Introducción a Terraform y Salt Project

Unidad 04. Caso práctico 01

Autor: Sergi García

Actualizado Noviembre 2025

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
UNIÓN EUROPEA
Fondo Social Europeo
El FSE invierte en tu futuro
GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura, Universitats i Ocupació
CFR CEFIRE
FORMACIÓ PROFESSIONAL ENSENYANCES ARTÍSTIQUES I ESPORTIVES
FPCV
Formació Professional Comunitat Valenciana

Licencia

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

Unidad 04 - Caso práctico 01

🔥 Proyecto: Infraestructura local dockerizada con Terraform - 1 contenedor expuesto

📘 1. Explicación: ¿Qué haremos en este caso práctico?

En este caso práctico aprenderemos a **crear una infraestructura local con Docker gestionada por Terraform**, aplicando los principios de *infraestructura como código (IaC)*.

Terraform se encargará de **definir y desplegar varios contenedores NGINX** conectados entre sí por una red común. Cada contenedor tendrá un **nombre único generado aleatoriamente** y servirá una **página HTML personalizada**, creada dinámicamente mediante una plantilla (templatefile()).

El sistema será **escalable**, ya que podremos ajustar fácilmente el número de contenedores mediante una variable. Además, solo el **primer contenedor** estará **expuesto al exterior** a través del puerto 8888, lo que permitirá acceder desde el navegador a la página generada.

Con este ejercicio pondremos en práctica:

- El uso del proveedor **Docker de Terraform**.
- La **creación dinámica** de recursos y archivos locales.
- El **montaje automático** de contenido en los contenedores.
- El uso de **variables, outputs y redes personalizadas**.

Este proyecto sentará las bases para el siguiente caso práctico, donde añadiremos **balanceo de carga** y configuración de **reverse proxy** con NGINX.

🎯 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()
- Exposición opcional de puertos
- Variables para escalar el número de contenedores web
- Exposición hacia fuera por **el puerto 8888 del primer contenedor.**
    - **Esto lo mejoraremos en el siguiente caso práctico.**

💡 3. Arquitectura

📦 Escalable a N contenedores dinámicamente, **pero solo uno está expuesto. Esto lo mejoraremos en el siguiente ejemplo práctico.**
🔌 Todos interconectados por red Docker terraform-net.
📝 Cada contenedor sirve un archivo HTML personalizado y generado por Terraform.

🧱 4. Estructura del proyecto

UD04-Caso01/

├── main.tf
├── variables.tf
├── outputs.tf
├── terraform.tfvars
├── templates/
│ └── index.html.tmpl
├── .gitignore

✅ 5. Ejecución paso a paso

1. Inicializa el proyecto:

```
terraform init
```

```
alumno@alumno-virtualbox:~/Desktop/UD02-Caso01$ 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/local v2.5.3...
- Installed hashicorp/local v2.5.3 (signed by HashiCorp)
- Installing kreuzwerker/docker v3.6.2...
```

2. Aplica la infraestructura:

```
terraform apply
```

```
alumno@alumno-virtualbox:~/Desktop/UD02-Caso01$ terraform apply
random_pet.web_names[2]: Refreshing state... [id=brief-sloth]
random_pet.web_names[0]: Refreshing state... [id=settling-polecat]
random_pet.web_names[1]: Refreshing state... [id=willing-bullfrog]
local_file.html_files[0]: Refreshing state... [id=74a72ab4af54bd1f8fe4dc26793e5b23a1e1ad2a]
local_file.html_files[2]: Refreshing state... [id=4dc6900120273adb84c4a1588149848bdac1ac8b]
local_file.html_files[1]: Refreshing state... [id=ea8bacfbe351684019eb1b2d14214f8ba68a9b61]
```

