TERRAFORM

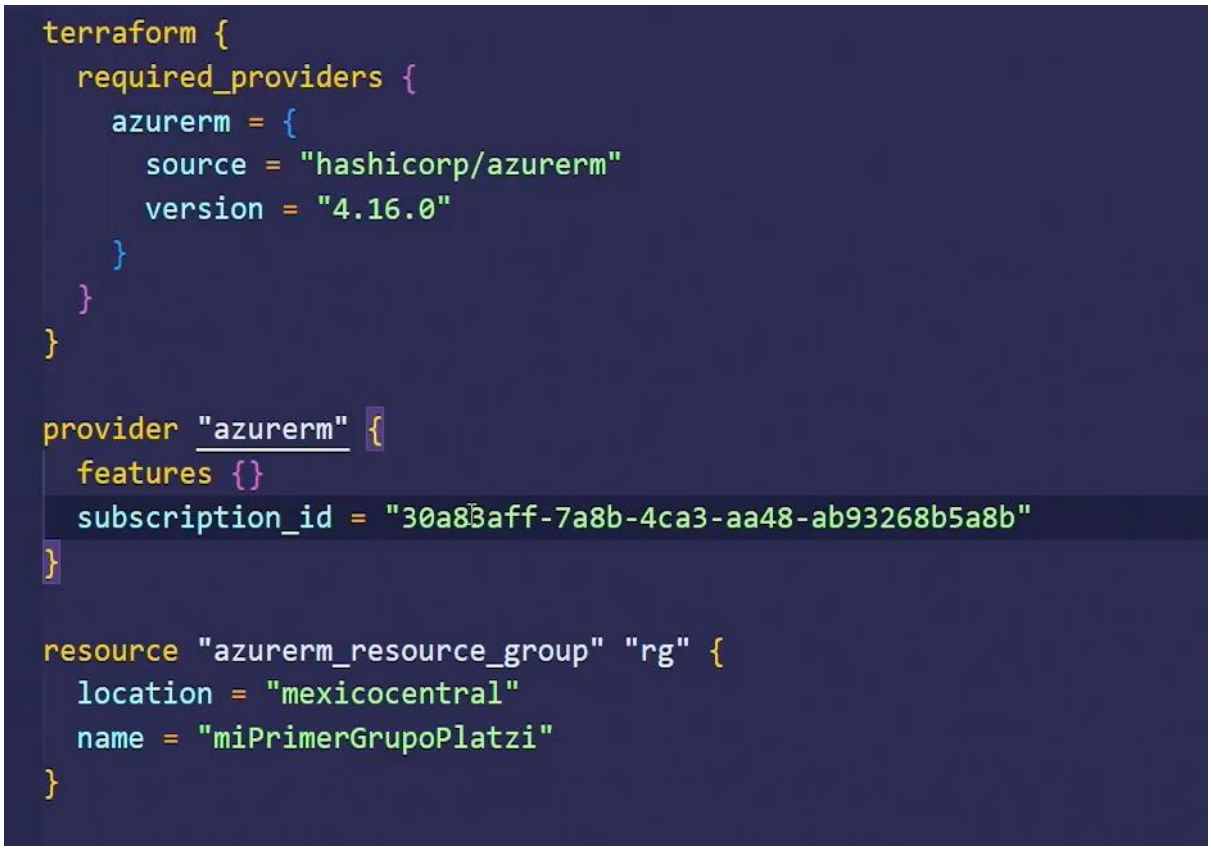
Comandos Básicos de Terraform

- `terraform init`: Prepara tu directorio de trabajo. **¡Siempre al iniciar!**
- `terraform plan`: Muestra qué cambios hará Terraform. **¡Siempre antes de aplicar!**
- `terraform apply`: Ejecuta los cambios propuestos. **¡Crea o modifica recursos!**
- `terraform destroy`: Elimina los recursos gestionados. **¡Cuidado, borra todo!**
- `terraform validate`: Comprueba la sintaxis de tus archivos. **¡Para errores rápidos!**

Documentacion Azure

https://registry.terraform.io/

Grupo de recursos



```
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "4.16.0"
    }
  }
}

provider "azurerm" {
  features {}
  subscription_id = "30a83aff-7a8b-4ca3-aa48-ab93268b5a8b"
}

resource "azurerm_resource_group" "rg" {
  location = "mexicocentral"
  name = "miPrimerGrupoPlatzi"
}
```

Terraform Plan

terraform plan -out plan.out
Este comando **calcula qué cambios hará Terraform y guarda ese plan exacto en un archivo** (`plan.out`).
Sirve para **revisar y compartir** los cambios antes de aplicarlos

Aplicar plan anterior:

terraform apply “plan.out”

Variables

En el mismo archivo main.tf
Usa default para asignar un valor por defecto a la variable

```hcl
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "4.16.0"
    }
  }
}

provider "azurerm" {
  features {}
  subscription_id = "30a83aff-7a8b-4ca3-aa48-ab93268b5a8b"
}

variable "rg_nombre" {
  type = string
  default = "miPrimerGrupoAmin"
}

resource "azurerm_resource_group" "rg" {
  location = "mexicocentral"
  name = var.rg_nombre
}
```

Configurar variable en otro archivo:

El archivo debe llamarse si o si [variables.tf]

Asignar valores a las variables

El archivo debe llamarse terraform.tfvars
Este archivo debe agregarse el gitignore porque puede tener información sensible

En cambio debe de crearse un archivo terraform.tfvars.example que si se puede subir a git

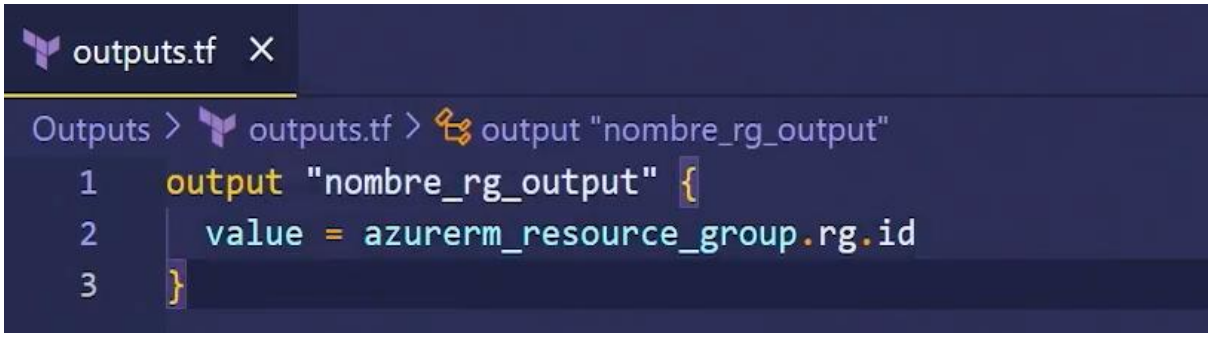
Outputs

En el mismo archivo main.tf

Este comando sirve para saber el valor de alguna variable del recurso

Configurar output en otro archivo

Este archivo debe llamarse si o si “outputs.tf”



Storage account (cuentas de almacenamiento)

```hcl
provider "azurerm" {
  features {}
  subscription_id = "30a83aff-7a8b-4ca3-aa48-ab93268b5a8b"
}

resource "azurerm_resource_group" "rg" {
  location = "brazilsouth"
  name = "grupoAlmacenamiento"
}

resource "azurerm_storage_account" "storage_account" {
  name                     = "amindevplatzi"
  resource_group_name      = azurerm_resource_group.rg.name
  location                 = azurerm_resource_group.rg.location
  account_tier             = "Standard"
  account_replication_type = "GRS"

  tags = {
    environment = "staging"
  }
}
```

variable heradada

Se pueden agregar nuevas propiedades que no aparecen por defecto

```hcl
account_tier                  = "Standard"
account_replication_type      = "GRS"
public_network_access_enabled = false
```

Obtener valores del Storage Account

```
output "cadena_conexion" {
  value = azurerm_storage_account.storage_account.primary_connection_string
}
```

Tener cuidado con valores sensibles como este que son de un access token

```
Planning failed. Terraform encountered an error while generating this plan.

  Error: Output refers to sensitive values

    on main.tf line 33:
    33: output "cadena_conexion" {
```

Marcarlo como sensible

```
output "cadena_conexion" {
  value = azurerm_storage_account.storage_account.primary_connection_string
  sensitive = true
}
```

Crear proveedor

Este archivo debe de llamarse si o si “provider.tf”

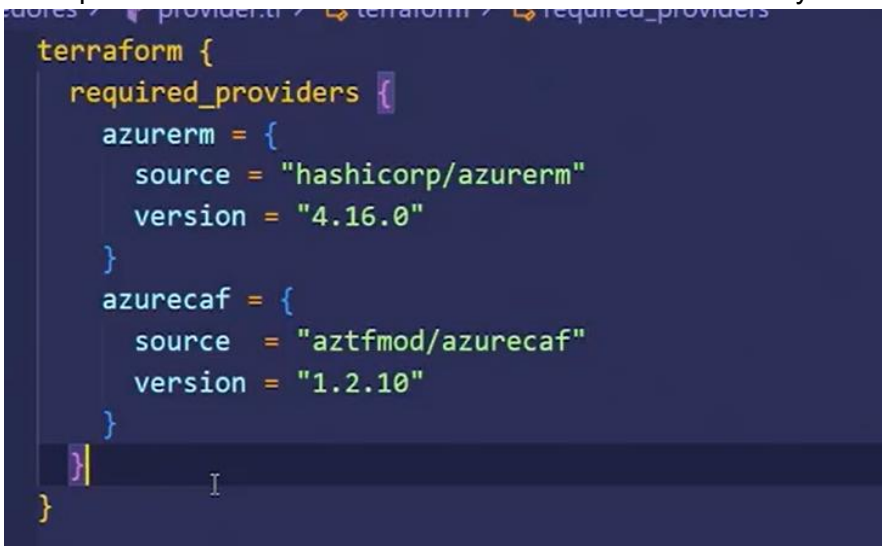
main.tf × provider.tf ×

proveedores > provider.tf > provider "azurerm"

```
1   terraform {
2     required_providers {
3       azurerm = {
4         source = "hashicorp/azurerm"
5         version = "4.16.0"
6       }
7     }
8   }
9
10  provider "azurerm" {
11    features {}
12    subscription_id = "30a83aff-7a8b-4ca3-aa48-ab93268b5a8b"
13  }
```

