

3º Básico Terraform

Tarea 1: Instalación de Terraform y primer script con LocalStack

Objetivo: Instalar Terraform y LocalStack. Crear un recurso básico usando Terraform conectado a LocalStack.

Paso 1: Instalar Terraform

Terraform es la herramienta que usaremos para definir nuestra infraestructura como código.

1. Actualizamos nuestros paquetes e instalamos dependencias:

```
sudo apt install -y curl gnupg software-properties-common
```

2. Añadimos la clave GPG de HashiCorp:

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
```

3. Añadimos el repositorio oficial de HashiCorp:

```
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
```

4. Actualizamos la lista de paquetes e instalar Terraform:

```
sudo apt update && sudo apt install terraform -y
```

```
salva@salva-ubuntu:~$ sudo apt install terraform -y
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Los paquetes indicados a continuación se instalaron de forma automática y ya no son necesarios.
  apturl apturl-common gir1.2-gst-plugins-base-1.0 gir1.2-rb-3.0 libcdio-cdda2
  libcdio-paranoia2 libdmapsharing-3.0-2 libgsoap-2.8.117 liblirc-client0 liblzfl
  libmtp-common libmtp-runtime libmtp9 libnfs13 libqt5opengl5 libqt5printsupport5
  libqt5x11extras5 libsgutils2-2 libzip4 python3-mako python3-markupsafe
Utilice «sudo apt autoremove» para eliminarlos.
Se instalarán los siguientes paquetes NUEVOS:
  terraform
0 actualizados, 1 nuevos se instalarán, 0 para eliminar y 2 no actualizados.
Se necesita descargar 27,6 MB de archivos.
Se utilizarán 90,6 MB de espacio de disco adicional después de esta operación.
Des:1 https://apt.releases.hashicorp.com jammy/main amd64 terraform amd64 1.11.4-1 [27,6 MB]
Descargados 27,6 MB en 3s (9.903 kB/s)
Seleccionando el paquete terraform previamente no seleccionado.
(Leyendo la base de datos ... 243293 ficheros o directorios instalados actualmente.)
Preparando para desempaquetar .../terraform_1.11.4-1_amd64.deb ...
Desempaquetando terraform (1.11.4-1) ...
Configurando terraform (1.11.4-1) ...
```

5. Verificamos la instalación:

terraform -v

```
salva@salva-ubuntu:~$ terraform -v
Terraform v1.11.4
on linux_amd64
```

Paso 2: Instalar LocalStack

LocalStack nos permite simular servicios de AWS en nuestra máquina local, ideal para desarrollo y pruebas sin incurrir en costes de AWS.

Ejecutamos LocalStack en un contenedor Docker: Usaremos la imagen oficial de LocalStack.

docker run --rm -d -p 4566:4566 -p 4510-4559:4510-4559 localstack/localstack

```
Digest: sha256:c768c428b8cb369936765bdbb3aa30f31fbd4bb166df36af7d9ede30392532f8
Status: Downloaded newer image for localstack/localstack:latest
27b305b37ac1c6e42edb319b5c618615626bd7e66228c1a6ba6e4d8961e64cb1
```

Verificamos que LocalStack está corriendo con ***docker ps***.

```
salva@salva-ubuntu:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
27b305b37ac1	localstack/localstack	"docker-entrypoint.sh"	5 minutes ago	Up 5 minutes (healthy)
e6801effa9a1	portainer/portainer-ce	"/portainer"	9 days ago	Up 2 hours

Paso 3: Configurar AWS CLI para LocalStack

Antes de crear nuestro script, configuramos **AWS CLI** para interactuar con LocalStack:

Instalamos **AWS CLI**:

`sudo apt install -y awscli`

```
Procesando disparadores para gnome-menus (3.36.0-1ubuntu3) ...
Procesando disparadores para libc-bin (2.35-0ubuntu3.9) ...
Configurando awscli (1.22.34-1) ...
```

Configuramos un perfil para LocalStack:

`aws configure --profile localstack`

Introducimos cualquier valor para **Access Key** y **Secret Key** (LocalStack no los valida), y establecemos la región como **us-east-1** y el **output format** como **json**.

