Laboratorio

PRACTICA

TERRAFORM

CONTROL DE VERSIONES

Elaborado por:	Jonatan Stiven Gutierrez	**No. de Versión:**	1.0.0
Revisado por:		**Fecha de revisión:**	
Aprobado por:		**Fecha de Aprobación:**	

Historia de Modificaciones

No. de Versión	Fecha de Versión	Autor	Revisado por	Aprobado por	Descripción
1.0.0	21/02/2024	Jonatan Stiven Gutierrez			Documento Original

Lista de distribución

Para	Acción*	Empresa	Firma/Medio de Entrega

** Tipos de acción: Aprobar, Revisar, Informar, Archivar, Complementar, Asistir a junta, Otras (por favor especificar)*

Contenido

INTRODUCCION 4
PRERREQUISITOS 4
EJERCICIO 5:..... 5

INTRODUCCION

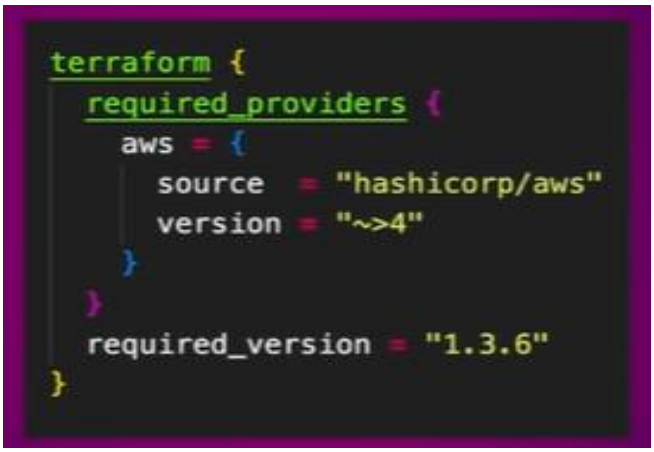
El siguiente documento proporciona una introducción detallada a los ejercicios realizados en el laboratorio.

PRERREQUISITOS

- Tener entendimiento de los temas vistos anteriormente.

EJERCICIO 5:

Manejo de versiones de terraform o de los providers.



```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~>4"
    }
  }
  required_version = "1.3.6"
}
```

Terraform constraints

Required Version	Meaning	Considerations
`0.15.0`	Only Terraform v0.15.0 exactly	To upgrade Terraform, first edit the `required_version` setting
`>= 0.15`	Any Terraform v0.15.0 or greater	Includes Terraform v1.0.0 and above
`~> 0.15.0`	Any Terraform v0.15.x, but not v1.0 or later	Minor version updates are intended to be non-disruptive
`>= 0.15, < 2.0.0`	Terraform v0.15.0 or greater, but less than v2.0.0	Avoids major version updates

Nota: Lo recomendado por Terraform es usar la opción 3: "**~>**".

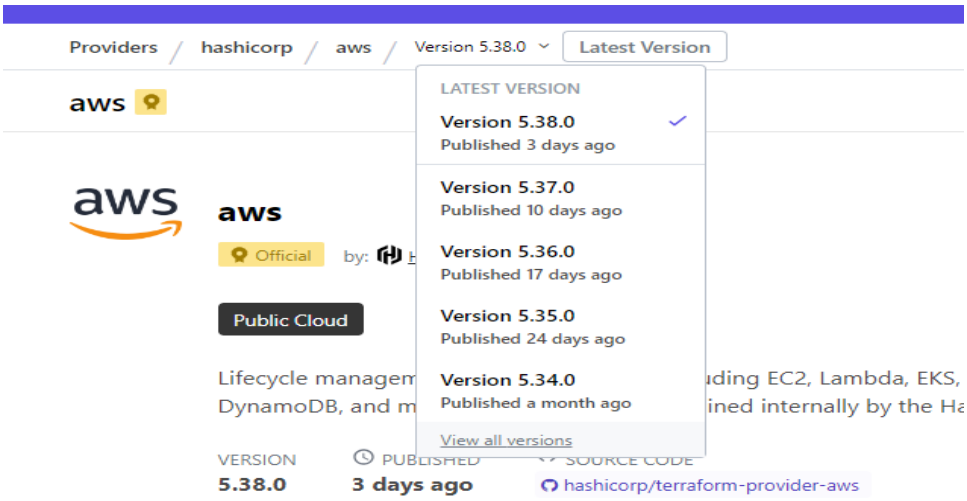
En este ejercicio vamos a trabajar las VPC.