Agregar proveedor Azure CAF

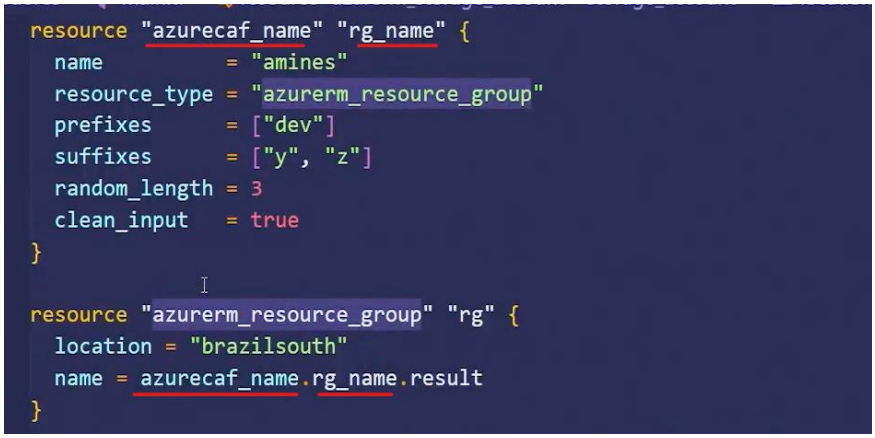
Sirve para colocar nombres a los recursos automáticamente y evitar que sea manual



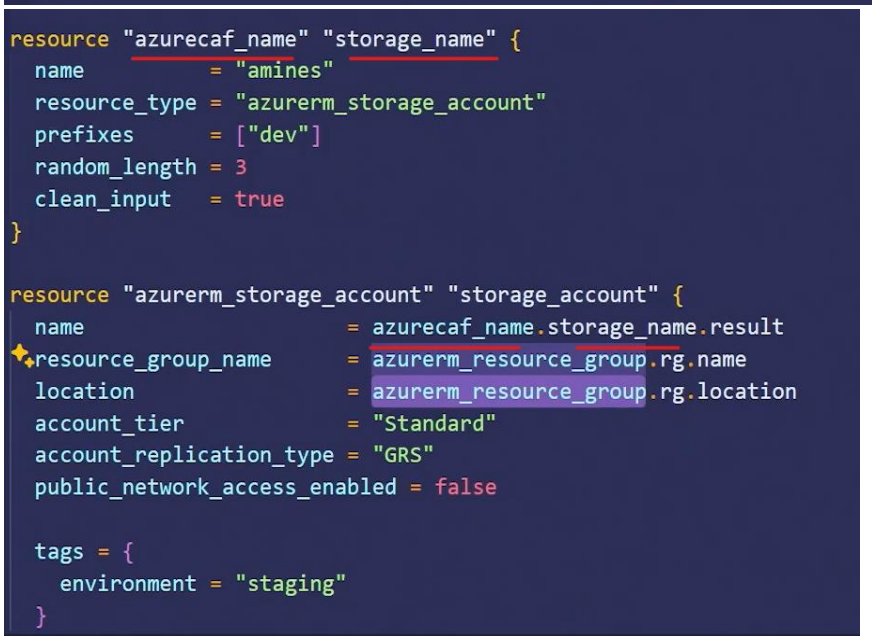
```hcl
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "4.16.0"
    }
    azurecaf = {
      source  = "aztfmod/azurecaf"
      version = "1.2.10"
    }
  }
}
```

Se debe de hacer un nuevo terraform init, al agregar un nuevo proveedor

Para cada recurso se tiene que crear un azurecaf

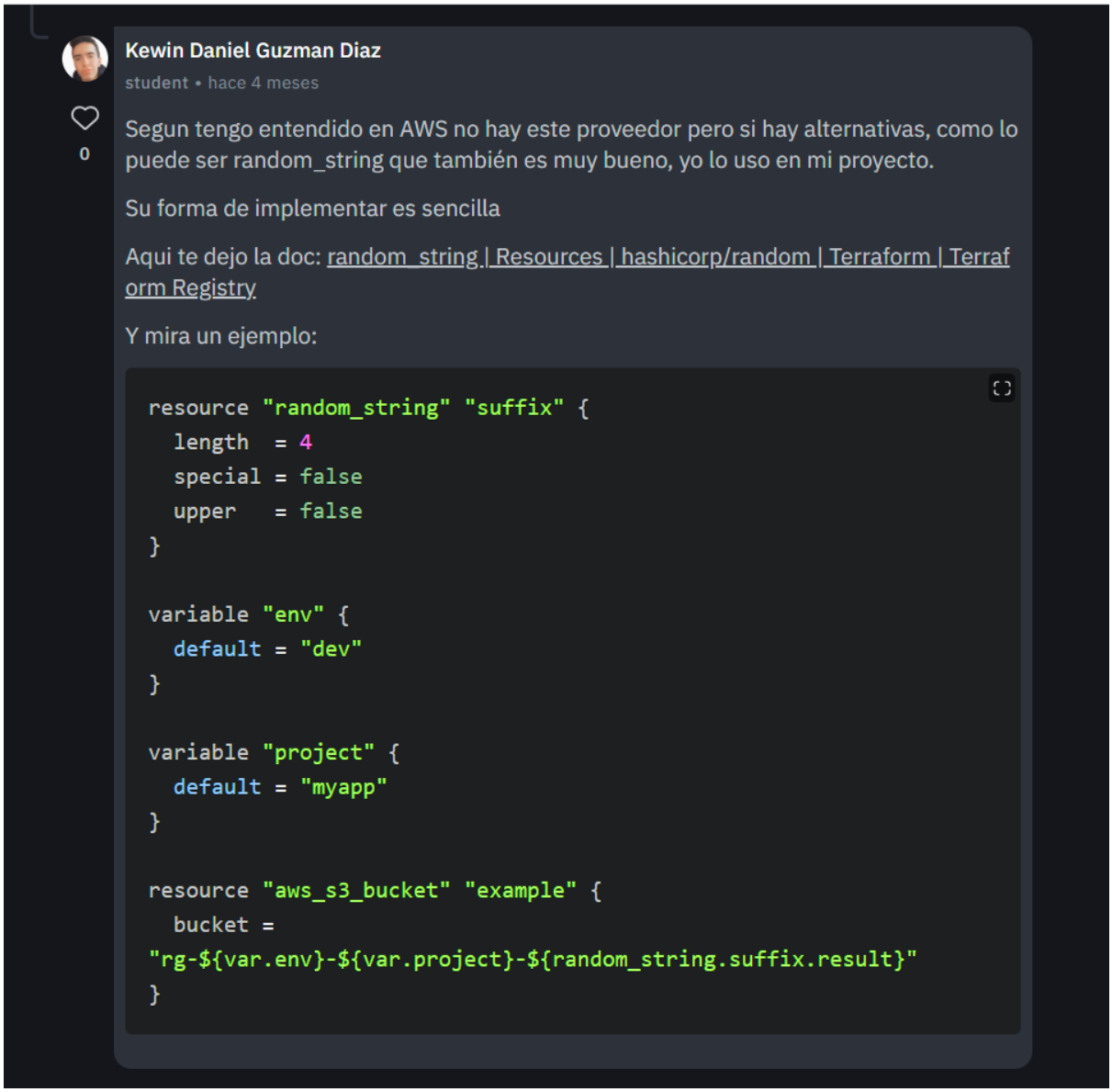


```hcl
resource "azurecaf_name" "rg_name" {
  name          = "amines"
  resource_type = "azurerm_resource_group"
  prefixes      = ["dev"]
  suffixes      = ["y", "z"]
  random_length = 3
  clean_input   = true
}

resource "azurerm_resource_group" "rg" {
  location = "brazilsouth"
  name = azurecaf_name.rg_name.result
}
```



```hcl
resource "azurecaf_name" "storage_name" {
  name          = "amines"
  resource_type = "azurerm_storage_account"
  prefixes      = ["dev"]
  random_length = 3
  clean_input   = true
}

resource "azurerm_storage_account" "storage_account" {
  name                     = azurecaf_name.storage_name.result
  resource_group_name      = azurerm_resource_group.rg.name
  location                 = azurerm_resource_group.rg.location
  account_tier             = "Standard"
  account_replication_type = "GRS"
  public_network_access_enabled = false

  tags = {
    environment = "staging"
  }
```

Agregar alternativa de Azure CAF para otros proveedores



Kewin Daniel Guzman Diaz

student • hace 4 meses

0

Segun tengo entendido en AWS no hay este proveedor pero si hay alternativas, como lo puede ser random_string que también es muy bueno, yo lo uso en mi proyecto.

