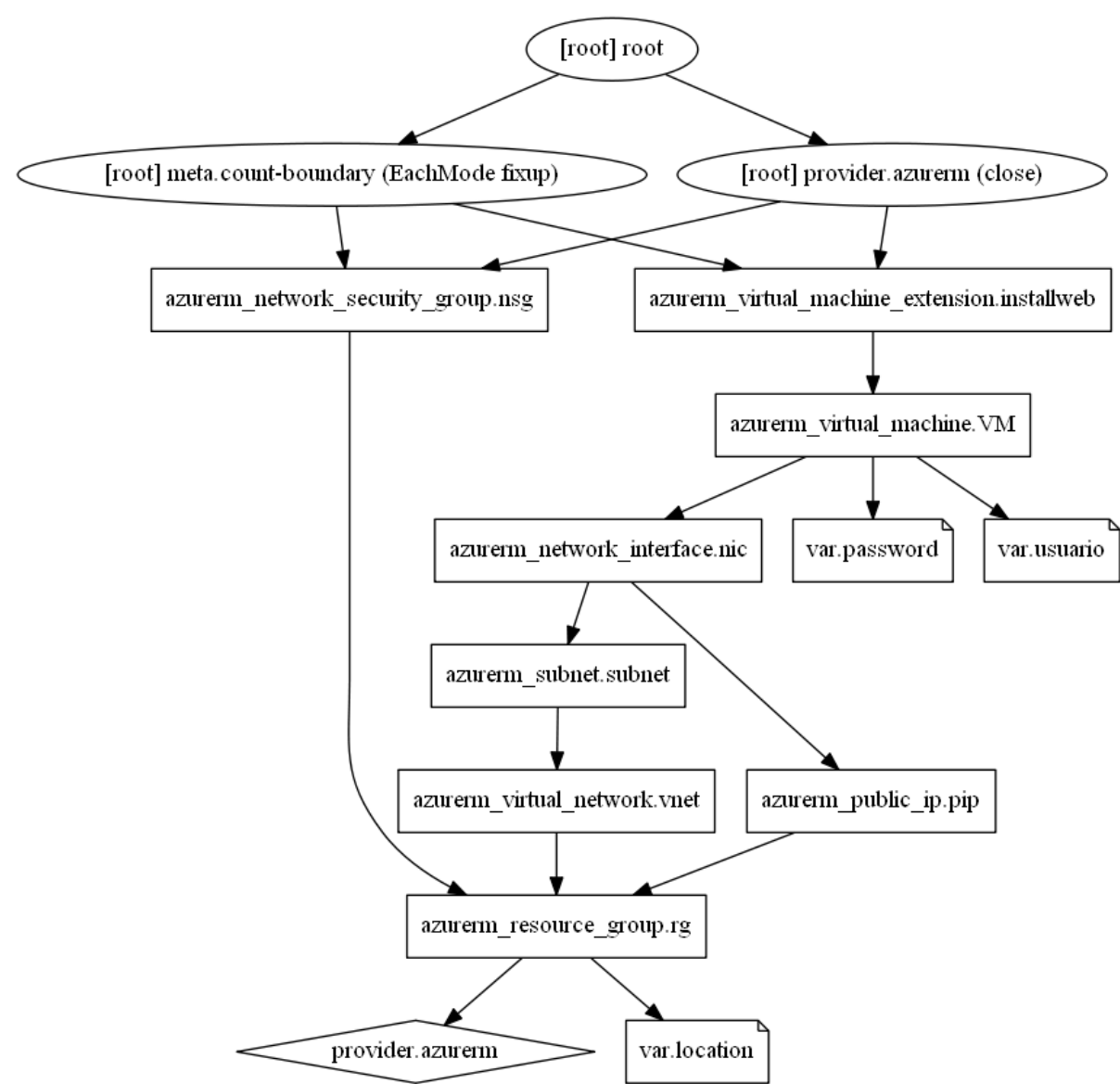
```


}

Terraform graph:

Después de terraform plan visualizamos el despliegue:

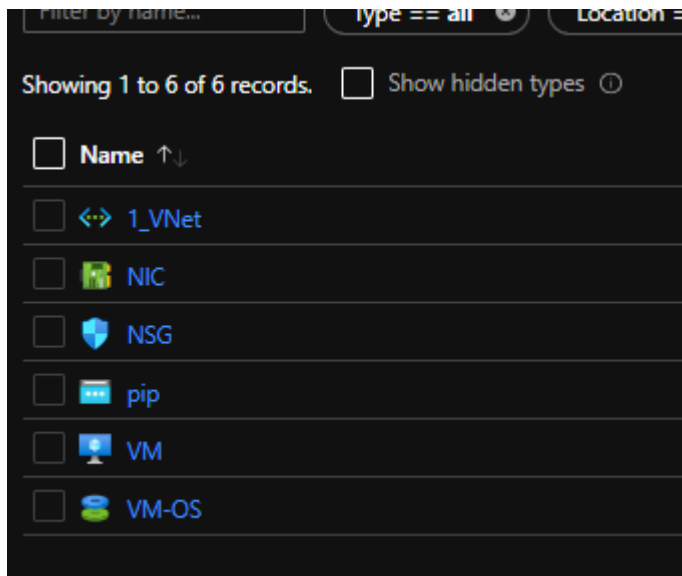
```
terraform graph -draw-cycles > despliegue_final.svg
```



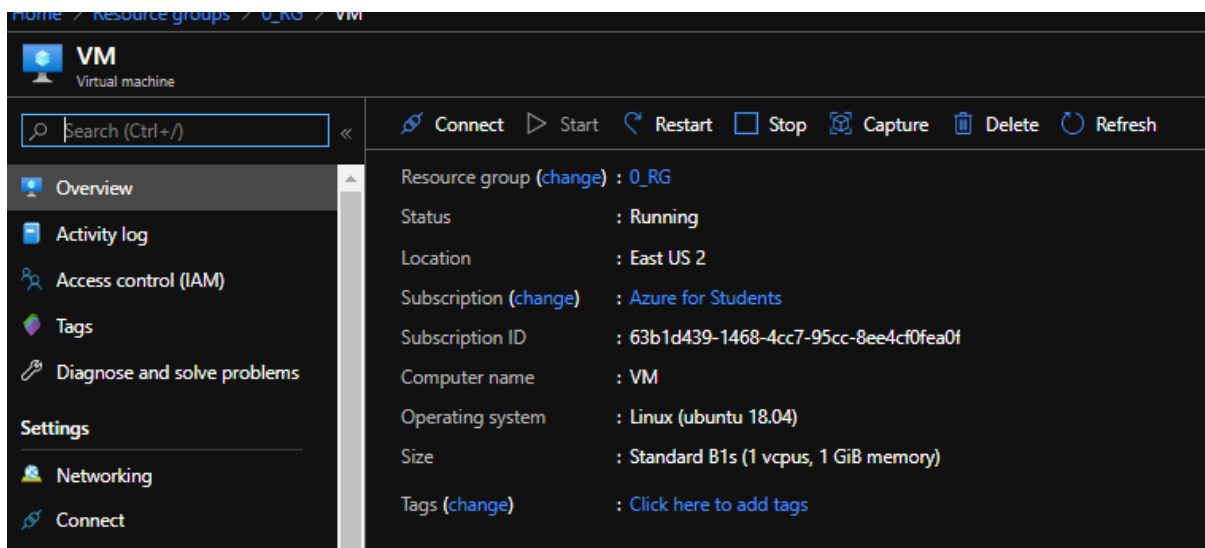
Resultado final

Actualmente, con los últimos cambios en Azure con las extensiones no podemos subir un script desde local.