VPC es un servicio de redes virtuales que te permite crear y administrar redes privadas en la nube de AWS. Puedes utilizar Terraform para definir VPCs, subredes, tablas de ruteo, grupos de seguridad y otros componentes de red.

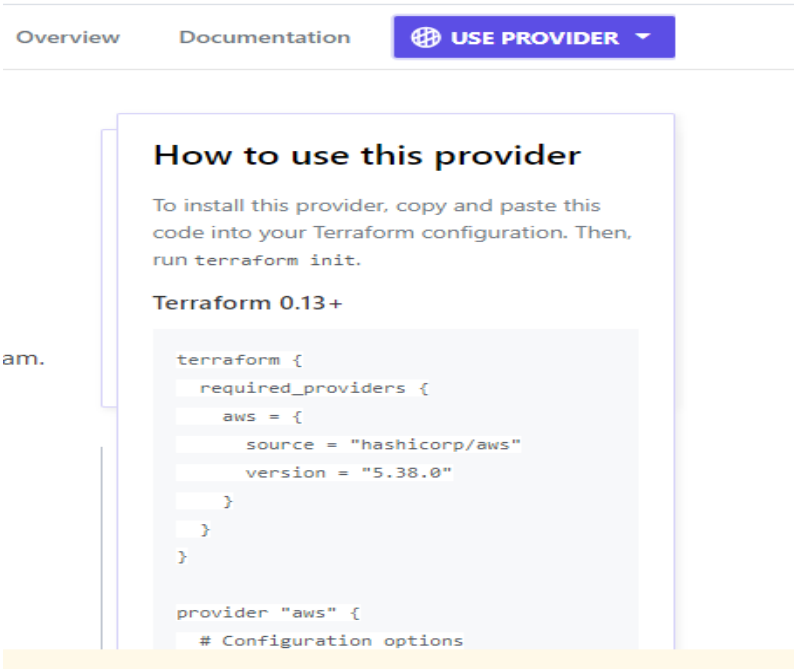
Nota: Este ejemplo no tiene costo alguno de uso de nube.

1. Vamos a la página de documentación, al provider de AWS.
    - https://registry.terraform.io/providers/hashicorp/aws/latest

2. Estando en la página, podemos mirar que versión usar.



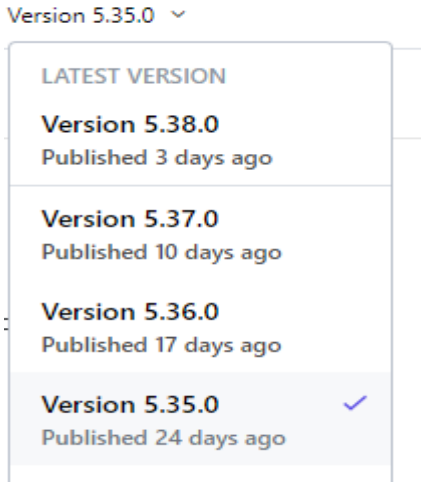
3. Al darle al botón de "USE PROVIDER", nos muestra un ejemplo de cómo usar una determinada versión.



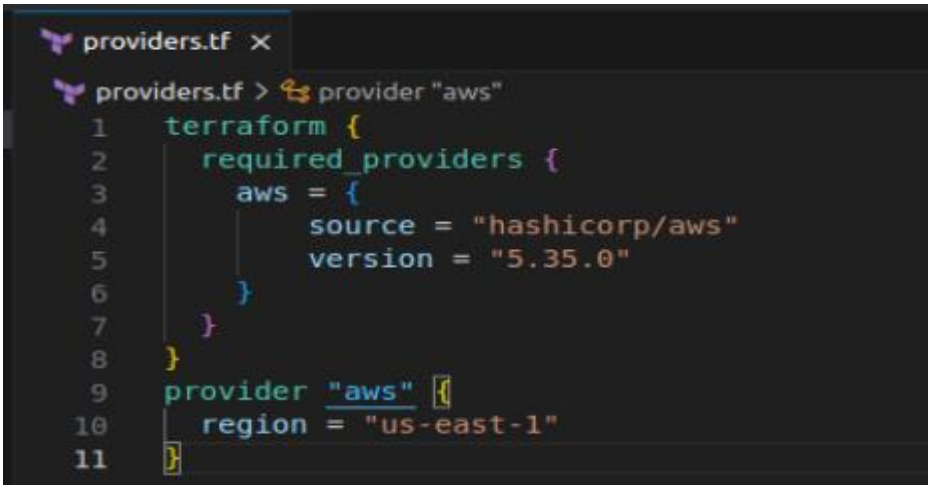
4. Vamos a crear una nueva carpeta, para el ejercicio 5.

```
jonatan@ubuntu:~/practica_terraform$ mkdir practica_05
jonatan@ubuntu:~/practica_terraform$ cd practica_05
jonatan@ubuntu:~/practica_terraform/practica_05$ code .
jonatan@ubuntu:~/practica_terraform/practica_05$
```