Su forma de implementar es sencilla

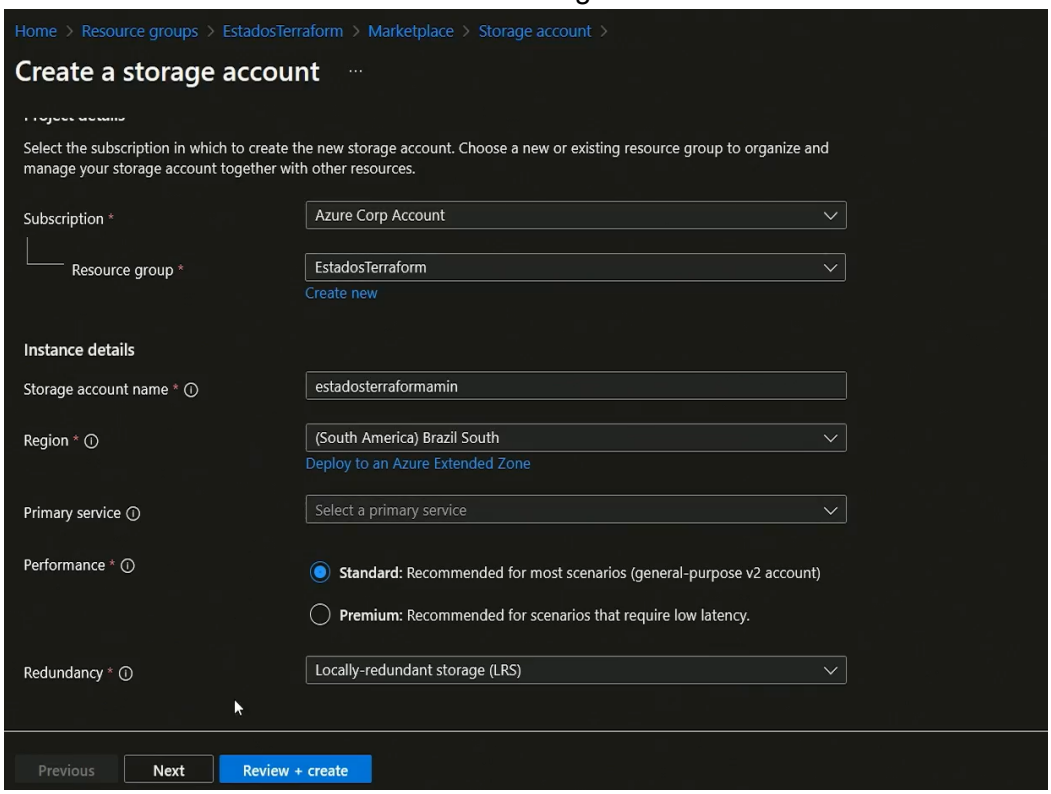
Aqui te dejo la doc: random_string | Resources | hashicorp/random | Terraform | Terraform Registry

Y mira un ejemplo:

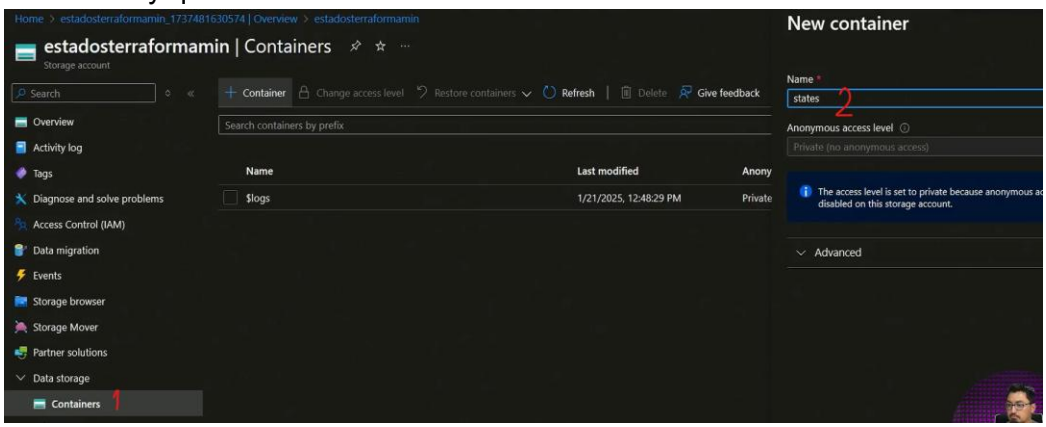
```
resource "random_string" "suffix" {
  length  = 4
  special = false
  upper   = false
}

variable "env" {
  default = "dev"
}

variable "project" {
  default = "myapp"
}

resource "aws_s3_bucket" "example" {
  bucket =
"rg-${var.env}-${var.project}-${random_string.suffix.result}"
}
```

Estado remoto en Azure

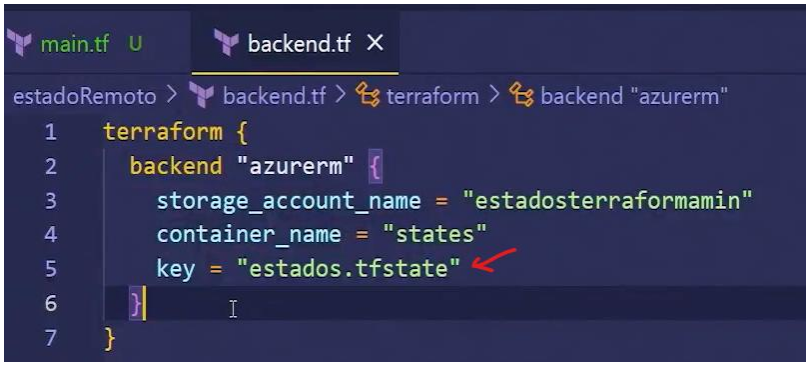
Para hacer esto es necesario crear un storage account manualmente



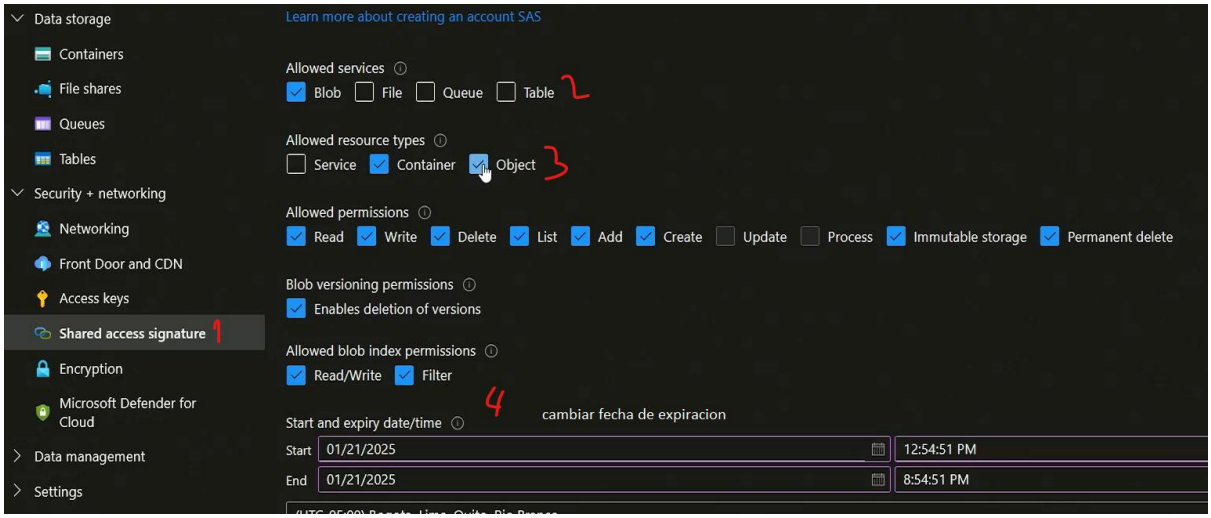
Primero hay que crear un contenedor



Ahora hay que crear el archivo que debe de llamarse si o si “backend.tf”



Crear el SAS para el Storage Account



Copiar el valor

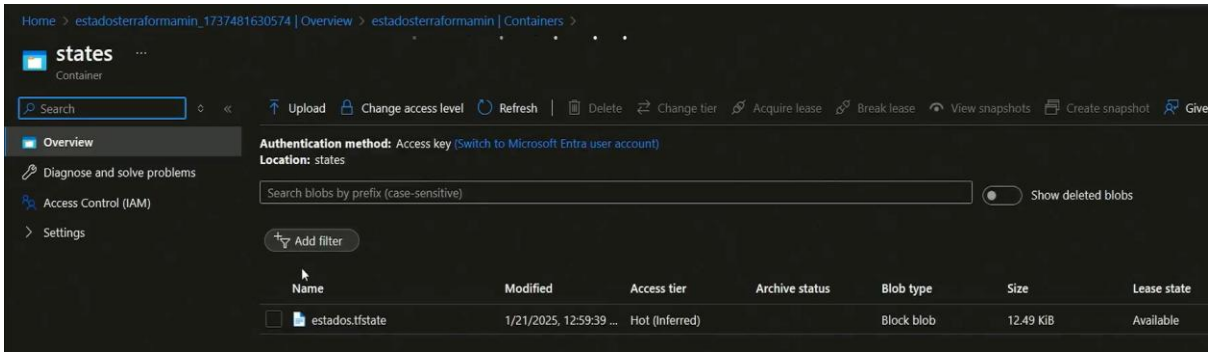
Ejecutar el terraform init de la siguiente manera incluyendo el backend

```
aminespinoza@CompuAmin:/mnt/c/Users/amine/Documents/Github/curso-terraform/estadoRemoto$ terra
form init -backend-config="sas_token=sv=2022-11-02&ss=b&srt=co&sp=rwdlaciytfx&se=2025-01-29T01
:54:51Z&st=2025-01-21T17:54:51Z&spr=https&sig=6Hm3RUlMntxJyJD8JDRVO60exOR78EUACVDp6KimWB8%3D"
Initializing the backend...
```

Despues debe aplicar los siguientes comandos

```
terraform plan -out plan.out

terraform apply “plan.out”
```

A partir de crear este estado en Azure, ya no existirá el terraform.tfstate en local sino estará en la nube, el cual esta dentro del contenedor antes creado.