```
ed\n      Recommends: ssl-cert but it is not installable\n\n[stderr]\n\nWARNING: apt\nscript.sh: line 4: /var/www/html/index.html: No such file or directory\n\nMore in\n\non scripts.tf line 1, in resource \"azurerm_virtual_machine_extension\" \"installweb\":\n  1: resource \"azurerm_virtual_machine_extension\" \"installweb\" {\n\nC:\\Users\\frs\\Desktop\\terraform\\4_Virtual_Machine>
```



Vemos como se ha creado todo el grupo de recursos y visualizamos la VM:



Azure Spot	: N/A
Public IP address	: 52.177.78.42
Private IP address	: 10.0.1.4
Public IP address (IPv6)	: -
Private IP address (IPv6)	: -
Virtual network/subnet	: 1_VNet/subnet
DNS name	: Configure

Nos conectamos mediante Putty a la máquina virtual:

```
See "man sudo_root" for details.

usuario@VM:~$ hostname
VM
usuario@VM:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.0.1.4  netmask 255.255.255.0  broadcast 10.0.1.255
    inet6 fe80::20d:3aff:fee6:96d5  prefixlen 64  scopeid 0x20<link>
    ether 00:0d:3a:e6:96:d5  txqueuelen 1000  (Ethernet)
    RX packets 24560  bytes 32634543 (32.6 MB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 5881  bytes 1677408 (1.6 MB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop txqueuelen 1000  (Local Loopback)
    RX packets 13852  bytes 1071680 (1.0 MB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 13852  bytes 1071680 (1.0 MB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

usuario@VM:~$
```

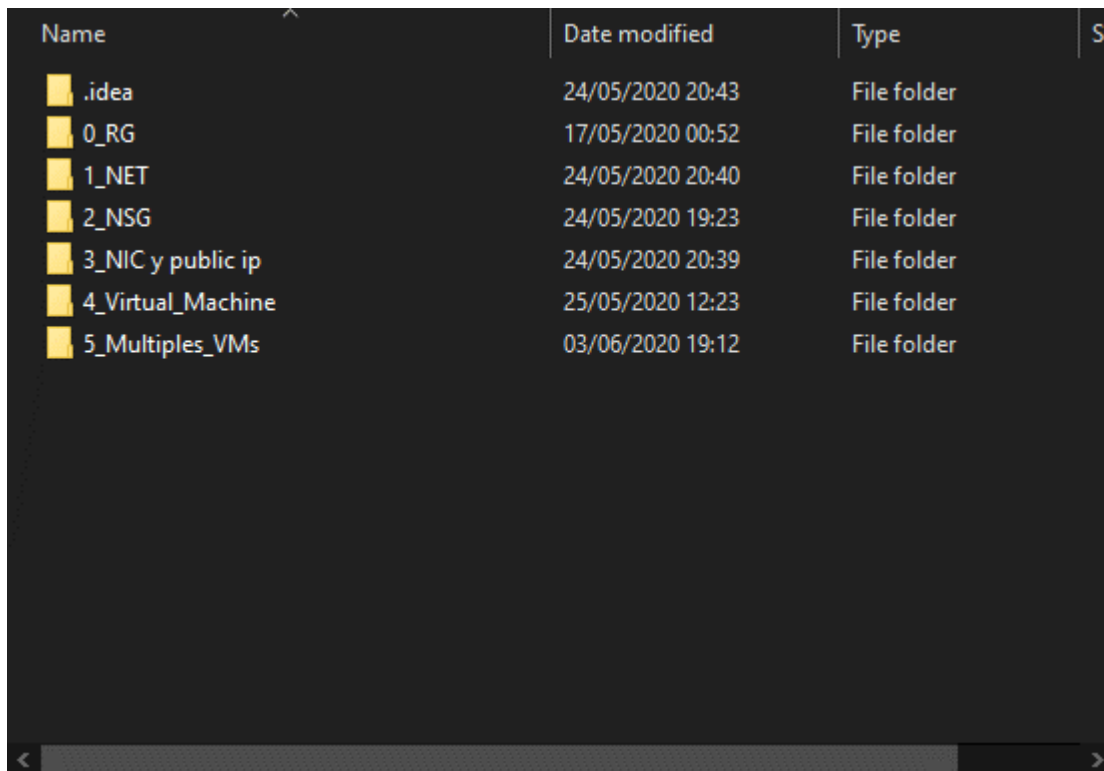
Comprobamos que tiene el nombre que hemos declarado, la ip interna y que podemos acceder gracias al NSG.

Continuamos.

5_Multiples_VMs: Creación de varias VMs mediante una declaración.

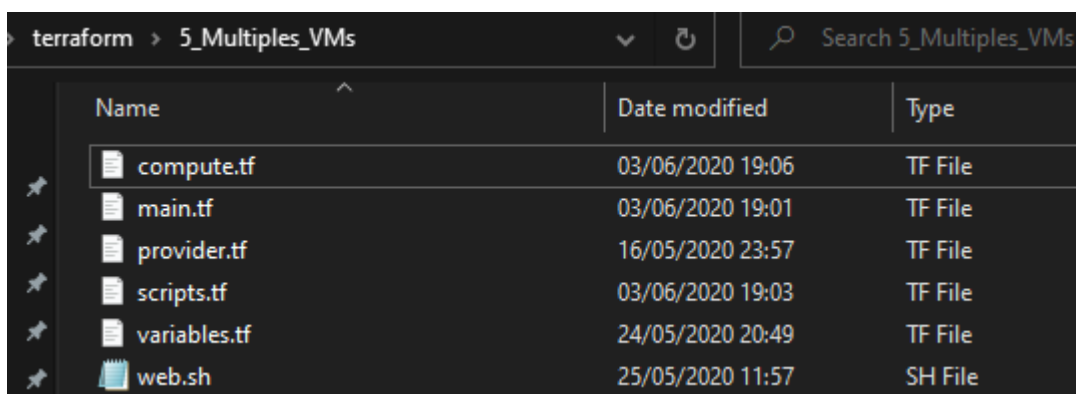
Hay veces que necesitamos crear muchas máquinas y no podemos escribir el código de todas las máquinas una a una, así que hoy voy a enseñaros una manera para crear varias máquinas mediante la declaración de una sola.

Partiendo de la base del ejercicio 4, vamos a copiar los archivos en una carpeta nueva que se llamará 5_Multiples_VMs:



Name	Date modified	Type	Size
.idea	24/05/2020 20:43	File folder	
0_RG	17/05/2020 00:52	File folder	
1_NET	24/05/2020 20:40	File folder	
2_NSX	24/05/2020 19:23	File folder	
3_NIC y public ip	24/05/2020 20:39	File folder	
4_Virtual_Machine	25/05/2020 12:23	File folder	
5_Multiples_VMs	03/06/2020 19:12	File folder	

Copiaremos los archivos `*.tf`, `*.sh` que están en la carpeta 4 y los pasaremos a la carpeta del ejercicio 5:



Name	Date modified	Type
compute.tf	03/06/2020 19:06	TF File
main.tf	03/06/2020 19:01	TF File
provider.tf	16/05/2020 23:57	TF File
scripts.tf	03/06/2020 19:03	TF File
variables.tf	24/05/2020 20:49	TF File
web.sh	25/05/2020 11:57	SH File

Vamos a modificar el archivo `main.tf` para crear 2 NICs y 2 public ips mediante una sola declaración resource.

Main.tf

Buscamos al final del archivo `main.tf` y encontraremos la declaración de `azurerm_network_interface`:

```
resource "azurerm_network_interface" "nic" {
  location = var.location
  name     = "NIC"
  resource_group_name = azurerm_resource_group.rg.name
  ip_configuration {
    name = "ipprivada"
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id = azurerm_public_ip.pip.id
    subnet_id = azurerm_subnet.subnet.id
  }
}
```

LOOP: count y count.index

Vamos a añadir una línea justo después de la primera llave y escribiremos:

```
Count = 2
```

Con esto, indicamos que cuente 2 (empezando desde 0 al 1). En realidad, estamos haciendo un loop que empieza por 0 y acaba en 1. Esto solo no sirvió para crear 2 NICs, hemos de modificar 2 apartados más:

Name:

En la línea que declaramos el nombre de la NIC, añadiremos un guion bajo o lo que queráis, y pondremos:

```
${count.index}
```

El resultado sería:

```
resource "azurerm_network_interface" "nic" {
  #indicamos que cuente 2 (empezando por el 0 y acabando en 1)
  count = 2
  location = var.location
  name     = "3_NIC_${count.index}"
  resource_group_name = azurerm_resource_group.rg.name
}
```

Seguimos modificando la NIC y dentro de `ip_configuration`, añadimos lo mismo en el nombre de la ip privada:

```

resource_group_name = azurerm_resource_group.rg.name
ip_configuration {
  name = "ipprivada_${count.index}"
  private_ip_address_allocation = "Dynamic"
}

```

Tenemos que modificar la línea de la `public_ip_address_id` para dejar correctamente la declaración:

```

private_ip_address_allocation = "Dynamic"
public_ip_address_id = azurerm_public_ip.pip.id
subnet_id = azurerm_subnet.subnet.id

```

Modificamos justo antes del id de pip y añadimos `[count.index]`

[count.index]

Con esto recorreremos el array del loop que hemos declarado con el count.

El resultado final sería:

```

private_ip_address_allocation = "Dynamic"
public_ip_address_id = azurerm_public_ip.pip[count.index].id
subnet_id = azurerm_subnet.subnet.id

```

Y la declaración de las NICs:

```

resource "azurerm_network_interface" "nic" {
  #indicamos que cuente 2 (empezando por el 0 y acabando en 1)
  count = 2
  location = var.location
  name = "3_NIC_${count.index}"
  resource_group_name = azurerm_resource_group.rg.name
  ip_configuration {
    name = "ipprivada_${count.index}"
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id = azurerm_public_ip.pip[count.index].id
    subnet_id = azurerm_subnet.subnet.id
  }
}

```

Tenemos nuestro primer apartado modificado, Modificaremos la IP Pública puesto que debemos tener 1 por cada NIC en estos ejercicios:

```
resource "azurerm_public_ip" "pip" {
  count = 2
  location = var.location
  name = "pip_${count.index}"
  resource_group_name = azurerm_resource_group.rg.name
  allocation_method = "Dynamic"
  sku = "Basic"
}
```

Esta vez solamente hemos añadido el loop y que modifique el nombre de la Public IP.

Actualmente tenemos modificado:

- 2 NICs-> 3_NIC_0 y 3_NIC_1
 - 2 IP Privadas-> ipprivada_0 y ipprivada_1
- 2 Public IP-> pip_0 y pip_1

Compute.tf

Partimos del archivo del ejercicio 4:

```
resource "azurerm_virtual_machine" "VM" {
  location              = var.location
  name                  = "VM"
  network_interface_ids = [azurerm_network_interface.nic.id]
  resource_group_name   = azurerm_resource_group.rg.name
  vm_size               = "Standard_B1s"

  storage_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "18.04-LTS"
    version   = "latest"
  }

  storage_os_disk {
    create_option = "FromImage"
    name          = "VM-OS"
    managed_disk_type = "Standard_LRS"
    caching       = "ReadWrite"
  }
}
```

Hemos de añadir un salto de carro después de la primera llave y añadir count=2:

```
resource "azurerm_virtual_machine" "VM" {
  count = 2
  location              = var.location
```

Ahora en el campo del nombre (Cuidado con los guiones bajos en los nombres de máquina) añadiremos `${count.index}` al final del nombre y sin espacios:

```
name = "4VM${count.index}"
```

Ahora es importante que en el campo de `network_interface_ids` reemplazemos:

```
network_interface_ids = [azurerm_network_interface.nic.id]
```

Por

```
network_interface_ids = ["${element(azurerm_network_interface.nic.*.id, count.index)}"]
```

Con esto conseguimos que coja los ids uno a uno para la asignación de las NICs a la hora de crear las VMs.

Quedan 2 campos más, el nombre del disco gestionado y el hostname de las VMs, añadimos:

```
storage_os_disk {  
  create_option = "FromImage"  
  name          = "VM-OS-${count.index}"  
  managed_disk_type = "Standard_LRS"  
  caching       = "ReadWrite"  
}
```

```
os_profile {  
  admin_username = var.usuario  
  admin_password = var.password  
  computer_name  = "4VM${count.index}"  
}  
boot_diagnostics {  
  enabled = false  
  storage_uri = ""  
}
```

Scripts.tf

Con esto ya estaríamos, pero nos falla un detalle, la extensión script debemos indicar el id de las VMs, por ello añadiremos:

Un loop con `count = 2` y `[count.index]` en `virtual_machine_id`


```

resource "azurerm_virtual_machine_extension" "installweb" {
  count = 2
  name = "installweb"
  publisher = "Microsoft.Azure.Extensions"
  type = "CustomScript"
  type_handler_version = "2.0"
  virtual_machine_id = azurerm_virtual_machine.VM[count.index].id

  settings = <<SETTINGS
  {
    "script": "${filebase64("web.sh")}"
  }
  SETTINGS
}

```

Ya podemos hacer un az login, terraform init, terraform plan.

Vemos que nos va a crear 2 NICs, 2 PIPs y 2 VMs:

Terraform will perform the following actions:

```

# azurerm_network_interface.nic[0] will be created
+ resource "azurerm_network_interface" "nic" {

```

```

# azurerm_network_interface.nic[1] will be created
+ resource "azurerm_network_interface" "nic" {
  + applied_dns_servers = (known after apply)

```

```

# azurerm_public_ip.pip[0] will be created
+ resource "azurerm_public_ip" "pip" {

```

```

# azurerm_public_ip.pip[1] will be created
+ resource "azurerm_public_ip" "pip" {

```

```

# azurerm_virtual_machine.VM[0] will be created
+ resource "azurerm_virtual_machine" "VM" {

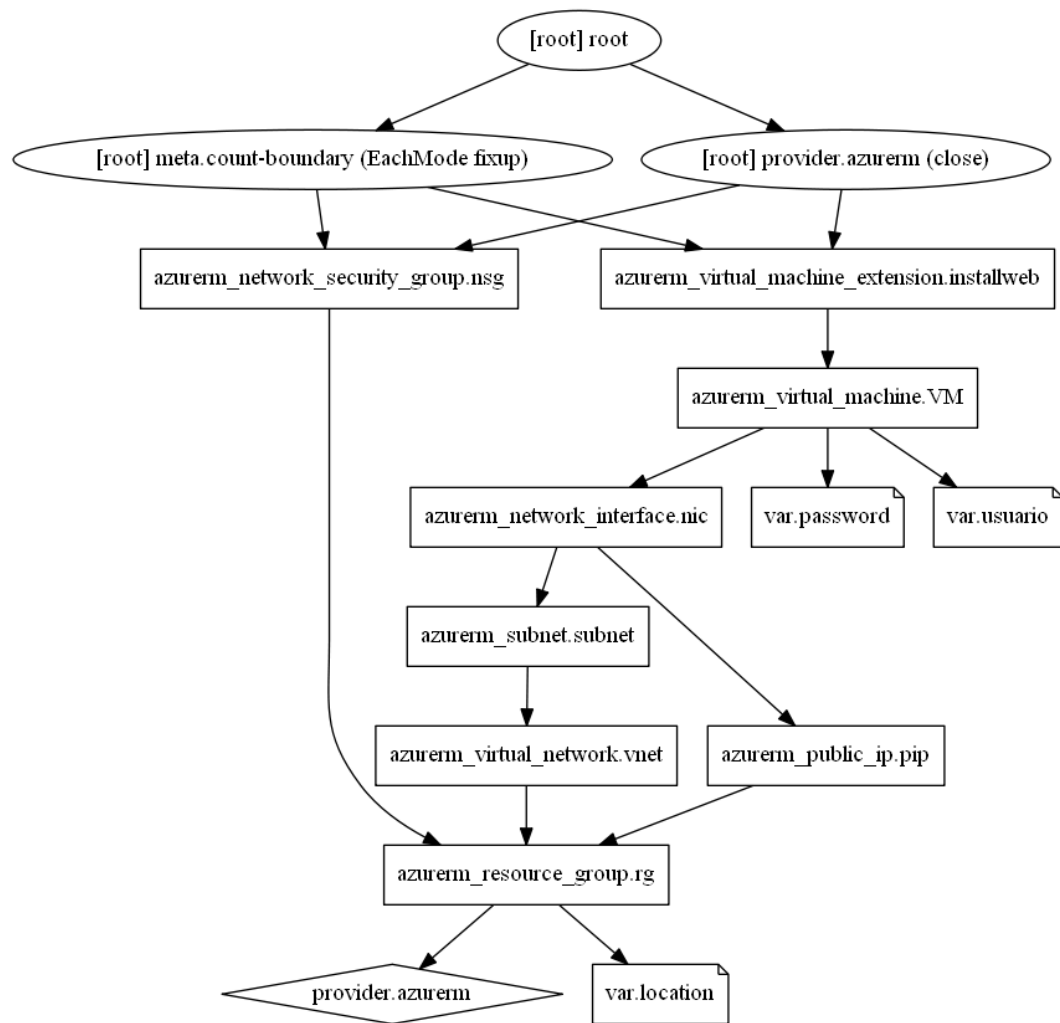
```

```

# azurerm_virtual_machine.VM[1] will be created
+ resource "azurerm_virtual_machine" "VM" {

```

Si realizamos un terraform graph:



Vemos que solamente hemos declarado 1 NIC, 1 PIP y 1 VM.

Realizamos un terraform apply y aceptamos:

```

azurerm_public_ip.pip[0]: Creating...
azurerm_public_ip.pip[1]: Creating...
azurerm_network_security_group.nsg: Creating...
azurerm_public_ip.pip[0]: Creation complete after 4s
azurerm_public_ip.pip[1]: Creation complete after 5s
azurerm_network_security_group.nsg: Creation complete
azurerm_virtual_network.vnet: Still creating... [10s
azurerm_virtual_network.vnet: Creation complete after
azurerm_subnet.subnet: Creating...
azurerm_subnet.subnet: Creation complete after 2s [i
azurerm_network_interface.nic[0]: Creating...
azurerm_network_interface.nic[1]: Creating...
azurerm_network_interface.nic[0]: Creation complete
azurerm_network_interface.nic[1]: Creation complete
azurerm_virtual_machine.VM[1]: Creating...
azurerm_virtual_machine.VM[0]: Creating...
□

```

Vemos como se van creando múltiples recursos habiendo declarado 1 por cada.