Verificamos la conexión:

`aws --endpoint-url=http://localhost:4566 s3 ls --profile localstack`

```
salva@salva-ubuntu:~$ aws configure --profile localstack
AWS Access Key ID [None]: admin
AWS Secret Access Key [None]: admin
Default region name [us-east-1]: us-east-1
Default output format [json]: json
salva@salva-ubuntu:~$ aws --endpoint-url=http://localhost:4566 s3 ls --profile localstack
salva@salva-ubuntu:~$
```

Paso 4: Crear nuestro primer script de Terraform

Ahora vamos a crear nuestro archivo de configuración de Terraform:

1. Creamos un directorio para nuestro proyecto:

`mkdir terraform-localstack && cd terraform-localstack`

2. Creamos el archivo **main.tf**:

`nano main.tf`

3. Agregamos el siguiente contenido:

```
main.tf
~/Proyectos/terraform-localstack

1 terraform {
2   required_providers {
3     aws = {
4       source = "hashicorp/aws"
5       version = "~> 4.0"
6     }
7   }
8 }
9
10 provider "aws" {
11   region = "us-east-1"
12   access_key = "admin" # Credenciales para LocalStack
13   secret_key = "admin" # Credenciales para LocalStack
14   skip_credentials_validation = true
15   skip_metadata_api_check = true
16   skip_requesting_account_id = true
17   s3_use_path_style = true # Importante para LocalStack
18
19   endpoints {
20     s3 = "http://localhost:4566"
21   }
22 }
23
24 resource "aws_s3_bucket" "mi_primer_bucket" {
25   bucket = "mi-bucket-prueba-${formatdate("YYYYMMDD", timestamp())}"
26   force_destroy = true # Permite borrar el bucket aunque no esté vacío
27 }
28
29 output "bucket_name" {
30   value = aws_s3_bucket.mi_primer_bucket.bucket
31 }
```

Paso 5: Ejecutar nuestro script de Terraform

Ahora vamos a inicializar y aplicar nuestra configuración:

1. Inicializamos Terraform:

terraform init

```
salva@salva-ubuntu:~/Proyectos/terraform-localstack$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 4.0"...
- Installing hashicorp/aws v4.67.0...
- Installed hashicorp/aws v4.67.0 (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.
```

2. Aplicamos la configuración:

terraform apply

3. Confirmamos la acción escribiendo "yes" cuando nos lo solicite.

```
salva@salva-ubuntu: ~/Proyectos/terraform-localstack
+ server_side_encryption_configuration (known after apply)
+ versioning (known after apply)
+ website (known after apply)
}

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

Changes to Outputs:
+ bucket_name = (known after apply)

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

aws_s3_bucket.mi_primer_bucket: Creating...
aws_s3_bucket.mi_primer_bucket: Creation complete after 0s [id=mi-bucket-prueba-20250410]

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

Outputs:

bucket_name = "mi-bucket-prueba-20250410"
```

Paso 6: Verificar el recurso creado

Finalmente, verificamos que nuestro **bucket S3** se ha creado correctamente en **LocalStack**:

Usando **AWS CLI**:

aws --endpoint-url=http://localhost:4566 s3 ls --profile localstack

```
salva@salva-ubuntu:~/Proyectos/terraform-localstack$ aws --endpoint-url=http://localhost:4566 s3 ls --profile localstack
2025-04-10 11:55:50 mi-bucket-prueba-20250410
```

También podemos ver el nombre del bucket en los outputs de Terraform:

terraform output bucket_name

```
salva@salva-ubuntu:~/Proyectos/terraform-localstack$ terraform output bucket_name
"mi-bucket-prueba-20250410"
```

Completada nuestra primera configuración de **Terraform** con **LocalStack**.

Tarea 2: Crear un recurso con Terraform usando LocalStack

Objetivo: Crear un recurso simulado (elegiré una instancia EC2, ya que ya tenemos un bucket S3) en LocalStack usando Terraform.

Ya tenemos gran parte del trabajo hecho con nuestro archivo **main.tf**. Ahora lo modificaremos para añadir una instancia **EC2** simulada.