Importante: algunos usuarios reportan que no es necesario el SAS

Módulo Data (no terraform)

Usar recursos que ya han sido creados anteriormente en Azure sin terraform

main.tf

data > main.tf > resource "azurerm_storage_account" "storage_account"

```
1  data "azurerm_resource_group" "imported_rg" {
2    name = "GrupoPrevio"
3  }
4
5  resource "azurerm_storage_account" "storage_account" {
6    name = "almacenamientoprevio"
7    resource_group_name = data.azurerm_resource_group.imported_rg.name
8    location = data.azurerm_resource_group.imported_rg.location
9    account_tier             = "Standard"
10   account_replication_type = "GRS"
11 }
```

Para proteger un recurso existente en producción y evitar que sea eliminado y recreado al ejecutar terraform apply

```
resource "aws_instance" "example" {
  ami           = "ami-12345678"
  instance_type = "t2.micro"

  lifecycle {
    prevent_destroy = true
  }
}
```

Comandos de terraform

terraform fmt –recursive (formatear el texto, para ordenarlo)
terraform validate (validaciones básicas antes de ejecutar el plan)

Máquinas virtuales (VM)

Elementos a crear

- Grupo de recursos
- Red virtual
- Subred
- IP Pública
- Grupo de seguridad
- Interfaz de red
- Máquina virtual

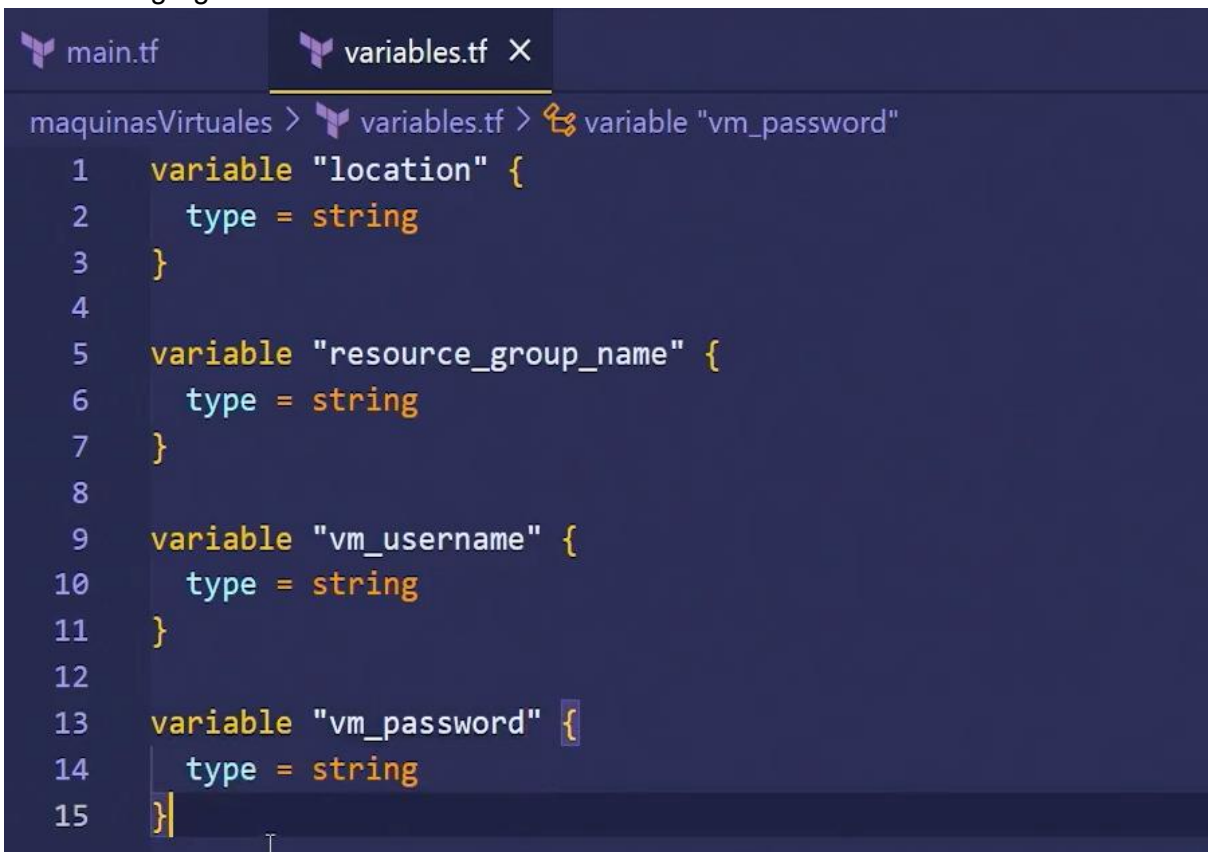
```
resource "azurerm_resource_group" "rg" {
  location = var.location
  name     = var.resource_group_name
}

resource "azurerm_virtual_network" "my_terraform_network" {
  name                = "amines-vnet"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
}

resource "azurerm_subnet" "my_terraform_subnet" {
  name                 = "amines-subnet"
  resource_group_name  = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.my_terraform_network.name
  address_prefixes     = ["10.0.1.0/24"]
}
```

```
resource "azurerm_public_ip" "my_terraform_public_ip" {
  name                = "amines-public-ip"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  allocation_method   = "Static"
}

resource "azurerm_network_security_group" "my_terraform_nsg" {
  name                = "amines-nsg"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  security_rule {
    name                       = "RDP"
    priority                   = 1000
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "*"
    source_port_range          = "*"
    destination_port_range     = "3389"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }
}
```

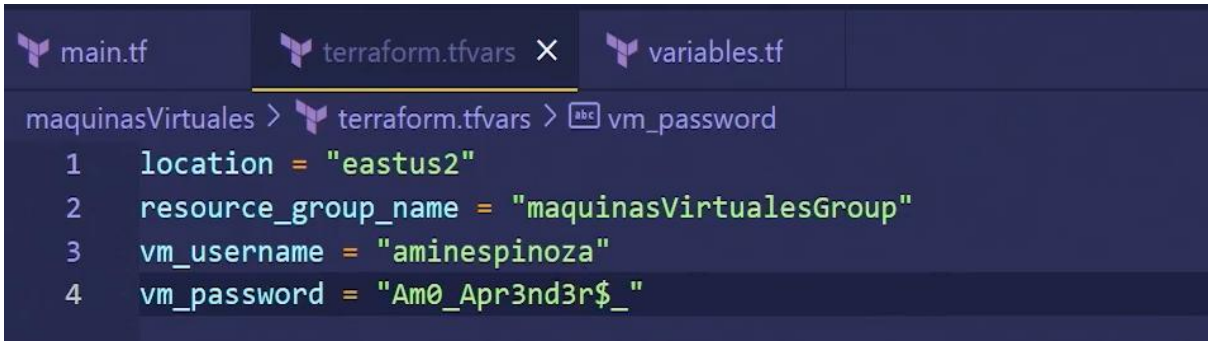
```
resource "azurerm_network_interface" "my_terraform_nic" {
  name                = "amines-nic"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  ip_configuration {
    name                          = "my_nic_configuration"
    subnet_id                     = azurerm_subnet.my_terraform_subnet.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = azurerm_public_ip.my_terraform_public_ip.id
  }
}

resource "azurerm_network_interface_security_group_association" "nic_association" {
  network_interface_id      = azurerm_network_interface.my_terraform_nic.id
  network_security_group_id = azurerm_network_security_group.my_terraform_nsg.id
}
```

```
 resource "azurerm_windows_virtual_machine" "main" {
   name                  = "amines-vm"
   admin_username        = var.vm_username
   admin_password        = var.vm_password
   location              = azurerm_resource_group.rg.location
   resource_group_name   = azurerm_resource_group.rg.name
   network_interface_ids = [azurerm_network_interface.my_terraform_nic.id]
   size                  = "Standard_DS1_v2"

   os_disk {
     name                 = "myOsDisk"
     caching              = "ReadWrite"
     storage_account_type = "Premium_LRS"
   }

   source_image_reference {
     publisher = "MicrosoftWindowsServer"
     offer     = "WindowsServer"
     sku       = "2022-datacenter-azure-edition"
     version   = "latest"
   }
 }
```

Variables agregadas



Valores para las variables



Ver el código completo aquí

https://github.com/platzi/curso-terraform/blob/main/maquinasVirtuales/main.tf

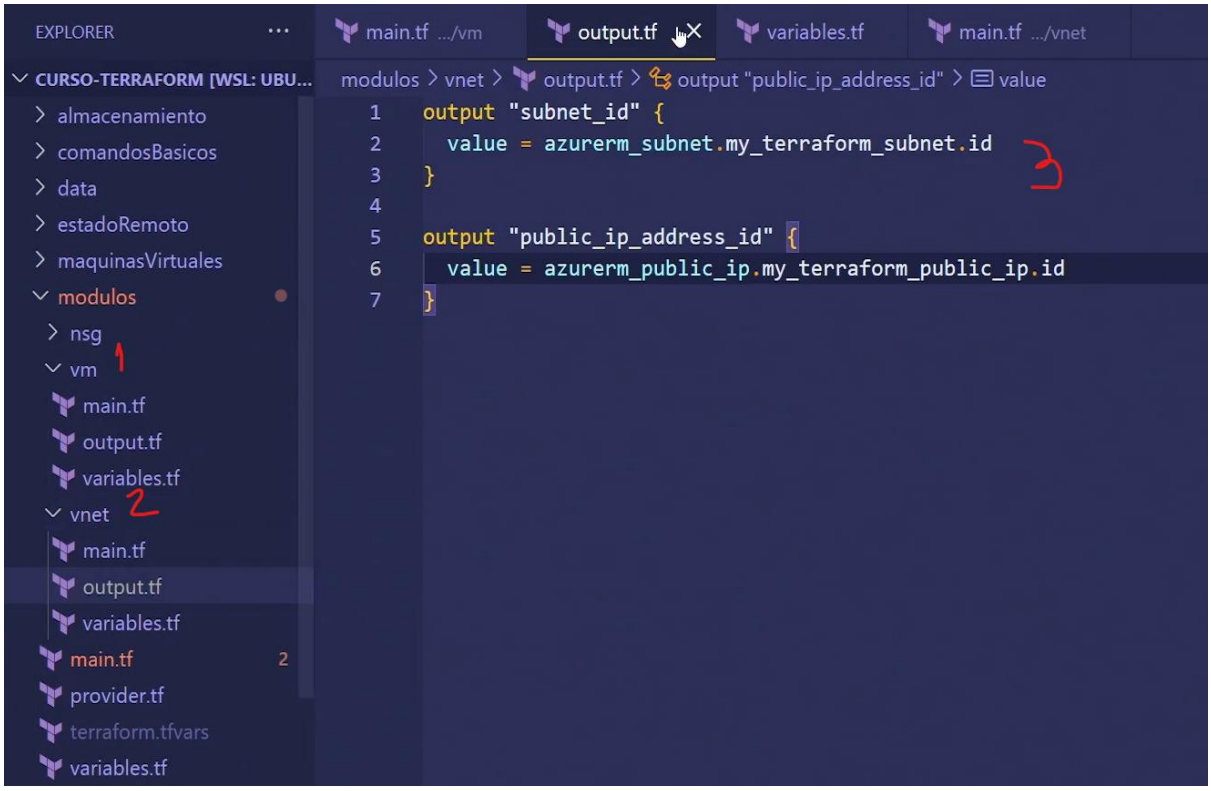
Instalar IIS en la VM

El código completo esta en la ruta anterior de git

```
resource "azurerm_virtual_machine_extension" "web_server_install" {
  name                       = "amines-wsi"
  virtual_machine_id         = azurerm_windows_virtual_machine.main.id
  publisher                  = "Microsoft.Compute"
  type                       = "CustomScriptExtension"
  type_handler_version       = "1.8"
  auto_upgrade_minor_version = true

  settings = <<SETTINGS
    {
      "commandToExecute": "powershell -ExecutionPolicy Unrestricted Install-Windows
    }
  SETTINGS
}
```