```

azurerm_virtual_machine.VM[1]: Creating...
azurerm_virtual_machine.VM[0]: Creating...
azurerm_virtual_machine.VM[1]: Still creating... [10s elapsed]
azurerm_virtual_machine.VM[0]: Still creating... [10s elapsed]
azurerm_virtual_machine.VM[1]: Still creating... [20s elapsed]
azurerm_virtual_machine.VM[0]: Still creating... [20s elapsed]
azurerm_virtual_machine.VM[0]: Still creating... [30s elapsed]
azurerm_virtual_machine.VM[1]: Still creating... [30s elapsed]
azurerm_virtual_machine.VM[1]: Creation complete after 31s [id=/subscriptions/
azurerm_virtual_machine.VM[0]: Still creating... [40s elapsed]
azurerm_virtual_machine.VM[0]: Still creating... [50s elapsed]
azurerm_virtual_machine.VM[0]: Still creating... [1m0s elapsed]
azurerm_virtual_machine.VM[0]: Still creating... [1m10s elapsed]
azurerm_virtual_machine.VM[0]: Still creating... [1m20s elapsed]
azurerm_virtual_machine.VM[0]: Still creating... [1m30s elapsed]
azurerm_virtual_machine.VM[0]: Still creating... [1m40s elapsed]
azurerm_virtual_machine.VM[0]: Creation complete after 1m41s [id=/subscriptions/
azurerm_virtual_machine_extension.installweb[1]: Creating...
azurerm_virtual_machine_extension.installweb[0]: Creating...
□

```

```

azurerm_virtual_machine_extension.installweb[1]: Still creating... [3m0s elapsed]
azurerm_virtual_machine_extension.installweb[0]: Still creating... [3m0s elapsed]
azurerm_virtual_machine_extension.installweb[1]: Still creating... [3m10s elapsed]
azurerm_virtual_machine_extension.installweb[0]: Still creating... [3m10s elapsed]
azurerm_virtual_machine_extension.installweb[1]: Creation complete after 3m12s [id=/subscriptions/63b1
azurerm_virtual_machine_extension.installweb[0]: Creation complete after 3m13s [id=/subscriptions/63b1

Apply complete! Resources: 12 added, 0 changed, 0 destroyed.

```

Accedemos a Azure y comprobamos el resultado:

Showing 1 to 10 of 10 records. ☐ Show hidden types ⓘ

<input type="checkbox"/> Name ↑↓	Type ↑↓
<input type="checkbox"/> <-> 1_VNet	Virtual network
<input type="checkbox"/> 2_NSG	Network security group
<input type="checkbox"/> 3_NIC_0	Network interface
<input type="checkbox"/> 3_NIC_1	Network interface
<input type="checkbox"/> 4VM0	Virtual machine
<input type="checkbox"/> 4VM1	Virtual machine
<input type="checkbox"/> pip_0	Public IP address
<input type="checkbox"/> pip_1	Public IP address
<input type="checkbox"/> VM-OS-0	Disk
<input type="checkbox"/> VM-OS-1	Disk

6_Metadatos

Queremos modificar la página de inicio para que indique diferentes valores sobre la VM.

Accediendo a la ip pública de las VMs comprobamos que tenemos un texto genérico el cual indica que se ha realizado mediante un script.

Accedemos a una de las VMs mediante ssh y comprobamos el hostname y sacamos las ips:

```

@4VM0:~$ hostname
4VM0
@4VM0:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.1.4 netmask 255.255.255.0 broadcast 10.0.1.255
    inet6 fe80::20d:3aff:fe0f:582a prefixlen 64 scopeid 0x20<link>
    ether 00:0d:3a:0f:58:2a txqueuelen 1000 (Ethernet)
    RX packets 41513 bytes 52276104 (52.2 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 13420 bytes 4495329 (4.4 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 2644 bytes 310938 (310.9 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2644 bytes 310938 (310.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

@4VM0:~$

```

Podemos comprobar que nos muestra correctamente el hostname pero no nos muestra la ip pública.

Para visualizar todos los metadatos de una VM en azure hemos de introducir el comando siguiente:

```
curl -H Metadata:true http://169.254.169.254/metadata/instance?api-version=2017-08-01
```

```

@4VM0:~$ curl -H Metadata:true "http://169.254.169.254/metadata/instance?api-version=2017-08-01"
{"compute":{"location":"eastus2","name":"4VM0","offer":"UbuntuServer","osType":"Linux","placementGroupId":"","platformFaultDomain":"0","platformUpdateDomain":"0","publisher":"Canonical","resourceGroupName":"0 RG","sku":"18.04-LTS","subscriptionId":"","tags":"","version":"18.04.202005220","vmId":"f22e5b86-519e-4494-8958-0b7668ddcf01","vmSize":"Standard_B1s"},"network":{"interface":[{"ipv4":{"ipAddress":[{"privateIpAddress":"10.0.1.4","publicIpAddress":"52.254.82.82"}],"subnet":[{"address":"10.0.1.0","prefix":"24"}],"ipv6":{"ipAddress":[]},"macAddress":"000D3A0F582A"}]}}
@4VM0:~$

```

Vamos a personalizar la página de inicio del script dentro de la carpeta 5_Multiples_VMs. Vamos a indicar el hostname y la ip publica a través de los metadatos:

```

#!/bin/bash
apt update && apt -y upgrade
apt-get -y install apache2
export HOSTNAME=$(curl -H Metadata:true "http://169.254.169.254/metadata/instance?api-version=2017-08-01" | jq -r '.compute.tags["hostname"]')
export PUBLIC_IPV4=$(curl -H Metadata:true "http://169.254.169.254/metadata/instance?api-version=2017-08-01" | jq -r '.compute.network.interface[0].ipv4.ipAddress[0].publicIpAddress')
echo "<p><center><H1>Hostname: $HOSTNAME,</p><p><center><H1>Public IP: $PUBLIC_IPV4,</p>"

```