5. Estando en Visual Studio Code, creamos un archivo "providers.tf", Vamos a la página visitada anteriormente seleccionamos una versión un poco más antigua a la última disponible en el vamos a copiar el ejemplo que nos da al seleccionar "USE PROVIDER" y le añadimos la región en el provider "aws" (más adelante veras el ejemplo).
   - En mi caso usare la "5.38.0" del provider aws.



6. El archivo en Visual Studio Code, nos debe quedar así con el agregado de la región.



7. Vamos a la consola, estando en la carpeta ejecutamos "terraform init".
    - Como vemos, nos descarga la versión que le pedimos anteriormente del provider de aws "5.35.0".

```
jonatan@ubuntu:~/practica_terraform/practica_05$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.35.0"...
- Installing hashicorp/aws v5.35.0...
- Installed hashicorp/aws v5.35.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

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

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
jonatan@ubuntu:~/practica_terraform/practica_05$
```

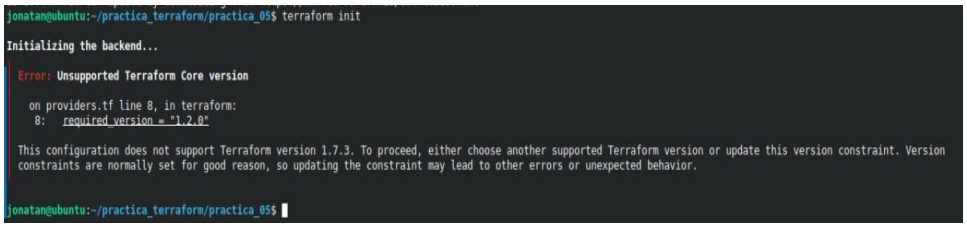
8. Ahora vamos a requerir una versión de terraform especifica.
9. Vamos al Visual Studio Code y vamos a editar el archivo "providers.tf" y añadimos la línea 8 que se ve en la siguiente imagen.

```
1  terraform {
2    required_providers {
3      aws = {
4          source = "hashicorp/aws"
5          version = "5.35.0"
6      }
7    }
8    required_version = "1.2.0"
9  }
```

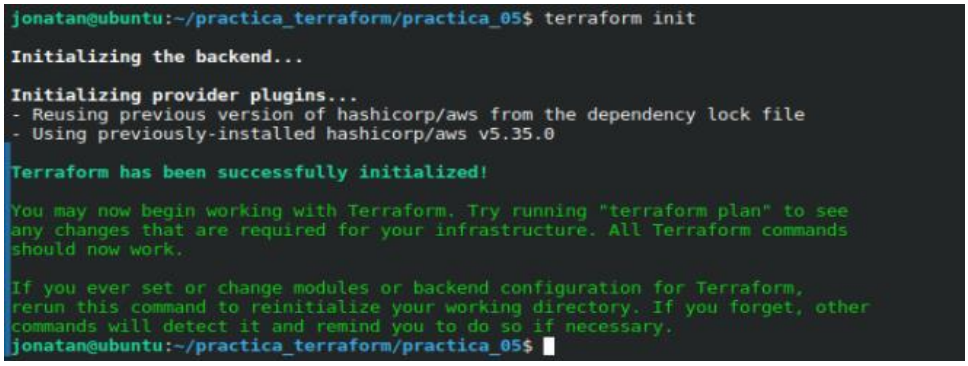
10. Volvemos a la consola y ejecutamos el comando "terraform --version"
    - En mi caso tengo la version "1.7.3"

```
jonatan@ubuntu:~/practica_terraform/practica_05$ terraform --version
Terraform v1.7.3
on linux_amd64
+ provider registry.terraform.io/hashicorp/aws v5.35.0

Your version of Terraform is out of date! The latest version
is 1.7.4. You can update by downloading from https://www.terraform.io/downloads.html
jonatan@ubuntu:~/practica_terraform/practica_05$
```

11. Ejecutamos el comando "terraform init".
    - Esto nos va a generar un error en caso de que sean diferentes, la versión instala y la versión requerida en el código.