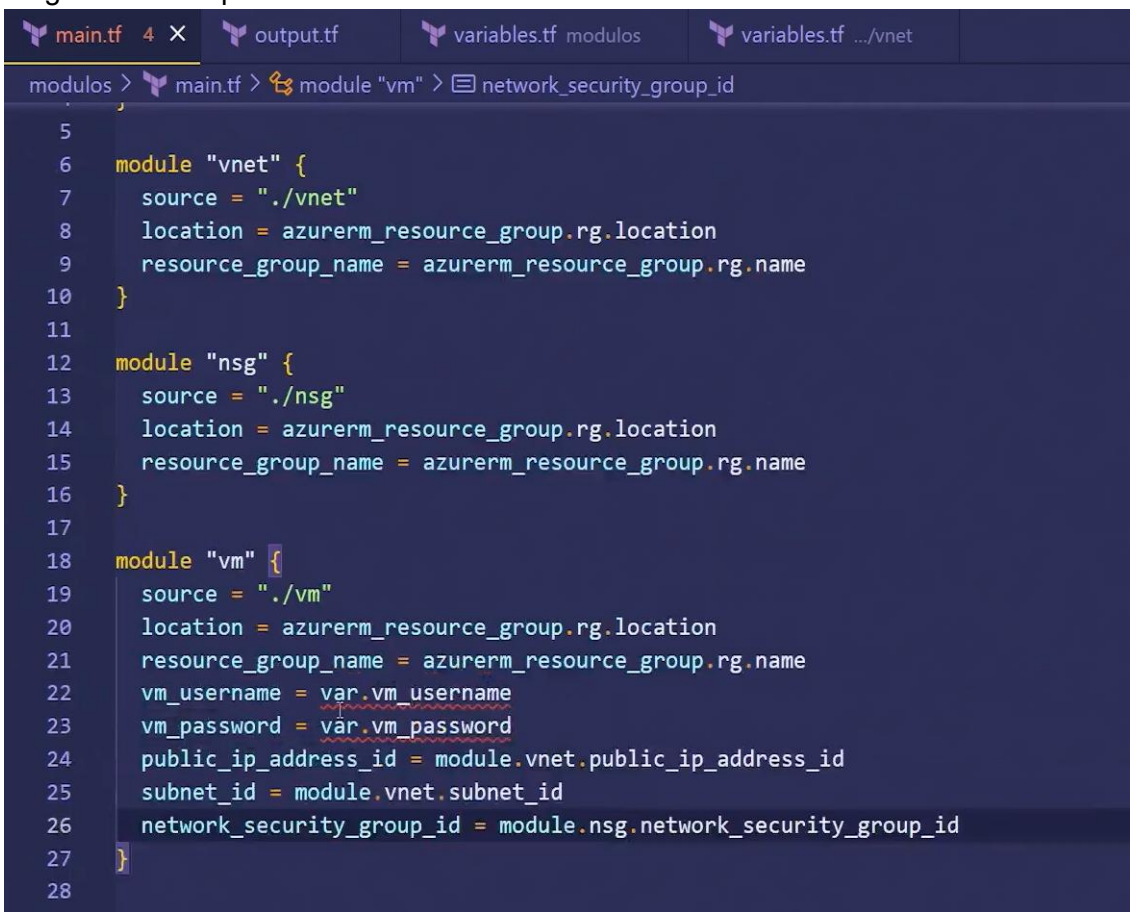
Comunicación entre módulos en Terraform

La idea de esto es compatir variables o información entre dos módulos (carpeta de vscode), ya una podría estar creando un recurso que necesita la otra carpeta, para este caso se está creando una VM y en otra carpeta la SubNet, para esto es necesario obtener a través de un output los valores resultantes de la SubNet para pasarlos a la VM.
IMPORTANTE: como se ve en la imagen todos estas carpetas pertenecen a una general llamada “módulos” en la cual también tiene sus archivos de terraform.



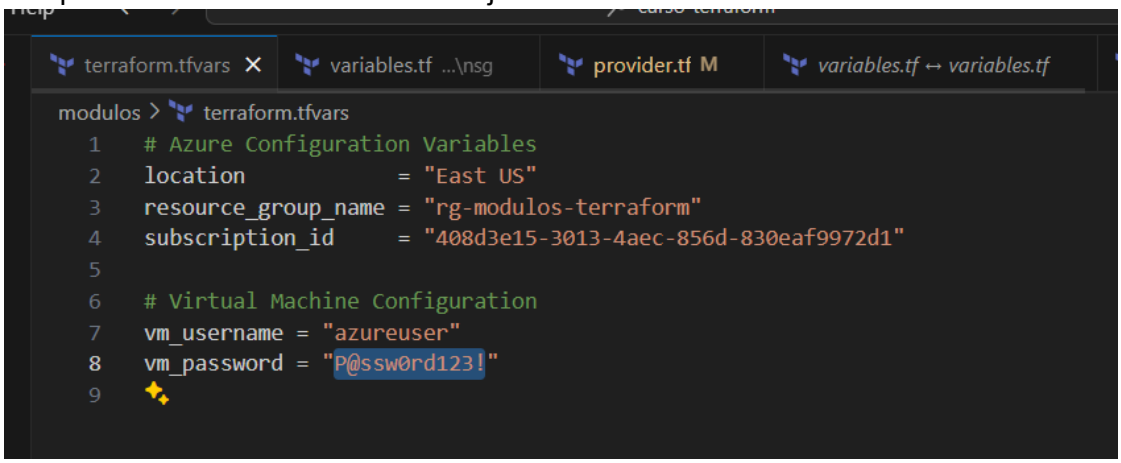
IMPORTANTE: Recordar que el nombre de los outputs, debe llamarse de igual manera que las variables que se van usar, es decir en el módulo de **vnet** el output subnet_id se llama igual que la variable del módulo de **vm** (subnet_id).

En la carpeta main.tf de la carpeta “módulos” vamos a crear la sección de módulos cargando las carpetas contenidas



Luego ejecutar los comandos de terraform, si se agregan nuevos módulos se tiene que ejecutar nuevamente terraform init

El archivo con los valores de las variables puede estar en el módulo general, y este compartirse con todos los módulos hijos.

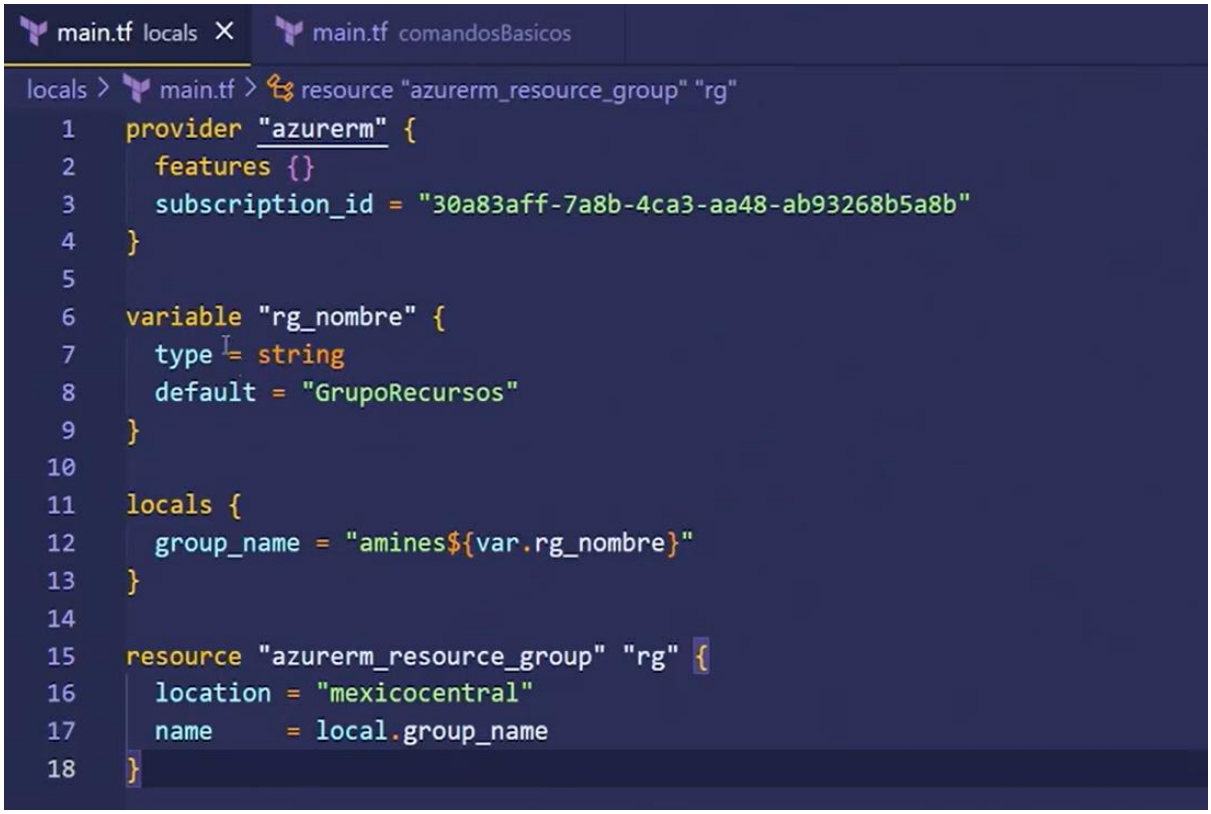


Locals

Le agrega un prefijo a nuestras variables es decir si:

- Local: amines
- Resource Group:GrupoPlatzi
- Resultante: aminesGrupoPlatzi

Se debe de definir un local por cada recurso.