Paso 1: Modificar main.tf para añadir la instancia EC2 y su endpoint

1. **Abrimos nuestro archivo main.tf** con el editor de texto.
2. **Actualizamos el bloque provider:** Necesitamos decirle a Terraform que use LocalStack también para el servicio EC2. Añadimos la línea `ec2 = "http://localhost:4566"` dentro del bloque `endpoints`.
3. **Añadimos el bloque resource "aws_instance":** Añadiremos este nuevo bloque al final del archivo para definir nuestra instancia EC2 simulada.



```
1 terraform {
2   required_providers {
3     aws = {
4       source  = "hashicorp/aws"
5       version = "~> 4.0" # Mantenemos la versión compatible que ya usábamos
6     }
7   }
8 }
9
10 provider "aws" {
11   region                = "us-east-1"
12   access_key            = "admin"      # Credenciales para LocalStack
13   secret_key            = "admin"      # Credenciales para LocalStack
14   skip_credentials_validation = true
15   skip_metadata_api_check  = true
16   skip_requesting_account_id = true
17   s3_use_path_style        = true      # Importante para LocalStack S3
18
19   # ACTUALIZADO: Añadimos el endpoint para EC2
20   endpoints {
21     s3 = "http://localhost:4566"
22     ec2 = "http://localhost:4566" # <--- NUEVA LÍNEA
23   }
24 }
25
26 # Mantenemos nuestro recurso S3 bucket
27 resource "aws_s3_bucket" "mi_primer_bucket" {
28   bucket = "mi-bucket-prueba-${formatdate("YYYYMMDD", timestamp())}"
29   force_destroy = true # Permite borrar el bucket aunque no esté vacío
30 }
31
32 # NUEVO: Definimos una instancia EC2 simulada
33 resource "aws_instance" "mi_instancia_local" {
34   ami = "ami-12345678" # Usamos un ID de AMI placeholder, LocalStack no la usa realmente
35   instance_type = "t2.micro" # El tipo de instancia también es simbólico en LocalStack
36
37   tags = {
38     Name = "MiInstanciaTerraformLocal"
39   }
40 }
41
42 # Mantenemos nuestro output para el bucket
43 output "bucket_name" {
44   value = aws_s3_bucket.mi_primer_bucket.bucket
45 }
46
47 # NUEVO: Añadimos un output para el ID de la instancia EC2
48 output "instance_id" {
49   value = aws_instance.mi_instancia_local.id
50 }
```

Ejecutamos **terraform init**.

```
salva@salva-ubuntu:~/Proyectos/terraform-localstack$ terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.67.0

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.
```

Ejecutamos **terraform apply**.

```
Plan: 2 to add, 0 to change, 1 to destroy.

Changes to Outputs:
  ~ bucket_name = "mi-bucket-prueba-20250410" -> (known after apply)
  + instance_id = (known after apply)

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

aws_s3_bucket.mi_primer_bucket: Destroying... [id=mi-bucket-prueba-20250410]
aws_instance.mi_instancia_local: Creating...
aws_s3_bucket.mi_primer_bucket: Destruction complete after 0s
aws_s3_bucket.mi_primer_bucket: Creating...
aws_s3_bucket.mi_primer_bucket: Creation complete after 0s [id=mi-bucket-prueba-20250410]
aws_instance.mi_instancia_local: Still creating... [10s elapsed]
aws_instance.mi_instancia_local: Creation complete after 12s [id=i-37625e9717cbd449b]

Apply complete! Resources: 2 added, 0 changed, 1 destroyed.

Outputs:

bucket_name = "mi-bucket-prueba-20250410"
instance_id = "i-37625e9717cbd449b"
```

Verificar la instancia EC2 con AWS CLI

Vamos a usar la AWS CLI para confirmar que nuestra instancia EC2 simulada existe en LocalStack.

