

Introducción a Terraform y Salt Project

Unidad 04. Caso práctico 02

Autor: Sergi García

Actualizado Noviembre 2025



Licencia



Reconocimiento - No comercial - CompartirIgual (BY-NC-SA): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se ha de hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán diferentes símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

Importante

Atención

Interesante

ÍNDICE

Proyecto: Infraestructura local dockerizada con Terraform - Balanceo de carga	3
1. Explicación: ¿Qué haremos en este caso práctico?	3
2. Objetivo	3
3. Arquitectura	3
4. Estructura del Proyecto	3
5. Ejecución paso a paso	4
6. Detalle del contenido de los ficheros	6
variables.tf	6
main.tf	7
outputs.tf	9
templates/index.html.tpl	9
templates/nginx.conf.tpl	9
terraform.tfvars	10

UNIDAD 04 - CASO PRÁCTICO 02

 PROYECTO: INFRAESTRUCTURA LOCAL DOCKERIZADA CON TERRAFORM - BALANCEO DE CARGA

1. EXPLICACIÓN: ¿QUÉ HAREMOS EN ESTE CASO PRÁCTICO?

En este segundo caso práctico daremos un paso más en el uso de **Terraform** para gestionar infraestructura Docker, añadiendo un **balanceador de carga Nginx** que distribuirá las peticiones entre varios contenedores web.

Terraform se encargará de **crear automáticamente**:

- Una **red Docker** común para conectar todos los contenedores.
- Varios **servidores NGINX backend** (cada uno con un nombre aleatorio y su propia página HTML personalizada, generada mediante `templatefile()`).
- Un **contenedor adicional** que actuará como **reverse proxy o balanceador de carga**, usando una configuración de Nginx generada dinámicamente con Terraform.

Gracias a este enfoque, la infraestructura será **completamente reproducible y escalable**: bastará modificar una variable para cambiar el número de servidores web y Terraform ajustará el balanceo sin intervención manual.

El resultado será un entorno local accesible desde `http://localhost:8888`, donde Nginx repartirá las peticiones entre los distintos contenedores, demostrando cómo Terraform puede **automatizar no solo la creación de contenedores, sino también su interconexión y configuración coordinada**.

2. OBJETIVO

Crear una infraestructura local compuesta por varios contenedores Docker usando Terraform, todos interconectados por una red Docker, cada uno con:

- Nombres aleatorios y únicos.
- Archivos HTML personalizados generados con `templatefile()`.
- Variables para escalar el número de contenedores web.
- Balanceo de carga entre los N contenedores escalados con Nginx Proxy.

3. ARQUITECTURA

 Escalable a N contenedores dinámicamente y balanceando carga entre ellos.

 Todos interconectados por red Docker terraform-net.

 Cada contenedor sirve un archivo HTML personalizado y generado por Terraform.

4. ESTRUCTURA DEL PROYECTO

UD04-Caso02/

```
└── main.tf
└── variables.tf
└── outputs.tf
└── terraform.tfvars
└── templates/
```

```

|   └── index.html.tpl
|   └── nginx.conf.tpl
└── .gitignore

```

✓ 5. EJECUCIÓN PASO A PASO

1. Inicializa el proyecto:

```
terraform init
```

```

alumno@alumno-virtualbox:~/Desktop/UD04-Caso02$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding kreuzwerker/docker versions matching "~> 3.0"...
- Finding latest version of hashicorp/random...
- Finding latest version of hashicorp/local...
- Installing hashicorp/random v3.7.2...
- Installed hashicorp/random v3.7.2 (signed by HashiCorp)
- Installing hashicorp/local v2.5.3...
- Installed hashicorp/local v2.5.3 (signed by HashiCorp)
- Installing kreuzwerker/docker v3.6.2...
- Installed kreuzwerker/docker v3.6.2 (self-signed, key ID BD080C4571C6104C)

```

2. Aplica la infraestructura:

```
terraform apply
```

```

alumno@alumno-virtualbox:~/Desktop/UD04-Caso02$ terraform apply
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:

# docker_container.nginx_proxy will be created
+ resource "docker_container" "nginx_proxy" [
    + attach                               = false
    + bridge                               = (known after apply)
    + command                             = (known after apply)
    + container_logs                      = (known after apply)
    + container_read_refresh_timeout_milliseconds = 15000
    + endpoint                           = (known after apply)
    + env                                = (known after apply)
    + exit_code                          = (known after apply)
    + hostname                           = (known after apply)
]