3. Resultado esperado similar a:

```
Apply complete! Resources: 7 added, 0 changed, 0 destroyed.
Outputs:
contenedores_web = [
  {
    archivo = "./site-cool-owl.html"
    nombre  = "nginx-cool-owl"
    puerto  = 8888
    red     = "terraform-net"
  },
```

```
  ...
]
url_local = "http://localhost:8888"
```

```
Outputs:

contenedores_web = [
  {
    "archivo" = "/home/alumno/Desktop/UD02-Caso01/site_settling-polecat.html"
    "nombre" = "nginx-settling-polecat"
    "puerto" = 8888
    "red" = "terraform-net"
  },
  {
    "archivo" = "/home/alumno/Desktop/UD02-Caso01/site_willing-bullfrog.html"
    "nombre" = "nginx-willing-bullfrog"
    "puerto" = 0
    "red" = "terraform-net"
  },
  {
    "archivo" = "/home/alumno/Desktop/UD02-Caso01/site_brief-sloth.html"
    "nombre" = "nginx-brief-sloth"
    "puerto" = 0
    "red" = "terraform-net"
  },
```

4. Podemos verificar otros contenedores con:

```
docker ps
```

```
alumno@alumno-virtualbox:~/Desktop/UD02-Caso01$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                    NAME
S
a277e57e79ee   nginx:latest   "/docker-entrypoint.…"   8 minutes ago   Up 8 minutes   0.0.0.0:32769->80/tcp    ngin
x-willing-bullfrog
9650524504aa   nginx:latest   "/docker-entrypoint.…"   8 minutes ago   Up 8 minutes   0.0.0.0:32768->80/tcp    ngin
x-brief-sloth
e5b5a4c75e91   nginx:latest   "/docker-entrypoint.…"   8 minutes ago   Up 8 minutes   0.0.0.0:8888->80/tcp     ngin
x-settling-polecat
```

5. Accede al navegador y comprobar:
http://localhost:8888

Aunque hay 3 contenedores en marcha, **no accedemos más que al primero, ya que en esta infraestructura no hemos incluido nada relacionada con el balanceo de carga**, como por ejemplo nginx_proxy actúa como reverse proxy (modo HTTP). Estilo veremos en el siguiente caso práctico.

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

```
terraform destroy
```

6. Detalle del contenido de los ficheros

variables.tf

```
# ------------------------------------------------------
# Número de contenedores web que queremos desplegar
# Puedes aumentar este número para escalar horizontalmente
# (es decir, lanzar más servidores NGINX en paralelo)
# ------------------------------------------------------
variable "web_container_count" {
  description = "Número de contenedores web (nginx) a desplegar"
  type        = number
  default     = 2
}

# ------------------------------------------------------
# Mensaje que se mostrará en el archivo HTML de cada contenedor
# Este mensaje será insertado en una plantilla (templatefile)
# útil para personalizar el contenido web desde Terraform
# ------------------------------------------------------
variable "html_message" {
  description = "Mensaje HTML que se mostrará en cada contenedor"
  type        = string
  default     = "¡Hola desde Terraform + Docker!"
}

# ------------------------------------------------------
# Imagen base de Docker a utilizar. Debe tener NGINX instalado
# Por defecto usamos 'nginx:latest', pero puedes cambiarlo
# por otra versión específica como 'nginx:1.21' si lo deseas.
# ------------------------------------------------------
variable "base_image" {
  description = "Imagen de contenedor base (debe tener nginx instalado)"
  type        = string
  default     = "nginx:latest"
}

# ------------------------------------------------------
# Esta variable define si queremos exponer el primer contenedor
# al puerto local del equipo. Es útil para poder acceder a la
# web desde el navegador en 'http://localhost:PUERTO'
# ------------------------------------------------------
variable "publish_first_port" {
  description = "¿Exponer el primer contenedor localmente?"
  type        = bool
  default     = true
}
```

```
# ------------------------------------------------------
# Puerto del host local donde se expondrá el contenedor NGINX
# Solo aplica si publish_first_port = true
# Ejemplo: si es 8080, podrás acceder en http://localhost:8080
# ------------------------------------------------------
variable "exposed_port" {
  description = "Puerto en el host local para el primer contenedor"
  type        = number
  default     = 8080
}
```