12. Vamos al Visual Studio Code y editamos el archivo "terraform.tf", y colocamos la versión local instalada (terraform --version).
13. Ejecutamos el comando "terraform init".
    - Salió todo correcto.

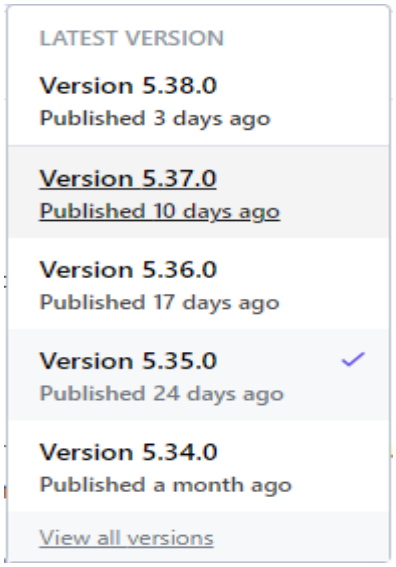


14. Ahora vamos a manejar los Terraform constraints.
    - "~>1.7.3", esto nos quiere decir que cualquier subversión en la 1.7.0, por ejemplo "1.7.1", "1.7.2", son aceptadas.
    - Vamos al Visual Studio Code y editamos el archivo "terraform.tf"
    - En mi caso tengo la version "1.7.3" instalada de manera local, pero vamos a escribir "~>1.7.0" para decirle que acepta todas sus subversiones.

15. Volvemos a la consola, ejecutamos el comando “terraform init”.
    - Como veremos, todo funciona bien.

```
jonatan@ubuntu:~/practica_terraform/practica_05$ terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.35.0

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.
jonatan@ubuntu:~/practica_terraform/practica_05$
```

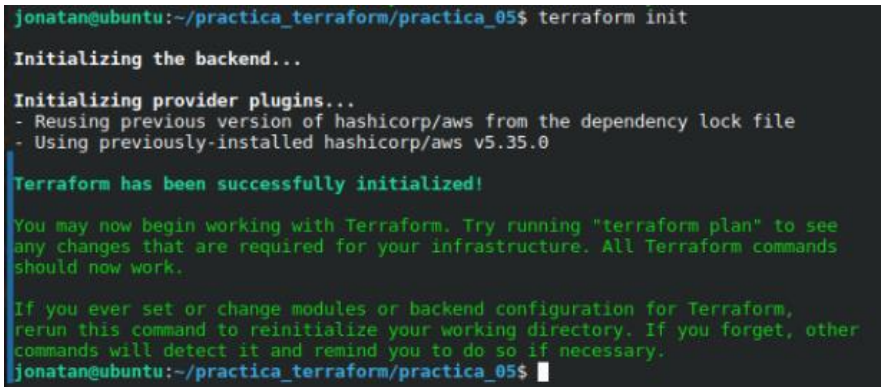
16. Ahora vamos a agregar condicionales con la versión del provider de aws.



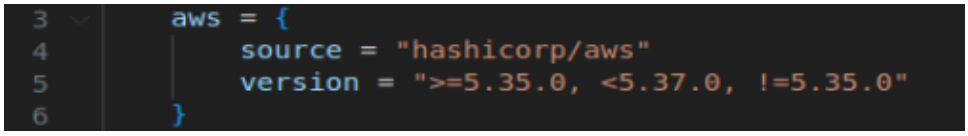
- La versión debe ser mayor o igual a la “5.35.0” y menor a la “5.37.0”