Importar recursos en terraform con Import

Para importar un recurso de la nube (Azure) es necesario obtener su id, como se muestra en la imagen, una forma fácil de obtener este id es ejecutar terraform apply y en el mensaje de error te dirá cual es este id.
Esta acción descarga todas las propiedades del recurso en local y además lo agrega en el state para poder modificarlo a voluntad usando terraform

main.tf data | main.tf import | terraform.tfstate

import > main.tf > resource "azurerm_storage_account" "mi_cuenta"

```
provider "azurerm" {
  features {}
  subscription_id = "30a83aff-7a8b-4ca3-aa48-ab93268b5a8b"
}

import {
  to = azurerm_storage_account.mi_cuenta
  id = "/subscriptions/30a83aff-7a8b-4ca3-aa48-ab93268b5a8b/resourceGroups/logsGroup/providers
}

resource "azurerm_storage_account" "mi_cuenta" {
  name                     = "logstorageaccount23"
  resource_group_name      = "logsGroup"
  location                 = "brazilsouth"
  account_tier             = "Standard"
  account_replication_type = "LRS"
}
```

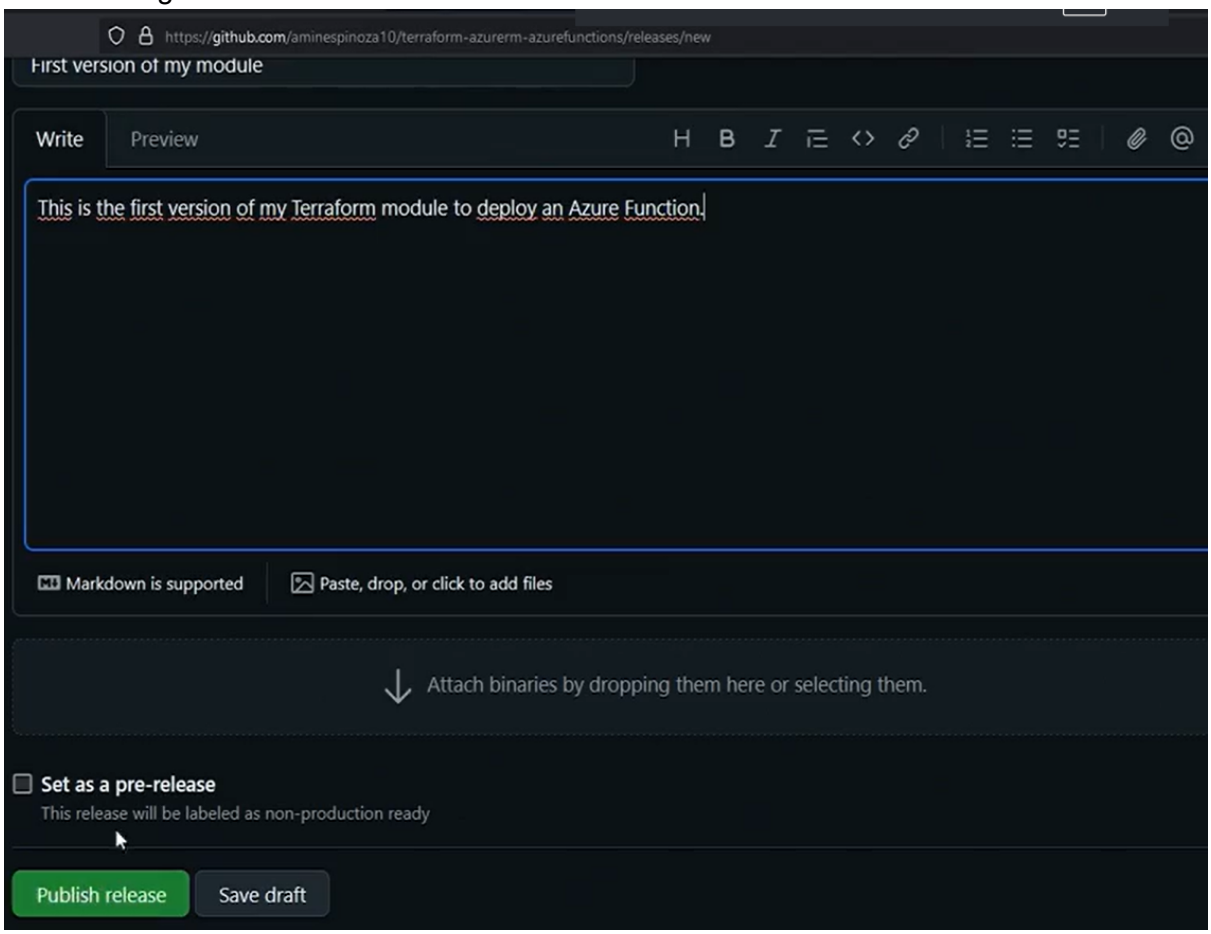
Diferencias entre Import y Data en terraform

Data: solo permite la lectura del recurso y obtener variables de ella, pero lo agrega al state de forma limitada y no permite modificarlo.

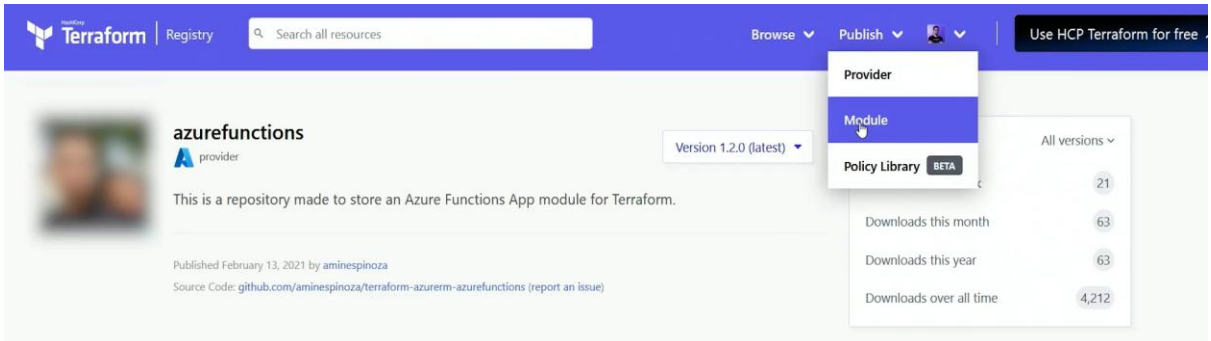
Import: Permite la modificación del recurso y lo agrega al state para ser alterado.

Publicar módulos en Terraform

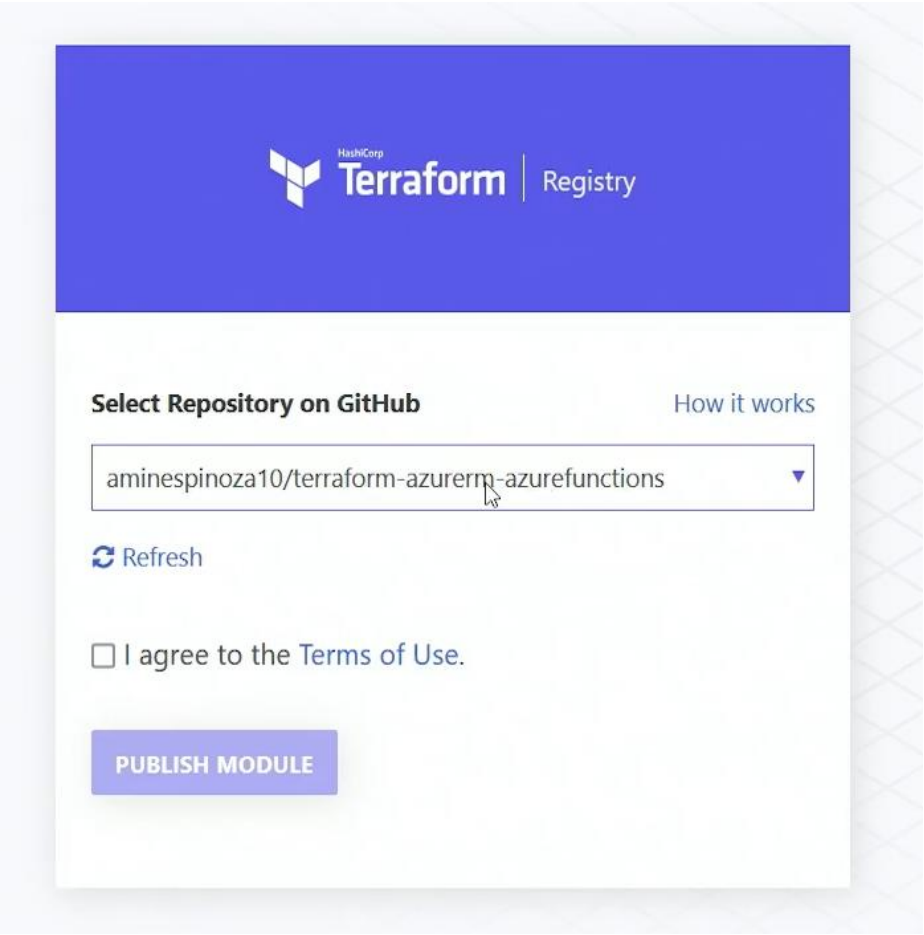
Para esto es necesario crear un repositorio en github de manera pública con la siguiente nomenclatura: **terraform-azurerm-(nombre del modulo)**
En este archivo se debe incluir los archivos de terraform como: main.tf, output.tf, variables.tf, también considerar el Readme y la License .
Al subirlo a github se debe crear una release