1. Ejecutamos el comando **describe-instances**, apuntando al endpoint EC2 de LocalStack:

```
aws --endpoint-url=http://localhost:4566 ec2 describe-instances --profile localstack
```

```

1 {
2   "Reservations": [
3     {
4       "Groups": [],
5       "Instances": [
6         {
7           "AmiLaunchIndex": 0,
8           "ImageId": "ami-12345678",
9           "InstanceId": "i-37625e9717cbd449b",
10          "InstanceType": "t2.micro",
11          "KernelId": "None",
12          "LaunchTime": "2025-04-10T11:31:02Z",
13          "Monitoring": {
14            "State": "disabled"
15          },
16          "Placement": {
17            "AvailabilityZone": "us-east-1a",
18            "GroupName": "",
19            "Tenancy": "default"
20          },
21          "PrivateDnsName": "ip-10-98-82-199.ec2.internal",
22          "PrivateIpAddress": "10.98.82.199",
23          "PublicDnsName": "ec2-54-214-18-223.compute-1.amazonaws.com",
24          "PublicIpAddress": "54.214.18.223",
25          "State": {
26            "Code": 16,
27            "Name": "running"
28          },
29          "StateTransitionReason": "",
30          "SubnetId": "subnet-1d2f21028c6169336",
31          "VpcId": "vpc-b8d9bb6ca1fac1779",
32          "Architecture": "x86_64",
33          "BlockDeviceMappings": [
34            {
35              "DeviceName": "/dev/sda1",
36              "Ebs": {
37                "AttachTime": "2025-04-10T11:31:02Z",
38                "DeleteOnTermination": true,
39                "Status": "in-use",
40                "VolumeId": "vol-a7c3ac2b50f86a000"
41              }
42            }
43          ],
44          "ClientToken": "ABCDE0000000000003",
45          "EbsOptimized": false,
46          "Hypervisor": "xen",
47          "NetworkInterfaces": [
48            {
49              "Association": {
50                "IpOwnerId": "000000000000",
51                "PublicIp": "54.214.18.223"
52              },
53              "Attachment": {
54                "AttachTime": "2015-01-01T00:00:00Z",
55                "AttachmentId": "eni-attach-cfc493a34e4132ce9",
56                "DeleteOnTermination": true,
57                "DeviceIndex": 0
58              }
59            }
60          ]
61        }
62      ]
63    }
64  ]
65 }

```

JSON ~ Anchura del tabulador: 8 ~ Ln 112, Col 1 ~ INS

Esto nos da una salida JSON con todos los detalles de las instancias simuladas.

2. Para una salida más limpia con `--query`

`aws --endpoint-url=http://localhost:4566 ec2 describe-instances --query "Reservations[].Instances[].InstanceId, InstanceType, Tags[?Key=='Name'].Value | [0]" --output table --profile localstack`

```

salva@salva-ubuntu:~/Proyectos/terraform-localstack$ aws --endpoint-url=http://localhost:4566 ec2 describe-instances --query "Reservations[].Instances[].InstanceId, InstanceType, Tags[?Key=='Name'].Value | [0]" --output table --profile localstack
-----
|                               DescribeInstances                               |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| i-37625e9717cbd449b | t2.micro | MiInstanciaTerraformLocal |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

Vemos nuestra instancia en la salida de la AWS CLI.

Tarea 3: Modificar un recurso usando Terraform y LocalStack

Objetivo: Practicar cómo Terraform gestiona las modificaciones en recursos ya existentes, interactuando con LocalStack.

Instrucciones:

Partiremos de nuestro archivo **main.tf** (con el bucket S3 y la instancia EC2).

Paso 1: Modificar main.tf para cambiar el tipo de instancia

1. Abrimos nuestro archivo **main.tf**.

2. Localizamos el bloque del recurso **aws_instance**:

Terraform

```
resource "aws_instance" "mi_instancia_local" {
  ami      = "ami-12345678"
  instance_type = "t2.micro" # <--- VAMOS A CAMBIAR ESTA LÍNEA

  tags = {
    Name = "Instancia-Simulada-LocalStack" # O el nombre que le pusimos
  }
}
```

3. **Cambiamos el valor de instance_type:** Lo modificaremos de "t2.micro" a "t2.small".

Terraform