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = ">=5.35.0, <5.37.0"
    }
  }
```

17. Ahora vamos a la consola, y ejecutamos de nuevo “terraform init”.



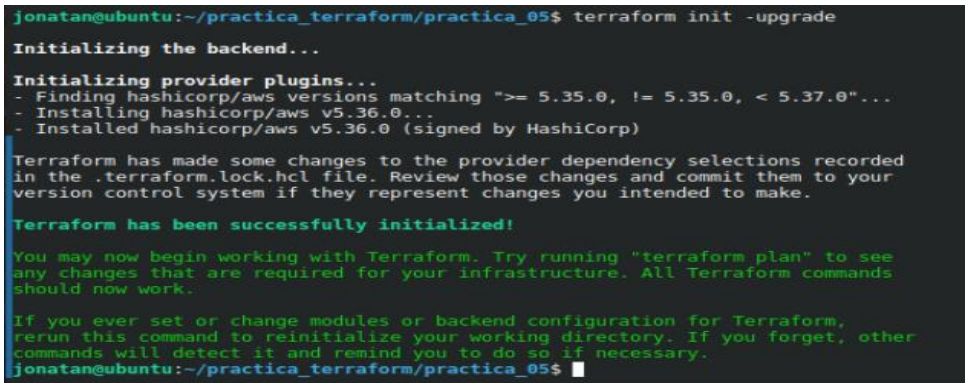
- Como vemos sigue con la misma versión que teníamos, pero cumpliendo con la condicional que pusimos.
- Ahora en el condicional vamos a colocar que sea una diferente a la que tenemos, en este caso tenemos la “5.35.0”, pero que siga con los parámetros anteriores.



- Volvemos a la consola, y ahora ejecutamos “terraform init -upgrade”

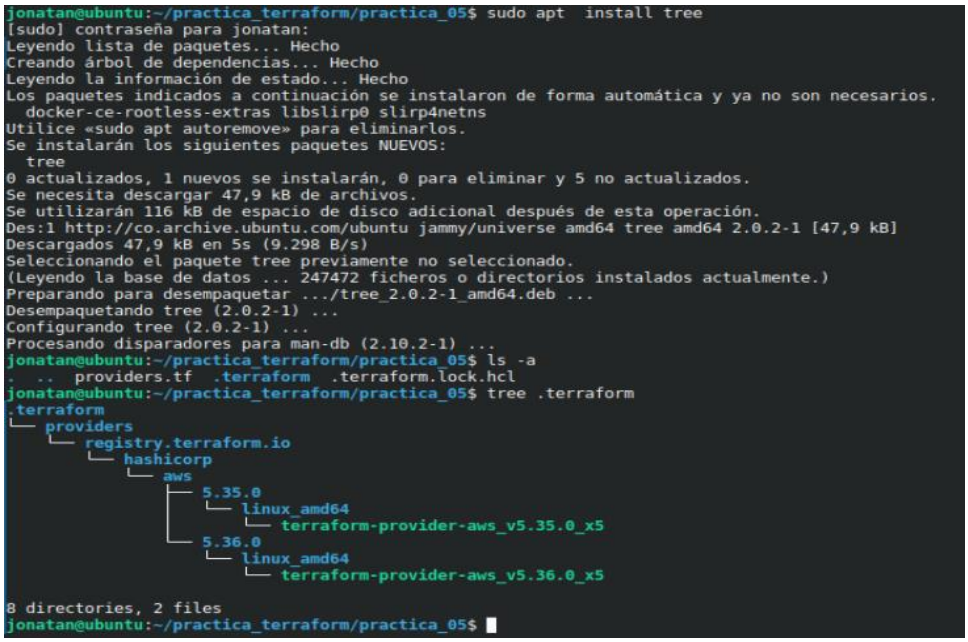
Nota: terraform init –upgrade : se utiliza en terraform para actualizar los plugins y módulos a la versión compatible con el proyecto.

- Volvemos a la consola, y ahora ejecutamos “terraform init -upgrade”.

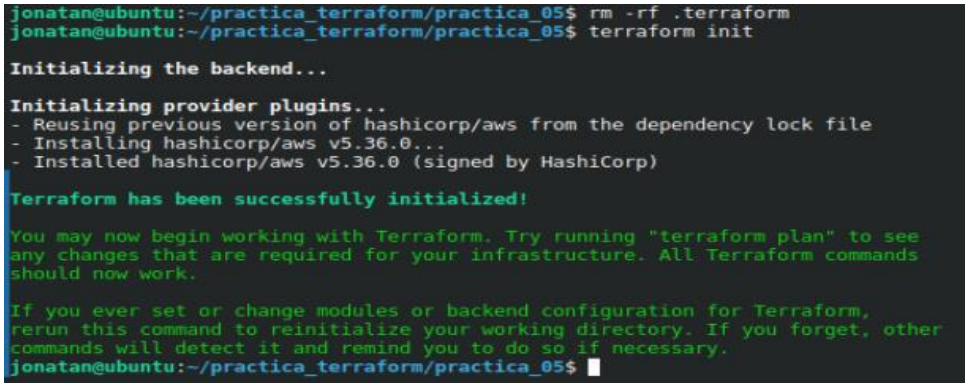


- Como vemos se instaló la versión “5.36.0”.

18. Ahora ejecutamos los siguientes comandos:
    - sudo apt install tree
    - ls –a
    - tree .terraform
    - Veremos las versiones del provider aws instaladas.



19. Ahora si queremos borrar, y dejar solo una versión, hacemos lo siguiente:
    - rm –rf .terraform
    - terraform init
    - tree .terraform
    - Veremos que ahora solo tendremos una versión del provider aws



20. Ahora vamos a usar VPC, para ello nos dirigimos a la documentación:
    - https://registry.terraform.io/providers/hashicorp/aws/5.35.0/docs/resources/vpc

21. Copiamos el ejemplo básico:

Example Usage

Basic usage:

```
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"
}
```

Copy

22. Creamos un archivo "vpc.tf", pegamos lo anteriormente copiado y lo editamos de la siguiente manera.

providers.tf | vpc.tf

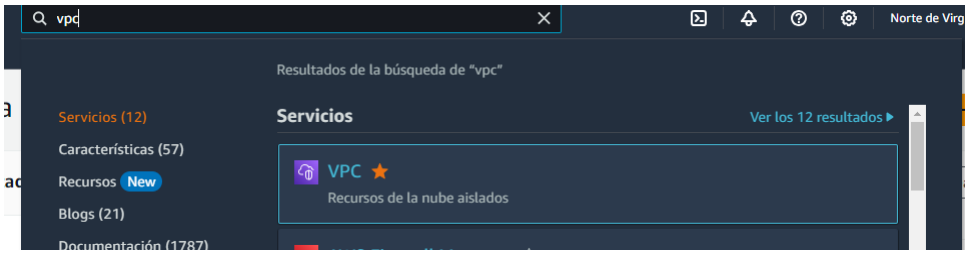
vpc.tf > resource "aws_vpc" "vpc_virginia"

```
resource "aws_vpc" "vpc_virginia" {
  cidr_block = "10.10.0.0/16"
  tags = {
    Name = "VPC_VIRGINIA"
    env = "Dev"
  }
}
```

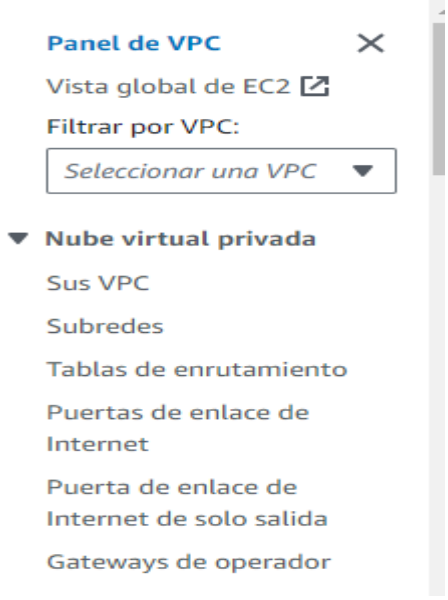
23. Nos dirigimos a la consola y ejecutamos "terraform plan".

```
jonatan@ubuntu:~/practica_terraform/practica_05$ terraform plan

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

Terraform will perform the following actions:

  # aws_vpc.vpc_virginia will be created
  + resource "aws_vpc" "vpc_virginia" {
      + arn                                  = (known after apply)
      + cidr_block                           = "10.10.0.0/16"
      + default_network_acl_id               = (known after apply)
      + default_route_table_id               = (known after apply)
      + default_security_group_id            = (known after apply)
      + dhcp_options_id                      = (known after apply)
      + enable_dns_hostnames                 = (known after apply)
      + enable_dns_support                   = true
      + enable_network_address_usage_metrics = (known after apply)
      + id                                   = (known after apply)
      + instance_tenancy                     = "default"
      + ipv6_association_id                  = (known after apply)
      + ipv6_cidr_block                      = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + tags                                 = {
          + "Name" = "VPC_VIRGINIA"
          + "env"  = "Dev"
        }
      + tags_all                             = {
          + "Name" = "VPC_VIRGINIA"
          + "env"  = "Dev"
        }
    }