Luego dirigirse a terraform registry para subir el modulo



Luego deberá seleccionar el nombre del repositorio a subir



CI/CD con Terraform y Github Actions

Es necesario crear primero un permiso para github action pueda manipular Azure

```
az ad sp create-for-rbac --name "terraformActions" --role contributor --scopes
"/subscriptions/408d3e15-3013-4aec-856d-830eaf9972d1"
```

Se obtiene lo siguiente:

```
{
  "appId": "ea7d02b9-c02f-40ec-9b18-29dd4ec35462",
  "displayName": "terraformActions",
  "password": "0p_8Q~TR3TaeYnOPoSFU542D7t1g2_ghx3xRgccP",
  "tenant": "36df6968-506c-497e-b068-4b0c71cdada9"
}
```

A partir de estos datos hay que dirigirse a Github

Settings -> Security -> Secrets and variables -> Actions -> Repository secrets -> New repository secret

CLIENT_ID -> appId

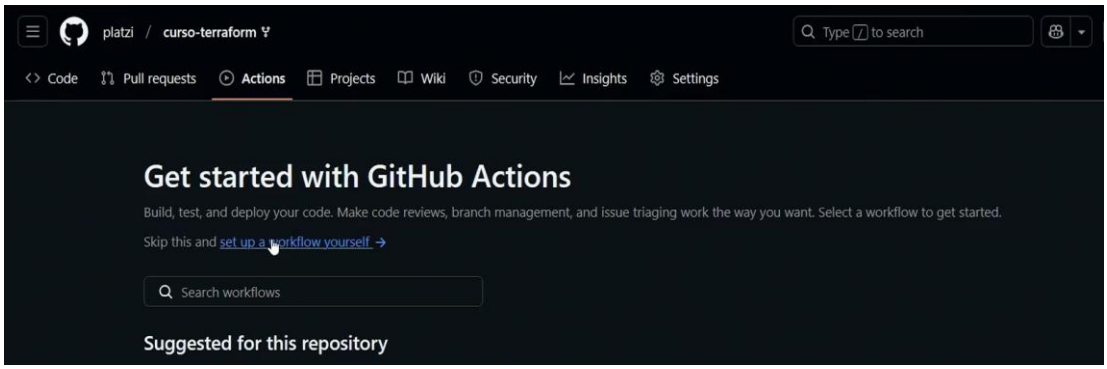
CLIENT_SECRET -> password

SUBSCRIPTION_ID -> subscription_id

TENANT_ID -> tenant

Dirigirse a Actions

Set up a workflow yourself



Crear el archivo de despliegue (terraformDeploy.yml)

```yaml
name: "Terraform deploy"

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  terraform:
    env:
      ARM_CLIENT_ID: ${{ secrets.CLIENT_ID }}
      ARM_CLIENT_SECRET: ${{ secrets.CLIENT_SECRET }}
      ARM_SUBSCRIPTION_ID: ${{ secrets.SUBSCRIPTION_ID }}
      ARM_TENANT_ID: ${{ secrets.TENANT_ID }}
    name: 'Terraform'
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: ${{ github.workspace }}/cicd
    steps:
      - uses: actions/checkout@v2
      - uses: hashicorp/setup-terraform@v1
```

```yaml
    - name: Terraform init
      id: init
      run: terraform init

    - name: Terraform validate
      id: validate
      run: terraform validate -no-color

    - name: Terraform plan
      id: plan
      run: terraform plan -out plan.out

    - name: Terraform apply
      id: apply
      run: terraform apply plan.out
```

Archivo terraform para desplegar

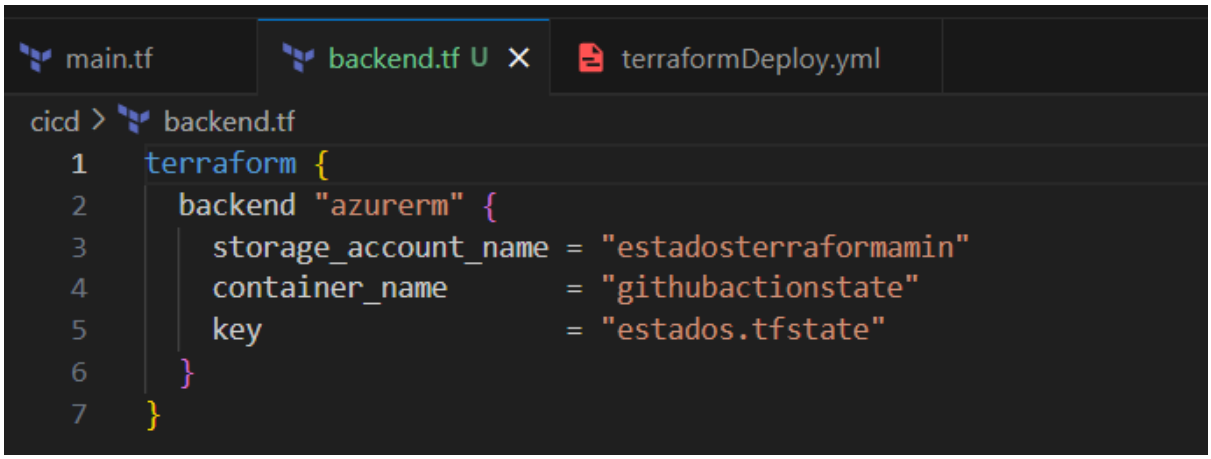
main.tf | terraformDeploy.yml

cicd > main.tf

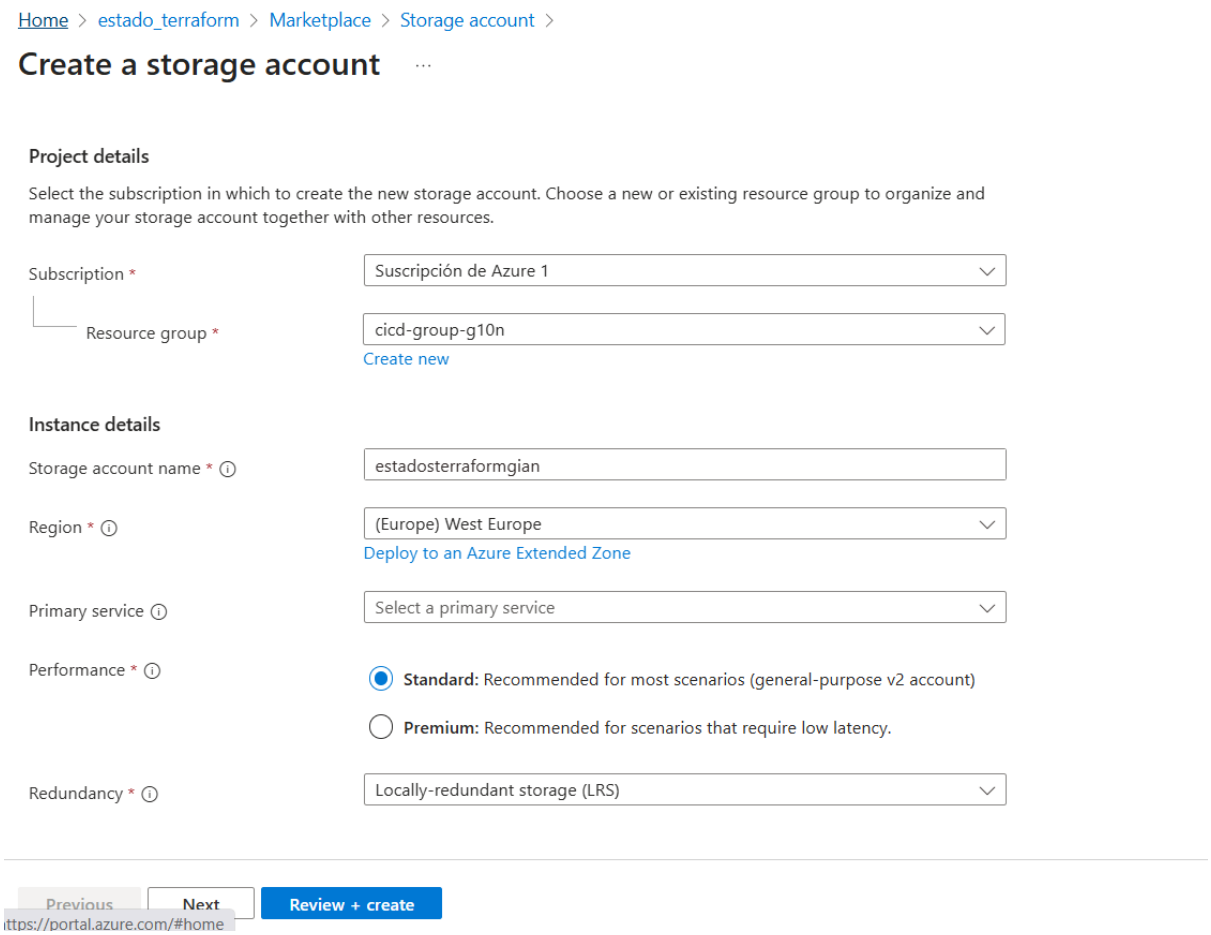
```hcl
provider "azurerm" {
  features {}
  subscription_id = "408d3e15-3013-4aec-856d-830eaf9972d1"
}

resource "azurerm_resource_group" "rg" {
  name     = "cicd-group-g10n"
  location = "eastus2"
}

resource "azurerm_storage_account" "storage" {
  name                     = "continousdeploymentg10n"
  resource_group_name      = azurerm_resource_group.rg.name
  location                 = azurerm_resource_group.rg.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
}

resource "azurerm_storage_container" "container_gian" {
  name                  = "platzimedellin"
  storage_account_id    = azurerm_storage_account.storage.id
  container_access_type = "container"
}

resource "azurerm_storage_container" "container_platzi" {
  name                  = "gianplatzi"
  storage_account_id    = azurerm_storage_account.storage.id
  container_access_type = "container"
}