```
p://169.254.169.254/metadata/instance/compute/name?api-version=2017-08-01&for
http://169.254.169.254/metadata/instance/network/interface/0/ipv4/ipAddress/0
><p>IP Address: $PUBLIC_IPV4</H1></center></p>" > /var/www/html/index.html
```

Os pongo el código:

```
export HOSTNAME=$(curl -H Metadata:true
"http://169.254.169.254/metadata/instance/compute/name?api-version=2017-08-
01&format=text")
export PUBLIC_IPV4=$(curl -H Metadata:true
"http://169.254.169.254/metadata/instance/network/interface/0/ipv4/ipAddress/0/pub
licIpAddress?api-version=2017-08-01&format=text")
echo "<p><center><H1>Hostname: $HOSTNAME,</p><p>IP Address:
$PUBLIC_IPV4</H1></center></p>" > /var/www/html/index.html
```

Lanzamos la creación de las máquinas con terraform apply:

```
~ {
  ~ script = "IyEvYm1uL2Jhc2gNCmFwdCB1cGRh
XB0IC15IHVwZ3JhZGUNCmFwdC1nZXQgLXkgaW5zdGFsbCBhcGFjaGU
VybCAtSCBNZXRhZGF0YTp0cnVlICJodHRwOi8vMTY5LjI1NC4xNjku
8cD5JUCBBZGRyZXNzOiAkUFVCTElDX0lQVjQ8L0gxPjwvY2VudGVyF
}
)
tags              = {}
type              = "CustomScript"
type_handler_version = "2.0"
virtual_machine_id = "/subscriptions/6

}

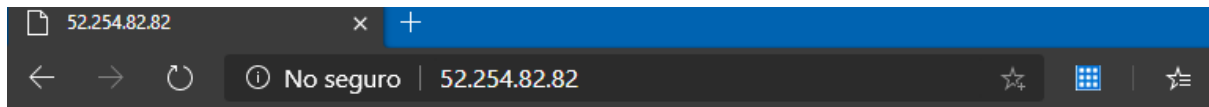
Plan: 0 to add, 2 to change, 0 to destroy.
```

```
azurerm_virtual_machine_extension.installweb[1]: Still modifying... [id=/subscriptions/63
azurerm_virtual_machine_extension.installweb[0]: Still modifying... [id=/subscriptions/63
azurerm_virtual_machine_extension.installweb[1]: Still modifying... [id=/subscriptions/63
azurerm_virtual_machine_extension.installweb[0]: Still modifying... [id=/subscriptions/63
azurerm_virtual_machine_extension.installweb[1]: Still modifying... [id=/subscriptions/63
azurerm_virtual_machine_extension.installweb[0]: Still modifying... [id=/subscriptions/63
azurerm_virtual_machine_extension.installweb[0]: Modifications complete after 1m2s [id=/s
azurerm_virtual_machine_extension.installweb[1]: Modifications complete after 1m3s [id=/s

Apply complete! Resources: 0 added, 2 changed, 0 destroyed.

C:\Users\frs\Desktop\terraform\5_Multiples_VMs>
```

Accedemos a la ip pública de una de las 2 máquinas:



Hostname: 4VM0,

IP Address: 52.254.82.82

Comprobamos que nos facilita el hostname y la ip pública. Si perdemos la ip al apagar la máquina, al reiniciar el servidor aparecerá reflejada la nueva ip.

Faltaría modificar el servidor apache para que haga un intervalo refresque el index.html con los cambios nuevos.

Os dejo la url de Microsoft con toda la información:

<https://docs.microsoft.com/en-us/azure/virtual-machines/windows/instance-metadata-service>

7_SSH: Como proteger nuestras VMs

Dentro de un entorno de producción, es indispensable que se creen recursos con protección por defecto. Una de las soluciones simples y fáciles de implementar seguridad en la creación de una máquina virtual, es una clave asimétrica. Esto evitará utilizar usuario y contraseña en la creación e inicia de sesión con terminal. Vamos a generar una clave asimétrica en Windows y luego modificaremos nuestro despliegue.

Creación de una clave asimétrica mediante Powershell:

Openssh utils

Abrimos un terminal de Powershell como administrador y ejecutamos:

```
Get-WindowsCapability -Online | ? Name -like 'OpenSSH*'
```

```
PS C:\Users\frs\Desktop\clase\terraform> Get-WindowsCapability -Online | ? Name -like 'OpenSSH*'

Name : OpenSSH.Client~~~~0.0.1.0
State : Installed

Name : OpenSSH.Server~~~~0.0.1.0
State : Installed

PS C:\Users\frs\Desktop\clase\terraform>
```

En vuestro caso, os saldrá: State : Not present

Instalamos el cliente y servidor de Openssh:

```
Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0
```

```
Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0
```

Iniciamos el servicio de ssh:

```
Start-Service sshd
```

Si queremos que se inicie al arrancar el sistema escribimos:

```
Set-Service -Name sshd -StartupType 'Automatic'
```

```
PS C:\Users\frs\Desktop\clase\terraform> Start-Service sshd
PS C:\Users\frs\Desktop\clase\terraform> Set-Service -Name sshd -StartupType 'Automatic'
PS C:\Users\frs\Desktop\clase\terraform>
```

Confirmamos que con la instalación se ha creado una regla para el servicio SSH en el firewall

```
Get-NetFirewallRule -Name *ssh*
```

```
PS C:\Users\frs\Desktop\clase\terraform> Get-NetFirewallRule -Name *ssh*

Name                : OpenSSH-Server-In-TCP
DisplayName          : OpenSSH SSH Server (sshd)
Description          : Inbound rule for OpenSSH SSH Server (sshd)
DisplayGroup         : OpenSSH Server
Group                : OpenSSH Server
Enabled              : True
Profile              : Any
Platform             : {}
Direction            : Inbound
Action               : Allow
EdgeTraversalPolicy   : Block
LooseSourceMapping    : False
LocalOnlyMapping      : False
Owner                :
PrimaryStatus         : OK
Status               : The rule was parsed successfully from the store. (65536)
EnforcementStatus     : NotApplicable
PolicyStoreSource     : PersistentStore
PolicyStoreSourceType : Local

PS C:\Users\frs\Desktop\clase\terraform>
```


Creamos una regla en el firewall de Windows:

```
New-NetFirewallRule -Name sshd -DisplayName 'OpenSSH Server (sshd)' -Enabled True -Direction Inbound -Protocol TCP -Action Allow -LocalPort 22
```

```
PS C:\Users\frs\Desktop\clase\terraform> New-NetFirewallRule -Name sshd -DisplayName 'OpenSSH Server (sshd)' -Enabled True -Direction Inbound -Protocol TCP -Action Allow -LocalPort 22

Name                : sshd
DisplayName          : OpenSSH Server (sshd)
Description         :
DisplayGroup        :
Group               :
Enabled             : True
Profile             : Any
Platform            : {}
Direction           : Inbound
Action              : Allow
EdgeTraversalPolicy : Block
LooseSourceMapping  : False
LocalOnlyMapping    : False
Owner               :
PrimaryStatus       : OK
Status              : The rule was parsed successfully from the store. (65536)
EnforcementStatus   : NotApplicable
PolicyStoreSource   : PersistentStore
PolicyStoreSourceType : Local

PS C:\Users\frs\Desktop\clase\terraform>
```

Con esto, ya podríamos acceder a cualquier máquina de Linux mediante ssh.

Conecto con una máquina de pruebas para comprobarlo:

```
ssh username@servername
```

```
PS C:\Users\frs\Desktop\clase\terraform> ssh usuario@40.115.26.249
The authenticity of host '40.115.26.249 (40.115.26.249)' can't be established.
ECDSA key fingerprint is SHA256:MYBsUmZUq+Xpy6cAumc1FqEXeRYPnSOdFz1H8rKgqkA.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '40.115.26.249' (ECDSA) to the list of known hosts.
usuario@40.115.26.249's password:
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.3.0-1028-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Mon Jun 15 18:42:50 UTC 2020

System load:  0.0               Processes:    109
Usage of /:   4.4% of 28.90GB   Users logged in:  0
Memory usage: 31%              IP address for eth0: 10.0.0.4
Swap usage:   0%

0 packages can be updated.
0 updates are security updates.

Last login: Mon Jun 15 16:46:21 2020 from 79.156.252.168
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

usuario@vm-ubuntu:~$
```

Creación de nuestra clave de usuario.

Con la instalación de del cliente/servidor ssh se ha creado una carpeta oculta en nuestra carpeta personal:

```
cd ~
```

Listamos el contenido de la carpeta personal usando el filtro .ssh:

```
PS C:\Users\frs> ls -Filter .ssh

Directory: C:\Users\frs

Mode                LastWriteTime         Length Name
----                -
d-----          08/06/2020   23:44             .ssh

PS C:\Users\frs>
```

Ahora podemos generar automáticamente una clave o par de claves con:

```
ssh-keygen
```

```
PS C:\Users\frs> ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\frs/.ssh/id_rsa):
C:\Users\frs/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\frs/.ssh/id_rsa.
Your public key has been saved in C:\Users\frs/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:fIFCEmUNnx+gyw09eFhsVI05U47o1Amihfm49V+1BD4 frs@frs-pc
The key's randomart image is:
+----[RSA 2048]-----+
| o oo**+ |
| o o .B+oo= |
| = ..=BB+ o |
| o o oBE=o..o |
| o +o *+Soo |
| . o. .o. |
| . . |
| . |
+-----[SHA256]-----+
PS C:\Users\frs>
```

Si no indicamos una contraseña, no nos pedirá la contraseña cada vez que iniciemos sesión mediante ssh.

Listamos el contenido de la carpeta para visualizar la clave:

```
PS C:\Users\frs> ls ~/.ssh\ -Recurse

Directory: C:\Users\frs\.ssh

Mode                LastWriteTime         Length Name
----                -
-a-----         28/03/2020    20:17         3247 
-a-----         28/03/2020    20:17          737 
-a-----         15/06/2020    21:03        1766 id_rsa
-a-----         15/06/2020    21:03         393 id_rsa.pub
-a-----         15/06/2020    20:42        3744 known_hosts

PS C:\Users\frs>
```

Opcional.

Si queremos asignar automáticamente un par de claves al usuario cuando inicie sesión, debemos ejecutar los siguientes comandos:

Como administrador:

```
start-Service ssh-agent
```

```
Get-Service ssh-agent
```

```
ssh-add ~/.ssh\<CLAVE_PRIVADA_QUE_QUEREMOS_USAR>
```

Permisos para no hacer accesible nuestras claves.

Uno de los errores más comunes al conectarnos en sistemas GNU/Linux mediante par de claves, es dejar los permisos demasiado accesibles a las claves:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@          WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions for ' ' are too open.
```

Comprobamos los permisos antes:

```
icacls <Nombre_del_archivo>
```

```
PS C:\Users\frs\.ssh> icacls .\id_rsa
.\id_rsa NT AUTHORITY\SYSTEM:(I)(F)
        BUILTIN\Administrators:(I)(F)
        FRS-PC\frs:(I)(F)
        S-1-5-21-1266629302-1647739438-3872959423-1001:(I)(F)

Successfully processed 1 files; Failed processing 0 files
PS C:\Users\frs\.ssh> █
```

Quitamos la itinerancia:

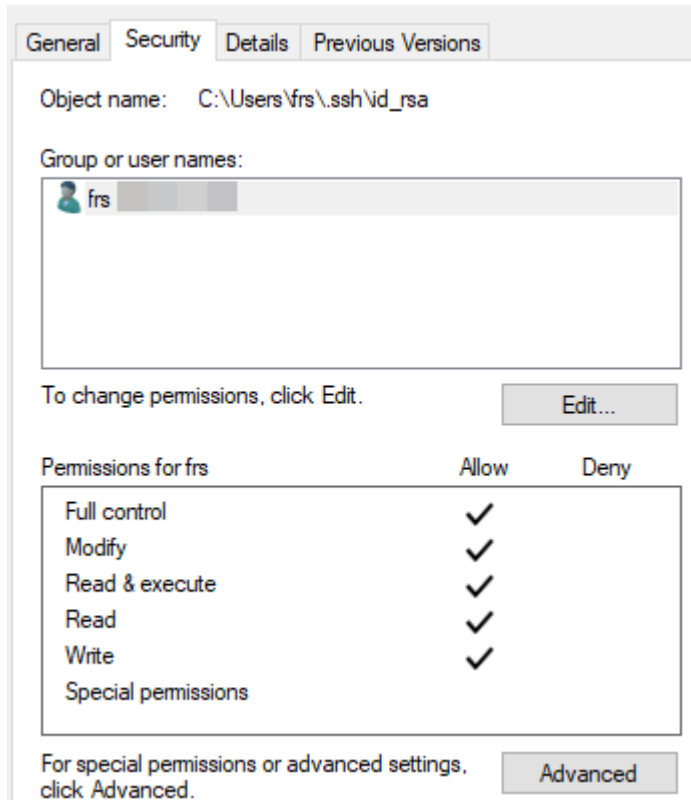
```
icacls <Nombre_del_archivo> /inheritance:r
```

```
PS C:\Users\frs\.ssh> icacls .\id_rsa /inheritance:r
processed file: .\id_rsa
Successfully processed 1 files; Failed processing 0 files
PS C:\Users\frs\.ssh> █
```