```
resource "aws_instance" "mi_instancia_local" {
  ami      = "ami-12345678"
  instance_type = "t2.small" # <--- LÍNEA MODIFICADA

  tags = {
    Name = "Instancia-Simulada-LocalStack"
  }
}
```

```
32 # NUEVO: Definimos una instancia EC2 simulada
33 resource "aws_instance" "mi_instancia_local" {
34   ami      = "ami-12345678" # Usamos un ID de AMI placeholder, LocalStack no la usa
    realmente
35   instance_type = "t2.small" # <--- LÍNEA MODIFICADA
36
37   tags = {
38     Name = "MiInstanciaTerraformLocal"
39   }
40 }
```

Aunque LocalStack no cambia el rendimiento real, este cambio simula una modificación común en AWS.

4. **Guardamos y cerramos** el archivo *main.tf*.

Paso 2: Ejecutar *terraform plan* para previsualizar

Antes de aplicar cualquier cambio, siempre es una buena práctica ejecutar *terraform plan*. Este comando nos mostrará qué cambios detecta Terraform entre nuestra configuración y el estado actual registrado, y qué acciones propone realizar.

terraform plan

```
salva@salva-ubuntu:~/Proyectos/terraform-localstack$ terraform plan
aws_s3_bucket.mi_primer_bucket: Refreshing state... [id=mi-bucket-prueba-20250410]
aws_instance.mi_instancia_local: Refreshing state... [id=i-37625e9717cbd449b]

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the following symbols:
  ~ update in-place
  -/+ destroy and then create replacement

Terraform will perform the following actions:

# aws_instance.mi_instancia_local will be updated in-place
~ resource "aws_instance" "mi_instancia_local" {
  id           = "i-37625e9717cbd449b"
  ~ instance_type = "t2.micro" -> "t2.small"
  tags         = {
    "Name" = "MiInstanciaTerraformLocal"
  }
  # (34 unchanged attributes hidden)

  # (1 unchanged block hidden)
}
```

Paso 3: Ejecutar *terraform apply* para aplicar el cambio

Una vez revisado el plan y estando de acuerdo con los cambios propuestos, aplicamos la modificación.

1. Ejecutamos *terraform apply*
2. Terraform nos mostrará el plan de nuevo y pedirá confirmación. Escribimos *yes* y pulsamos *Enter*.
3. Terraform contactará con LocalStack para realizar la modificación en la instancia EC2 simulada.

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

Outputs:

bucket_name = "mi-bucket-prueba-20250410"
instance_id = "i-37625e9717cbd449b"
```

Paso 4: Verificar la modificación con AWS CLI

Comprobamos la modificación usando AWS CLI con LocalStack.

aws --endpoint-url=http://localhost:4566 ec2 describe-instances --profile localstack

```
salva@salva-ubuntu:~/Proyectos/terraform-localstack$ aws --endpoint-url=http://localhost:4566 ec2 describe-instances --profile localstack
{
  "Reservations": [
    {
      "Groups": [],
      "Instances": [
        {
          "AmiLaunchIndex": 0,
          "ImageId": "ami-12345678",
          "InstanceId": "i-37625e9717cbd449b",
          "InstanceType": "t2.small",
          "KernelId": "None",
          "LaunchTime": "2025-04-10T11:31:02Z",
          "Monitoring": {
            "State": "disabled"
          }
        }
      ]
    }
  ]
}
```

Simplificado para mostrarlo en la tabla:

aws --endpoint-url=http://localhost:4566 ec2 describe-instances --query "Reservations[].Instances[].[InstanceId, InstanceType, Tags[?Key=='Name'].Value | [0]]" --output table --profile localstack

```
salva@salva-ubuntu:~/Proyectos/terraform-localstack$ aws --endpoint-url=http://localhost:4566 ec2 describe-instances --query "Reservations[].Instances[].[InstanceId, InstanceType, Tags[?Key=='Name'].Value | [0]]" --output table --profile localstack
-----
|                               DescribeInstances                               |
+-----+-----+-----+-----+
| i-37625e9717cbd449b | t2.small | MiInstanciaTerraformLocal |
+-----+-----+-----+-----+
```

Hemos modificado un recurso existente usando Terraform y verificado el cambio en nuestro entorno LocalStack.

Tarea 4: Introducción a los módulos de Terraform

Objetivo: Aprender a crear y reutilizar módulos con Terraform, aplicándolo a nuestro entorno LocalStack.