```

3. Resultado esperado similar a:

```
Apply complete! Resources: 13 added, 0 changed, 0 destroyed.
```

Outputs:

```

contenedores_web = [
  {
    "archivo" = "/home/alumno/Desktop/UD04-Caso02/site_magical-chimp.html"
    "nombre" = "nginx-magical-chimp"
  },
  {
    "archivo" = "/home/alumno/Desktop/UD04-Caso02/site_enjoyed-liger.html"
    "nombre" = "nginx-enjoyed-liger"
  },
  {

```

```

"archivo" = "/home/alumno/Desktop/UD04-Caso02/site_causal-kangaroo.html"
"nombre" = "nginx-causal-kangaroo"
},
]
url_balanceador = "http://localhost:8888"

```

```

Apply complete! Resources: 13 added, 0 changed, 0 destroyed.

Outputs:

contenedores_web = [
{
  "archivo" = "/home/alumno/Desktop/UD04-Caso02/site_magical-chimp.html"
  "nombre" = "nginx-magical-chimp"
},
{
  "archivo" = "/home/alumno/Desktop/UD04-Caso02/site_enjoyed-liger.html"
  "nombre" = "nginx-enjoyed-liger"
},
{
  "archivo" = "/home/alumno/Desktop/UD04-Caso02/site_causal-kangaroo.html"
  "nombre" = "nginx-causal-kangaroo"
},
]
url_balanceador = "http://localhost:8888"

```

4. Podemos verificar otros contenedores con:

```
docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b74a1ad1c0d5	nginx:latest	/docker-entrypoint....	2 hours ago	Up 2 hours	0.0.0.0:8888->80/tcp	nginx-proxy
05f746e4ac31	nginx:latest	/docker-entrypoint....	2 hours ago	Up 2 hours	80/tcp	nginx-causal-kangaroo
6444d884cc3b	nginx:latest	/docker-entrypoint....	2 hours ago	Up 2 hours	80/tcp	nginx-magical-chimp
1658e6b25e2b	nginx:latest	/docker-entrypoint....	2 hours ago	Up 2 hours	80/tcp	nginx-enjoyed-liger

5. Accede al navegador y comprobar que sirve y que a cada cambio se balancee la carga:

<http://localhost:8888>





Aquí el balanceador de Nginx hace que cada petición sea atendida por un contenedor distinto, realizando el balanceo de carga.

6. Una vez verificado su funcionamiento, puedes eliminar todo con:

```
terraform destroy
```

6. DETALLE DEL CONTENIDO DE LOS FICHEROS

variables.tf

```
# Número de contenedores NGINX backend que se desplegarán
variable "web_container_count" {
    description = "Número de contenedores web (nginx) a desplegar"
    type        = number
    default     = 3
}

# Mensaje HTML personalizado que verá el usuario en cada contenedor
variable "html_message" {
    description = "Mensaje HTML que se mostrará en cada contenedor"
    type        = string
    default     = "¡Hola desde Terraform + Docker!"
}

# Imagen base de NGINX. Puede ser "nginx:latest", "nginx:alpine", etc.
variable "base_image" {
    description = "Imagen de contenedor base (debe tener nginx)"
    type        = string
    default     = "nginx:latest"
}

# Puerto local donde se expondrá el balanceador de carga
variable "exposed_port" {
    description = "Puerto local donde se expone el balanceador"
    type        = number
    default     = 8888
}
```

 main.tf

```
# Especificamos que usaremos el provider "docker" (el oficial de Kreuzwerker)
terraform {
    required_providers {
        docker = {
            source  = "kreuzwerker/docker"
            version = "~> 3.0"
        }
    }
}

# Activamos el provider docker Local
provider "docker" {}

# Creamos una red Docker para conectar todos los contenedores
resource "docker_network" "terraform_net" {
    name = "terraform-net"
}

# Usamos nombres aleatorios (tipo "curious-cat") para cada contenedor
resource "random_pet" "web_names" {
    count  = var.web_container_count
    length = 2
}

# Creamos un archivo HTML para cada contenedor, basado en una plantilla
resource "Local_file" "html_files" {
    count      = var.web_container_count

    # Nombre del archivo, usando el nombre aleatorio del contenedor
    filename = "${path.module}/site_${random_pet.web_names[count.index].id}.html"