main.tf

```
# ------------------------------------------------------
# Configuración básica del proveedor Terraform
# Aquí definimos qué proveedor externo vamos a usar
# En este caso: Docker (usando el proveedor de kreuzwerker)
# ------------------------------------------------------
terraform {
 required_providers {
   docker = {
     source  = "kreuzwerker/docker"  # Fuente oficial del provider Docker
     version = "~> 3.0"              # Versión compatible
   }
 }
}

# ------------------------------------------------------
# Inicializa el proveedor Docker para que Terraform
# pueda interactuar con el servicio Docker local
# ------------------------------------------------------
provider "docker" {}

# ------------------------------------------------------
# Crea una red Docker personalizada para conectar todos
# los contenedores que Terraform desplegará
# Esto permite que los contenedores se comuniquen entre sí
# ------------------------------------------------------
resource "docker_network" "terraform_net" {
 name = "terraform-net"  # Nombre visible de la red Docker
}

# ------------------------------------------------------
# Genera una lista de nombres aleatorios y únicos para los
# contenedores web usando el recurso 'random_pet'
# Esto evita colisiones y facilita el despliegue escalado
# ------------------------------------------------------
resource "random_pet" "web_names" {
 count  = var.web_container_count  # Se crea un nombre por cada contenedor
 length = 2                        # El nombre tendrá dos palabras (ej. kind-fox)
}
```

```
# ------------------------------------------------------
# Genera archivos HTML personalizados para cada contenedor
# usando plantillas dinámicas (templatefile)
# Los archivos se guardan localmente y luego se montan
# en los contenedores como contenido web
# ------------------------------------------------------
resource "local_file" "html_files" {
 count    = var.web_container_count  # Un archivo por contenedor
 filename = "${path.module}/site_${random_pet.web_names[count.index].id}.html"  #
Nombre del archivo

 # Contenido dinámico basado en la plantilla 'index.html.tmpl'
 content = templatefile("${path.module}/templates/index.html.tmpl", {
   mensaje = var.html_message                                    # Texto definido
por el usuario
   nombre  = random_pet.web_names[count.index].id                # Nombre único
del contenedor
 })
}

# ------------------------------------------------------
# Se asegura de que la imagen base de Docker (nginx:latest)
# esté disponible localmente. Si no está, la descarga
# ------------------------------------------------------
resource "docker_image" "nginx" {
 name = var.base_image  # Imagen definida por variable (por defecto: nginx:latest)
}

# ------------------------------------------------------
# Crea los contenedores web (Nginx) usando la imagen
# descargada previamente y configura:
# - nombres únicos
# - conexión a la red docker
# - montaje de archivos HTML
# - exposición del primer contenedor al host (opcional)
# ------------------------------------------------------
resource "docker_container" "web" {
 count = var.web_container_count  # Crea uno o varios contenedores según se indique

 # Nombre único por contenedor usando nombre aleatorio
 name  = "nginx-${random_pet.web_names[count.index].id}"
  # Imagen a utilizar
 image = docker_image.nginx.name

 # Conexión del contenedor a la red personalizada
 networks_advanced {
   name = docker_network.terraform_net.name
 }

 # Exposición del puerto 80 del contenedor al host (solo el primer contenedor si está
activado)
 ports {
```

```
    internal = 80
    # Si publish_first_port está en true y es el primer contenedor (index 0), lo
expone
    # Si no, deja el valor en 0 (no expone)
    external = var.publish_first_port && count.index == 0 ? var.exposed_port : 0
  }

  # Montaje del archivo HTML generado en el contenedor
  volumes {
    host_path      = abspath(local_file.html_files[count.index].filename)
    container_path = "/usr/share/nginx/html/index.html"  # Ruta donde NGINX lo leerá
por defecto
  }

  # Opcional: evita reinicio automático
  restart = "no"
}
```