Paso 1: Crear la estructura de directorios del módulo

Dentro de la carpeta del proyecto creamos los directorios:

mkdir -p modules/mi-recurso

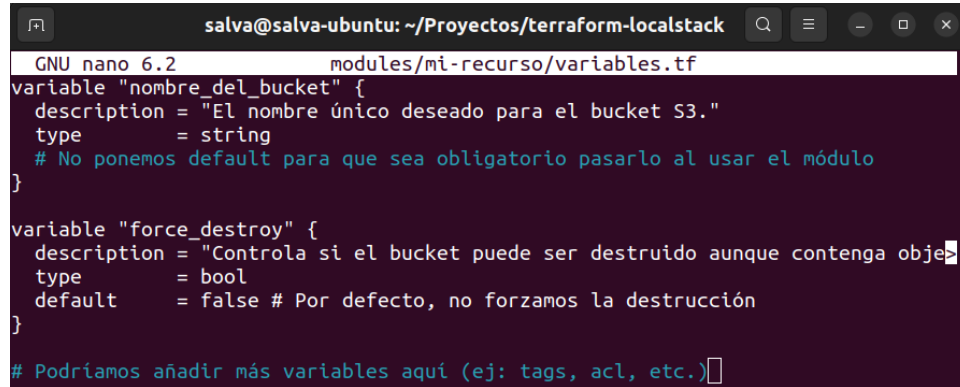
Paso 2: Definir el recurso S3 dentro del módulo

Ahora, dentro del directorio modules/mi-recurso, crearemos los archivos que definen nuestro módulo reutilizable para un bucket S3. Necesitamos al menos tres archivos:

variables.tf (para las entradas), **main.tf** (para los recursos) y **outputs.tf** (para las salidas).

1. **Creamos modules/mi-recurso/variables.tf:** Este archivo define qué parámetros podemos pasarle a nuestro módulo desde fuera.

nano modules/mi-recurso/variables.tf



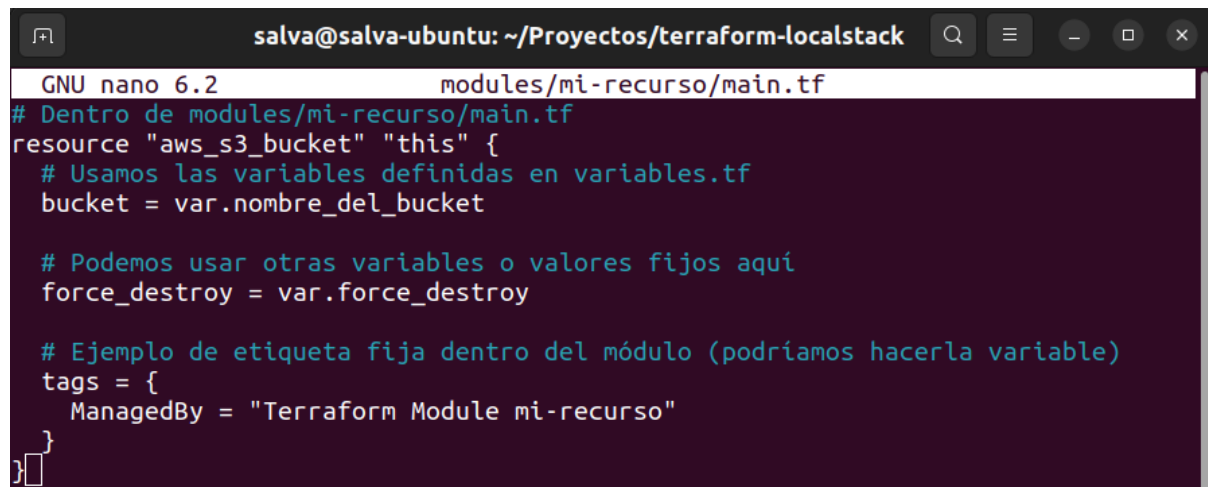
```
salva@salva-ubuntu: ~/Proyectos/terraform-localstack
GNU nano 6.2 modules/mi-recurso/variables.tf
variable "nombre_del_bucket" {
  description = "El nombre único deseado para el bucket S3."
  type        = string
  # No ponemos default para que sea obligatorio pasarlo al usar el módulo
}

variable "force_destroy" {
  description = "Controla si el bucket puede ser destruido aunque contenga obje>
  type        = bool
  default     = false # Por defecto, no forzamos la destrucción
}

# Podríamos añadir más variables aquí (ej: tags, acl, etc.)
```