    # Contenido generado dinámicamente con variables
    content  = templatefile("${path.module}/templates/index.html.tpl", {
        mensaje = var.html_message
        nombre  = random_pet.web_names[count.index].id
    })
}

# Nos aseguramos de tener la imagen NGINX Localmente
resource "docker_image" "nginx" {
    name = var.base_image
}

# Contenedores NGINX que sirven el HTML personalizado
resource "docker_container" "web" {
    count = var.web_container_count

    name  = "nginx-${random_pet.web_names[count.index].id}"
    image = docker_image.nginx.name

    # Conectamos el contenedor a la red compartida
}
```

```
networks_advanced {  
    name = docker_network.terraform_net.name  
}  
  
# Montamos el archivo HTML generado como página principal  
volumes {  
    host_path      = abspath(Local_file.html_files[count.index].filename)  
    container_path = "/usr/share/nginx/html/index.html"  
}  
  
restart = "no" # No reiniciar automáticamente si se detiene  
}  
  
# _____  
# Generar el nginx.conf del proxy  
# _____  
  
# Creamos una lista de líneas "server <nombre>:80;" para el bloque upstream  
locals {  
    backends = [  
        for container in docker_container.web :  
            "server ${container.name}:80;"  
    ]  
}  
  
# Usamos la plantilla nginx.conf.tpl para generar un archivo real  
resource "local_file" "nginx_config" {  
    filename = "${path.module}/nginx.conf"  
    content = templatefile("${path.module}/templates/nginx.conf.tpl", {  
        backends = local.backends  
    })  
}  
  
# Contenedor NGINX que actúa como reverse proxy (balanceador)  
resource "docker_container" "nginx_proxy" {  
    name   = "nginx-proxy"  
    image  = docker_image.nginx.name  
  
    # Exponemos el puerto 80 interno del contenedor como el 8888 en el host  
    ports {  
        internal = 80  
        external = var.exposed_port  
    }  
  
    # Conectado a la misma red Docker  
    networks_advanced {  
        name = docker_network.terraform_net.name  
    }  
  
    # Montamos el archivo de configuración NGINX generado automáticamente  
    volumes {  
        host_path      = abspath(Local_file.nginx_config.filename)  
        container_path = "/etc/nginx/nginx.conf"
```

```

    }

depends_on = [docker_container.web] # Nos aseguramos de que los backends existan
}

```

outputs.tf

```

# Mostramos cada contenedor web creado, su nombre y su HTML
output "contenedores_web" {
  description = "Contenedores web desplegados"
  value = [
    for i in docker_container.web :
    {
      nombre = i.name
      archivo = abspath(tolist(i.volumes)[0].host_path)
    }
  ]
}

# URL donde puedes acceder al balanceador reverse proxy
output "url_balanceador" {
  description = "URL del balanceador reverse proxy"
  value        = "http://localhost:${var.exposed_port}"
}

```

templates/index.html.tpl

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8"> <!-- Evita errores de codificación -->
  <title>Servidor ${nombre}</title> <!-- Nombre del contenedor como título -->
  <style>
    body { font-family: sans-serif; text-align: center; background-color: #f0f0f0;
padding-top: 50px; }
    h1 { color: #333; }
  </style>
</head>
<body>
  <h1>${mensaje}</h1> <!-- Mensaje HTML personalizado -->
  <p>Servidor generado automáticamente con <strong>Terraform</strong> 

```

templates/nginx.conf.tpl

```

# Configuración de eventos (requerido por NGINX aunque vacío aquí)
events {}

```

```
http {  
    # Definimos un grupo de servidores backend que serán balanceados  
    upstream backend {  
        %{ for srv in backends ~}  
            ${srv}      # Ejemplo generado: server nginx-settling-polecat:80;  
        %{ endfor ~}  
    }  
  
    # Definimos el servidor principal que escucha peticiones externas  
    server {  
        listen 80;  
        location / {  
            # Redirige las peticiones al grupo de servidores "backend"  
            proxy_pass http://backend;  
        }  
    }  
}
```

terraform.tfvars

```
# Número de servidores a desplegar  
web_container_count = 3  
# Mensaje HTML que mostrará cada servidor  
html_message      = "¡Esta es una web generada automáticamente!"  
# Puerto único por donde accedemos al balanceador desde el navegador  
exposed_port      = 8888
```