Garantizamos al usuario como Owner:

```
PS C:\Users\frs\.ssh> icacls .\id_rsa /grant frs:F
processed file: .\id_rsa
Successfully processed 1 files; Failed processing 0 files
PS C:\Users\frs\.ssh> █
```

Con estos comandos conseguimos que, solamente el usuario actual pueda leer las claves.



Si tenéis el subsistema de Linux (WSL) instalado en Windows solamente debéis de utilizar: `chmod 400`

Os dejo un enlace para que podáis exportar de RSA a PEM:

https://www.example-code.com/powershell/rsa_generatePEM.asp

Añadido:

Convertir claves ssh, a claves ssh2:

```
ssh-keygen -e -f c:\users\frs\.ssh\id_rsa > c:\users\frs\.ssh\id_rsa_ssh2.pub
```

REFERENCIAS:

(Microsoft, s.f.)

(Superuser.com, s.f.)

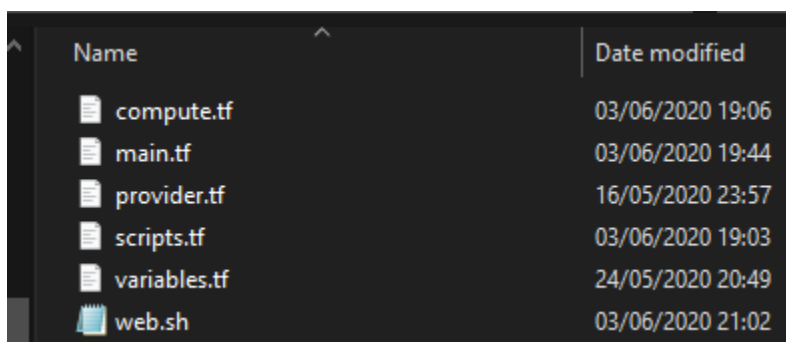
[Administración de claves de OpenSSH](#)

[Instalación de OpenSSH para Windows Server 2019 y Windows 10](#)

[Windows SSH: Permissions are too open](#)

Modificación de atributos en la declaración de la máquina virtual:

Como hago en este tutorial, copiamos el contenido del anterior ejercicio en la nueva carpeta: 7_SSH



Name	Date modified
compute.tf	03/06/2020 19:06
main.tf	03/06/2020 19:44
provider.tf	16/05/2020 23:57
scripts.tf	03/06/2020 19:03
variables.tf	24/05/2020 20:49
web.sh	03/06/2020 21:02

Abrimos la carpeta con Visual Code u otro. Abrimos el archivo **compute.tf**

```

os_profile_linux_config {
  disable_password_authentication = false
}
os_profile {
  admin_username = var.usuario
  admin_password = var.password
  computer_name  = "4VM${count.index}"
}
boot_diagnostics {
  enabled      = false
  storage_uri = ""
}
}

```

Modificamos los atributos siguientes:

```

os_profile_linux_config {
  disable_password_authentication = false
}

```

por:

```

os_profile_linux_config {
  disable_password_authentication = true
}

```

Agregamos `ssh_keys {}` en la línea siguiente:

```

os_profile_linux_config {
  disable_password_authentication = true
  ssh_keys {
    key_data = 
    path = 
  }
}

```

Explico los atributos `key_data` y `path`:

Key_data:

El contenido de la clave pública. He puesto el contenido de la clave publica en una variable. → `var.ssh`

En la declaración de la variable, ponemos el contenido de la clave publica abriéndolo con Notepad: `ssh-rsa ... usuario@equipousuario`

Path

La ruta destino de las claves autorizadas →

`/home/${var.usuario}/.ssh/authorized_keys`

```

os_profile_linux_config {
  disable_password_authentication = true
  ssh_keys {
    key_data = var.ssh
    path = "/home/${var.usuario}/.ssh/authorized_keys"
  }
}

```

```

variable "ssh" {
  type = string
  default = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQD
}

```

Por último, eliminamos la línea que incluye admin_password, dejando solamente el usuario:

```

os_profile {
  admin_username = var.usuario
  computer_name = "4VM${count.index}"
}

```

Toca realizar el despliegue.

Terraform init para descargar los archivos necesarios del provider.

Terraform plan para validar y preparar el despliegue.

Terraform apply para aplicar el despliegue:

```

Apply complete! Resources: 12 added, 0 changed, 0 destroyed.

```

Abrimos Powershell y conectamos mediante ssh a una de las máquinas creadas:

```

PS C:\Users\frs\.ssh> ssh usuario@52.254.0.128
The authenticity of host '52.254.0.128 (52.254.0.128)' can't be established.
ECDSA key fingerprint is SHA256:OorEoV5K8hw4X9ivyjIjiCNYExkzhWRyQ8bhw5PQWrg.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.254.0.128' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.3.0-1028-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Tue Jun 16 15:07:03 UTC 2020

System load:  0.32               Processes:    112
Usage of /:   4.9% of 28.90GB    Users logged in:  0
Memory usage: 34%              IP address for eth0: 10.0.1.4
Swap usage:   0%

0 packages can be updated.
0 updates are security updates.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

usuario@4VM0:~$

```

Vemos que no nos pide contraseña para loguearnos.

Fin del ejercicio 7.

Referencias

Microsoft. (s.f.). *Administración de claves de OpenSSH*. Obtenido de
 Administración de claves de OpenSSH: https://docs.microsoft.com/es-es/windows-server/administration/openssh/openssh_keymanagement

Superuser.com. (s.f.). Obtenido de
<https://superuser.com/questions/1296024/windows-ssh-permissions-for-private-key-are-too-open>

Terraform. (s.f.). *www.terraform.io*. Obtenido de [www.terraform.io](https://www.terraform.io/intro/index.html):
<https://www.terraform.io/intro/index.html>