2. **Creamos modules/mi-recurso/main.tf:** Este es el corazón del módulo, define el recurso S3 usando las variables de entrada.

nano modules/mi-recurso/main.tf



```
salva@salva-ubuntu: ~/Proyectos/terraform-localstack
GNU nano 6.2 modules/mi-recurso/main.tf
# Dentro de modules/mi-recurso/main.tf
resource "aws_s3_bucket" "this" {
  # Usamos las variables definidas en variables.tf
  bucket = var.nombre_del_bucket

  # Podemos usar otras variables o valores fijos aquí
  force_destroy = var.force_destroy

  # Ejemplo de etiqueta fija dentro del módulo (podríamos hacerla variable)
  tags = {
    ManagedBy = "Terraform Module mi-recurso"
  }
}
```

*Nota: Usar **this** como nombre lógico del recurso principal dentro de un módulo es una convención común, pero no obligatoria.*

3. **Creamos modules/mi-recurso/outputs.tf:** Este archivo define qué información devolverá el módulo una vez creado el recurso.

nano modules/mi-recurso/outputs.tf

```
salva@salva-ubuntu: ~/Proyectos/terraform-localstack
GNU nano 6.2 modules/mi-recurso/outputs.tf *
# Dentro de modules/mi-recurso/outputs.tf
output "bucket_id" {
  description = "El nombre (ID) del bucket S3 creado."
  value       = aws_s3_bucket.this.id # Referencia al recurso en main.tf del mó>
}

output "bucket_arn" {
  description = "El ARN (Amazon Resource Name) del bucket S3 creado."
  value       = aws_s3_bucket.this.arn # Referencia al recurso en main.tf del m>
}
```

Con esto, nuestro **módulo mi-recurso** está definido y listo para ser usado.

Paso 3: Usar el módulo en el *main.tf* raíz

Ahora, volvemos a nuestro archivo **main.tf** principal (el que está fuera del directorio modules) y lo modificamos para que use nuestro nuevo módulo en lugar de definir el bucket S3 directamente.

1. Editamos el main.tf raíz:

nano main.tf

2. Modificamos el contenido:

- a. **Eliminamos o comentamos** el bloque resource "aws_s3_bucket" ... original.
- b. **Eliminamos o comentamos** el bloque output "nombre_bucket" ... original que mostraba el nombre del bucket.
- c. **Añadimos un bloque module** para invocar a nuestro módulo.
- d. **Mantenemos** los bloques terraform {}, provider "aws" {} (con la configuración de LocalStack y los endpoints S3 y EC2), y el recurso aws_instance (para no perder ese estado).
- e. **Añadimos un nuevo output** si queremos ver la salida de nuestro módulo.

```
main.tf
~/Proyectos/terraform-localstack
Guardar

1 terraform {
2   required_providers {
3     aws = {
4       source = "hashicorp/aws"
5       version = "~> 5.0" # Actualizada de la 4 a la 5
6     }
7   }
8 }
9
10 # Configuración del proveedor AWS para LocalStack (;IMPORTANTE MANTENER!)
11 provider "aws" {
12   region          = "us-east-1"
13   access_key      = "test"
14   secret_key      = "test"
15   skip_credentials_validation = true
16   skip_metadata_api_check    = true
17   skip_requesting_account_id = true
18   s3_use_path_style          = true
19
20   endpoints {
21     s3 = "http://localhost:4566"
22     ec2 = "http://localhost:4566"
23   }
24 }
25
26 # COMENTAMOS O BORRAMOS EL RECURSO S3 ORIGINAL:
27 # resource "aws_s3_bucket" "mi_bucket_local" { # 0 "mi_primer_bucket", etc.
28 #   bucket = "mi-bucket-localstack-${formatdate("YYYYMMDD", timestamp())}"
29 #   force_destroy = true
30 # }
31
32 # Mantenemos la instancia EC2
33 resource "aws_instance" "mi_instancia_local" {
34   ami           = "ami-12345678"
35   instance_type = "t2.small" # La dejamos como quedó en la Tarea 3
36
37   tags = {
38     Name = "MiInstanciaTerraformLocal" // <-- Volver al valor anterior
39   }
40 }
41
42 # NUEVO: Usamos nuestro módulo para crear el bucket S3
43 module "mi_bucket_s3" {
44   source = "../modules/mi-recurso" # Ruta relativa al directorio del módulo
45
46   # Pasamos los valores para las variables definidas en el módulo
47   nombre_del_bucket = "mi-bucket-modulo-${formatdate("YYYYMMDD", timestamp())}"
48   force_destroy      = true # Sobreescribimos el default del módulo
49 }
50
51 # COMENTAMOS O BORRAMOS EL OUTPUT ORIGINAL DEL BUCKET:
52 # output "nombre_bucket" {
53 #   value = aws_s3_bucket.mi_bucket_local.bucket
54 # }
55
56 # Mantenemos el output de la instancia
57 output "id_instancia" {
58   value = aws_instance.mi_instancia_local.id
59 }
```