📄 outputs.tf

```
# ------------------------------------------------------
# Output principal que devuelve una lista con la información
# de cada contenedor web creado (uno por recurso docker_container.web)
#
# Se muestra:
# - El nombre del contenedor
# - El puerto en el host (si está expuesto)
# - La ruta al archivo HTML que se montó
# - El nombre de la red Docker a la que pertenece
# ------------------------------------------------------
output "contenedores_web" {
  description = "Lista de contenedores desplegados"

  value = [
    for i in docker_container.web :   # Recorremos cada contenedor web creado
    {
      nombre   = i.name    # Nombre único del contenedor (nginx-random)
      puerto   = try(i.ports[0].external, "No expuesto") # Si tiene un puerto
expuesto, lo muestra; si no, indica "No expuesto"
      archivo  = abspath(tolist(i.volumes)[0].host_path) # 🔧 Convertimos el set en
lista antes de acceder. Ruta local al archivo HTML montado
      red      = docker_network.terraform_net.name       # Nombre de la red Docker
utilizada (terraform-net)
    }
  ]
}

# ------------------------------------------------------
# Output que devuelve la URL local del contenedor principal
# Solo muestra la URL si publish_first_port = true
# En caso contrario, informa que no está expuesto
#
```

```
# Esto permite al usuario abrir el navegador y acceder al contenedor:
# Ejemplo: http://localhost:8888
# -----------------------------------------------------
output "url_local" {
 description = "URL del contenedor principal (si está expuesto)"

 # Si publish_first_port = true, devuelve la URL
 # Si no, muestra "No expuesto"
 value = var.publish_first_port ? "http://localhost:${var.exposed_port}" : "No
expuesto"
}
```

📄templates/index.html.tmpl

```
<!--
  Este archivo es una plantilla HTML que será procesada por Terraform
  utilizando la función templatefile().
  Las variables ${nombre} y ${mensaje} serán reemplazadas dinámicamente
  por valores definidos en el archivo Terraform.
-->

<!DOCTYPE html>
<html>
<head>
  <!--
    El título de la pestaña del navegador incluirá el nombre del contenedor.
    Este valor se genera con la función 'random_pet' desde Terraform.
  -->
  <title>Servidor ${nombre}</title>

  <!--
    Estilo básico para que la página se vea más bonita.
    Se centra el texto y se usa una fuente legible.
  -->
  <style>
    body {
      font-family: sans-serif;
      text-align: center;
      background-color: #f2f2f2;
      padding-top: 50px;
    }

    h1 {
      color: #333;
    }
  </style>
</head>

<body>
  <!--
    Este encabezado mostrará el mensaje HTML definido por el usuario
    en la variable 'html_message' desde Terraform.
```

```
  -->
  <h1>${mensaje}</h1>

  <!--
    Texto informativo para que el alumno o usuario sepa
    que esta página fue generada automáticamente por Terraform.
  -->
  <p>Servidor generado automáticamente con <strong>Terraform</strong> 🤖</p>

  <!--
    Se muestra el nombre del contenedor Docker en la página.
    Este nombre fue generado automáticamente con random_pet.
  -->
  <p>Contenedor: <code>${nombre}</code></p>
</body>
</html>
```

terraform.tfvars

```
# ------------------------------------------------------
# Número de contenedores web (NGINX) que se van a crear
# En este caso se crearán 3 contenedores diferentes
# Cada uno tendrá su propio archivo HTML y nombre único
# ------------------------------------------------------
web_container_count = 3

# ------------------------------------------------------
# Este es el mensaje que se insertará dinámicamente
# en cada página web HTML generada.
# Puedes modificarlo y volver a aplicar (terraform apply)
# para ver cómo se actualizan todas las páginas.
# ------------------------------------------------------
html_message = "¡Esta es una web generada automáticamente!"

# ------------------------------------------------------
# Puerto del host local donde se expondrá el PRIMER contenedor
# Si estás usando `publish_first_port = true` en variables.tf,
# podrás acceder al contenedor en: http://localhost:8888
#
# IMPORTANTE: Solo se expone el primer contenedor.
# Los otros siguen funcionando en red privada (docker network).
# ------------------------------------------------------
exposed_port = 8888
```