```

Estado Terraform con Github Actions

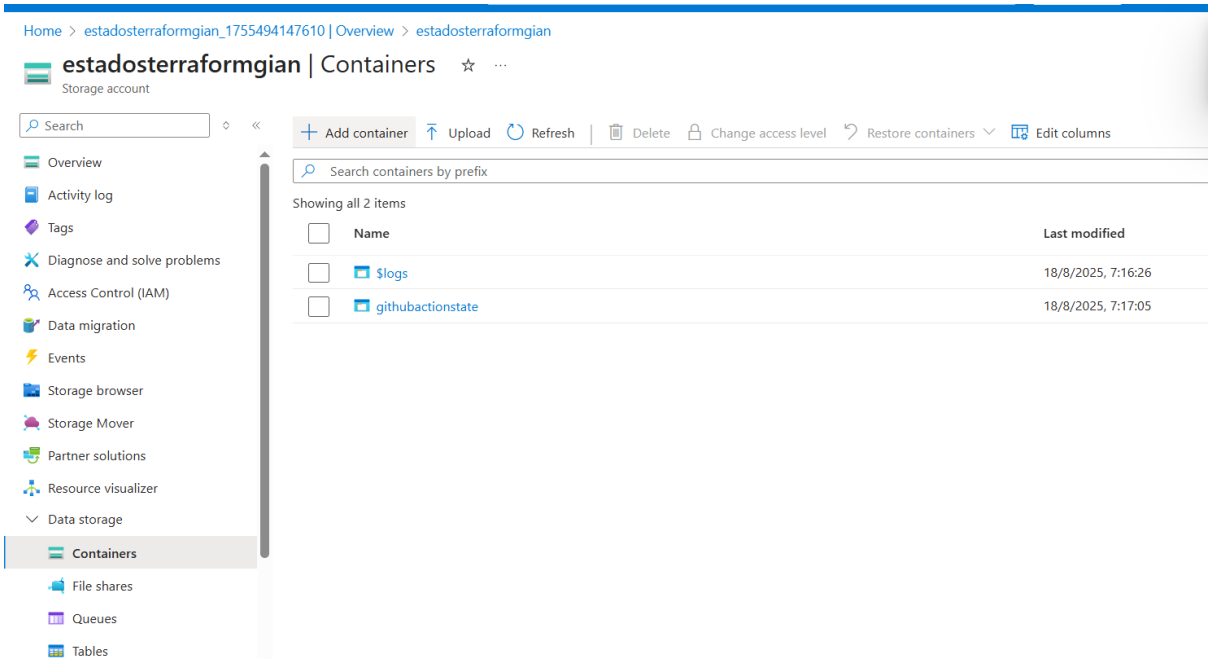
Crear archivo backend.tf



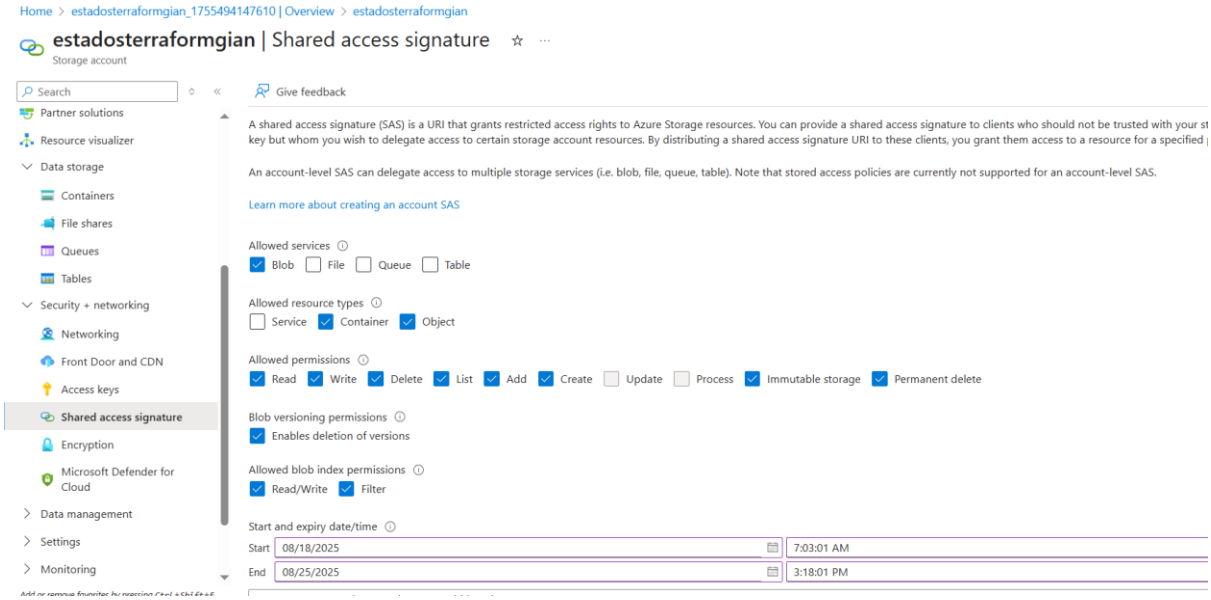
Crear storage account



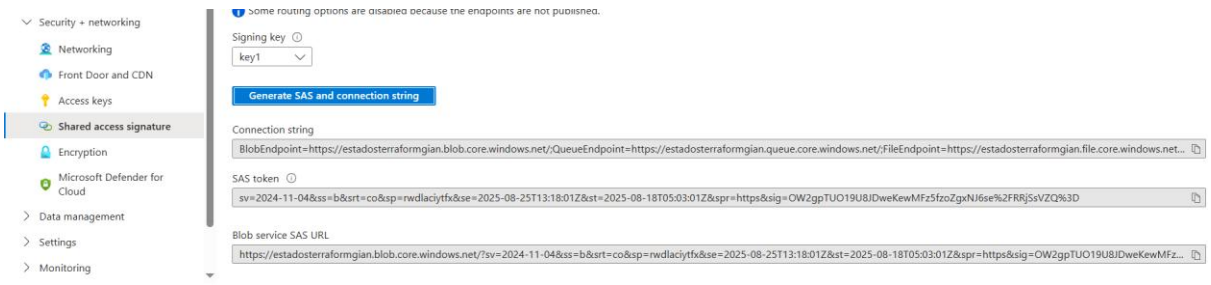
Crear contenedor



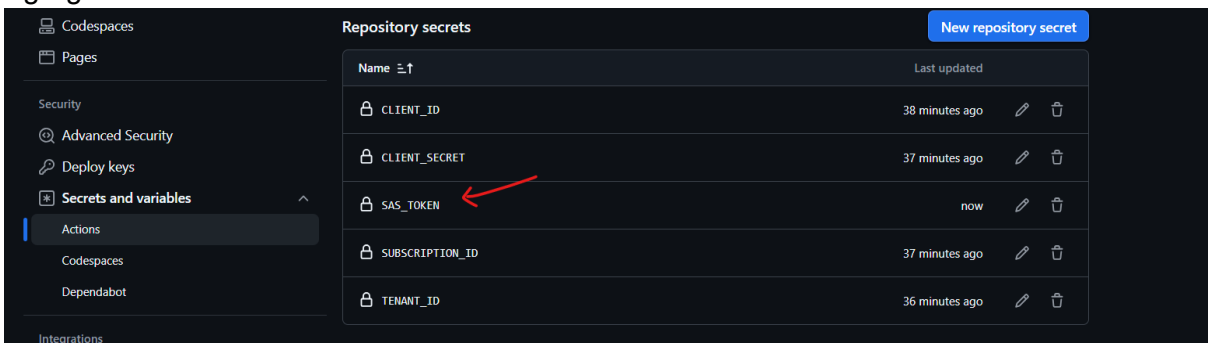
Crear un SAS token (cambiar la fecha End)



Guardar y copiar el SAS token



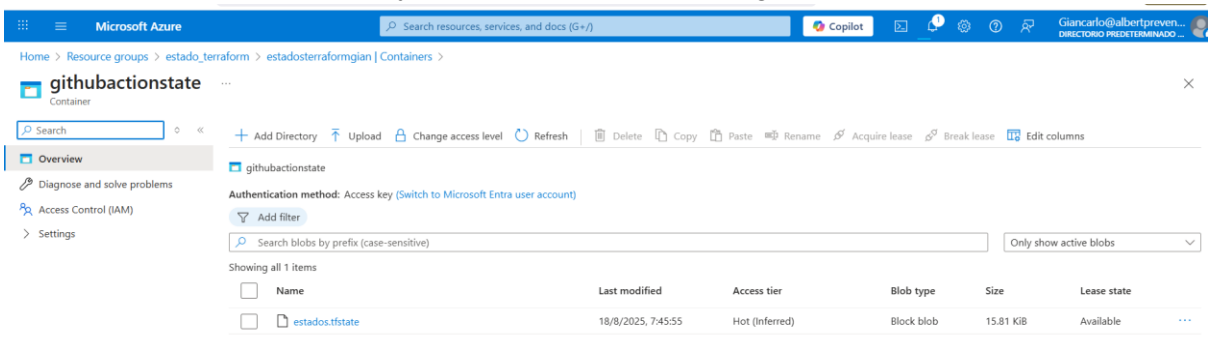
Agregar SAS token en los secretos de Github como se hizo anteriormente



Agregar la siguiente línea en el archivo yml

```
20        working-directory: ${{ github.workspace }}/cicd
21    steps:
22      - uses: actions/checkout@v2
23      - uses: hashicorp/setup-terraform@v1
24
25      - name: Terraform init
26        id: init
27+       run: terraform init -backend-config="sas_token=${{ secrets.SAS_TOKEN }}"
28+
29      - name: Terraform validate
30        id: validate
```

Hacer commit de los cambios y verificar que el estado se guarde en Azure



Si necesitas autenticarte en Azure desde la terminal y el navegador no se abre automáticamente, ¿qué comando alternativo podrías usar?
Elige la respuesta correcta
az login --use-device-code