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

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
jonatan@ubuntu:~/practica_terraform/practica_05$
```

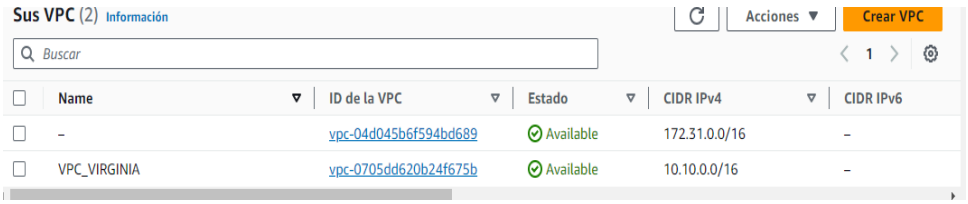
24. Vamos a la página de AWS, ingresamos con el usuario IAM, buscamos VPC, le damos en la estrella, y damos click en VPC.



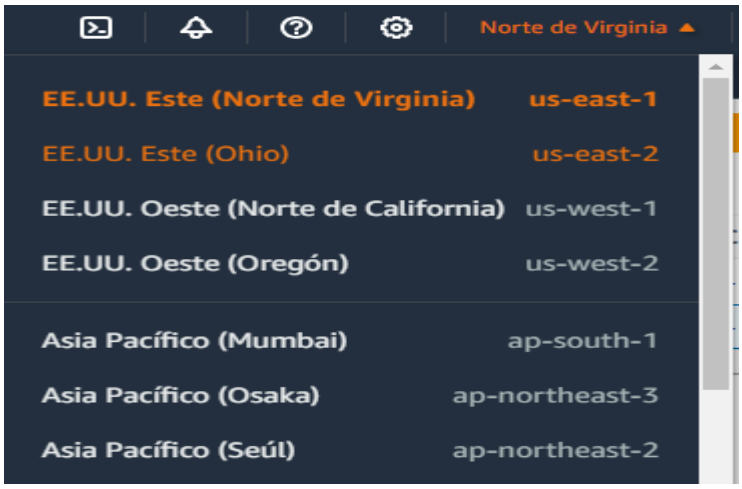
25. Ahora en el menú del lado izquierdo, damos click en sus VPC.



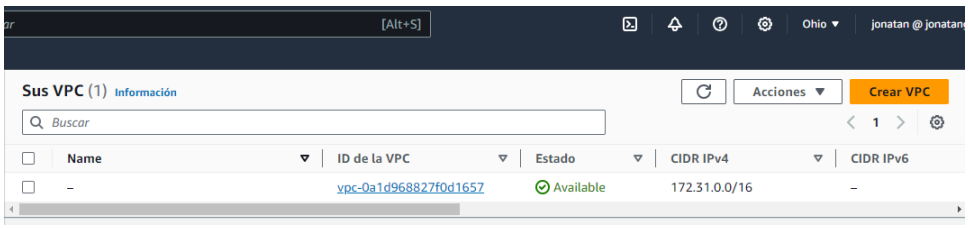
26. Volvemos a la consola, y ejecutamos el comando "terraform apply".
27. En la página de aws, actualizamos y veremos la VPC creada con una vpc predeterminada.



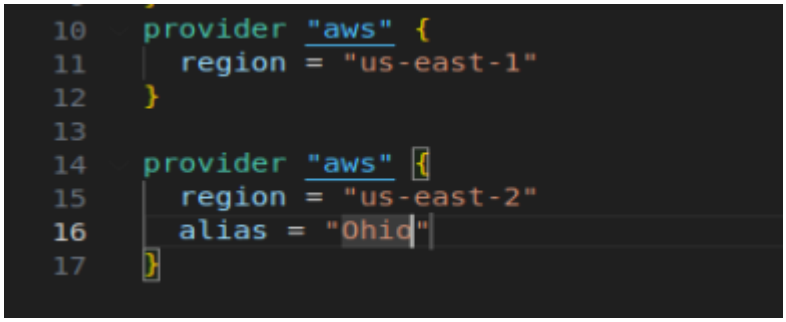
28. Ahora queremos cambiar de región, damos click en (Ohio).



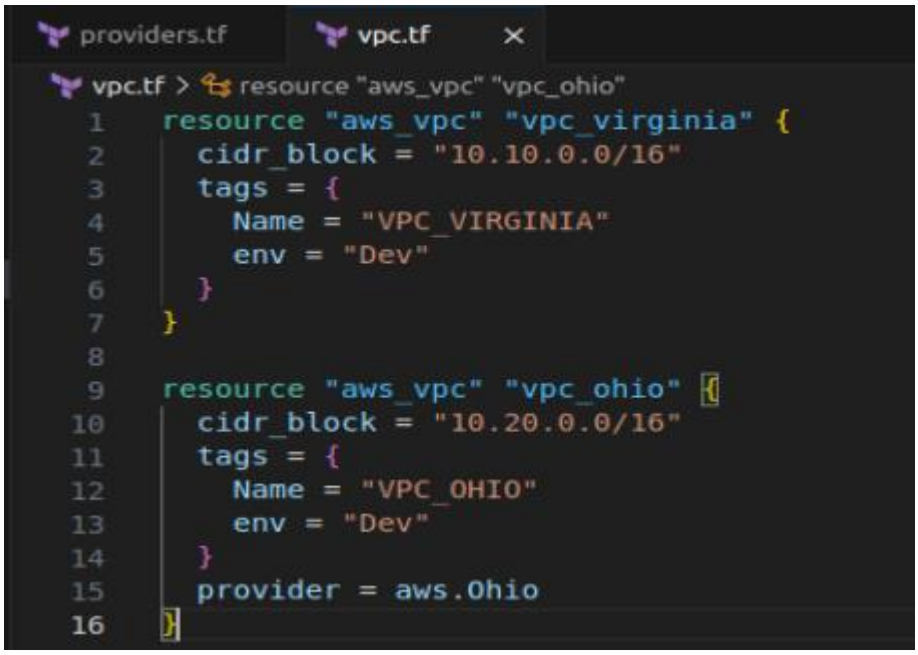
29. Ahora queremos cambiar de región, damos click en (Ohio), veremos que ya no está la anterior VPC creada y solo nos muestra la predeterminada.