Paso 4: Ejecutamos **terraform init** y **terraform apply**

```
salva@salva-ubuntu: ~/Proyectos/terraform-localstack
salva@salva-ubuntu:~/Proyectos/terraform-localstack$ terraform init
Initializing the backend...
Initializing modules...
- mi_bucket_s3 in modules/mi-recurso
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file

Error: Failed to query available provider packages

Could not retrieve the list of available versions for provider hashicorp/aws:
locked provider registry.terraform.io/hashicorp/aws 4.67.0 does not match
configured version constraint ~> 5.0; must use terraform init -upgrade to
allow selection of new versions

To see which modules are currently depending on hashicorp/aws and what
versions are specified, run the following command:
    terraform providers
```

Para permitir que Terraform seleccione una nueva versión del proveedor que cumpla con tu nueva restricción (~> 5.0) y actualice el archivo de bloqueo, debemos usar la opción -upgrade.

terraform init -upgrade

```
salva@salva-ubuntu:~/Proyectos/terraform-localstack$ terraform init -upgrade
Initializing the backend...
Upgrading modules...
- mi_bucket_s3 in modules/mi-recurso
Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.0"...
- Installing hashicorp/aws v5.94.1...
- Installed hashicorp/aws v5.94.1 (signed by HashiCorp)
Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.

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.
```

terraform plan

```
salva@salva-ubuntu: ~/Proyectos/terraform-localstack

+ replication_configuration (known after apply)

+ server_side_encryption_configuration (known after apply)

+ versioning (known after apply)

+ website (known after apply)
}

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

Changes to Outputs:
+ arn_bucket_desde_modulo = (known after apply)
- bucket_name             = "mi-bucket-prueba-20250410" -> null
+ id_instancia            = (known after apply)
- instance_id             = "i-37625e9717cbd449b" -> null
+ nombre_bucket_desde_modulo = (known after apply)

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.
```

terraform apply

```
salva@salva-ubuntu: ~/Proyectos/terraform-localstack
+ nombre_bucket_desde_modulo = (known after apply)

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

module.mi_bucket_s3.aws_s3_bucket.this: Creating...
aws_instance.mi_instancia_local: Creating...
module.mi_bucket_s3.aws_s3_bucket.this: Creation complete after 0s [id=mi-bucket-modulo-20250410]
aws_instance.mi_instancia_local: Still creating... [10s elapsed]
aws_instance.mi_instancia_local: Creation complete after 10s [id=i-b70767200d3be2128]

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

Outputs:

arn_bucket_desde_modulo = "arn:aws:s3:::mi-bucket-modulo-20250410"
id_instancia = "i-b70767200d3be2128"
nombre_bucket_desde_modulo = "mi-bucket-modulo-20250410"
```

Paso 5: Verificar que se haya desplegado correctamente

Para confirmar que nuestro módulo ha funcionado y el bucket S3 se ha creado en LocalStack como esperábamos, vamos a usar de nuevo nuestro comando de confianza de la AWS CLI para listar los buckets.

aws --endpoint-url=http://localhost:4566 s3 ls --profile localstack

```
salva@salva-ubuntu:~/Proyectos/terraform-localstack$ aws --endpoint-url=http://localhost:4566 s3 ls --profile localstack
2025-04-10 21:05:27 mi-bucket-modulo-20250410
```

Hemos creado infraestructura usando un módulo de Terraform apuntando a LocalStack.