30. Para crear una VPC en otra región al tiempo, haremos lo siguiente, nos dirigimos a Visual Studio Code en el archivo "providers.tf" y agregamos lo siguiente (línea 14-17).

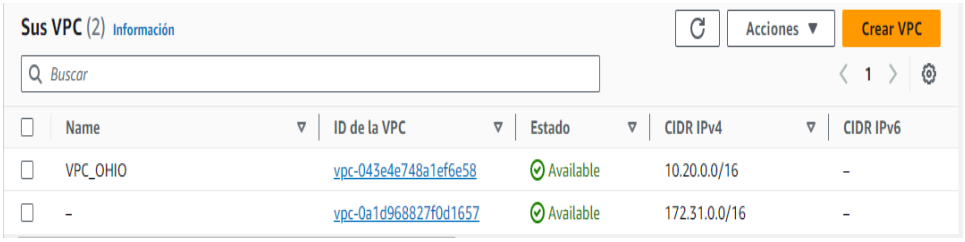


31. Ahora nos dirigimos al archivo "vpc.tf" y nos debe quedar así:
    - Cambiamos el id
    - Editamos la subnet, para evitar problemas si las llegamos a conectar.
    - Cambiamos el tag Name
    - Agregamos la etiqueta provider = aws.Ohio

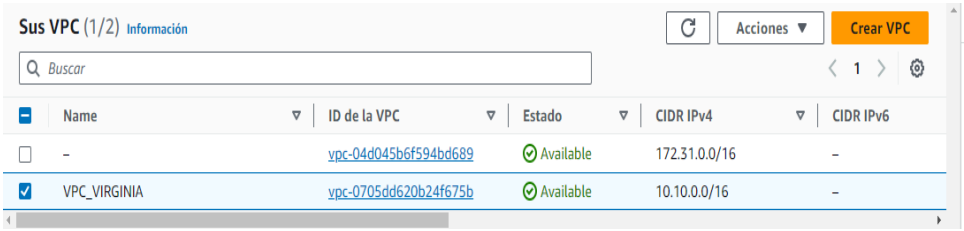


32. Ahora nos dirigimos a consola, ejecutamos:
    - terraform plan
    - terraform apply

33. Ahora nos dirigimos a la página de aws, en la región de Ohio actualizamos:



34. Ahora nos dirigimos a la región de Virginia y vemos que la vpc de esta región sigue ahí.



35. Ahora aplicamos "terraform destroy".

Nota: Las VPC no generan ningún costo a menos de que sean más de 5 por región.