Introducción a Terraform y Salt Project

Unidad 02. 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 02 - Caso práctico 01

🔥 Proyecto: WordPress + MySQL + phpMyAdmin con Terraform + Docker

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

En este caso práctico aprenderemos a **desplegar una infraestructura completa de WordPress con base de datos MySQL y panel de administración phpMyAdmin**, utilizando **Terraform** como herramienta de *infraestructura como código (IaC)* y **Docker** como entorno de ejecución.

Terraform se encargará de **crear automáticamente** todos los componentes necesarios:

- Una **red Docker interna** que conecta los tres servicios.
- Un **contenedor MySQL** con volumen persistente para almacenar los datos.
- Un **contenedor WordPress** configurado para conectarse a la base de datos.
- Un **contenedor phpMyAdmin** que permite administrar MySQL desde el navegador.

La infraestructura será **parametrizable** mediante variables, de modo que podrás modificar fácilmente contraseñas, nombres de base de datos o puertos de acceso. Todo el entorno se desplegará localmente con un simple comando (terraform apply) y se eliminará con otro (terraform destroy), sin necesidad de ejecutar manualmente órdenes de Docker.

Al finalizar el caso práctico, obtendrás un entorno funcional accesible desde el navegador:

- 🌐 WordPress: http://localhost:8080
- ⚙️ phpMyAdmin: http://localhost:8081

Este ejercicio te permitirá comprender cómo **Terraform gestiona y coordina servicios Docker interdependientes**, garantizando el orden de arranque, la persistencia de datos y la automatización total del despliegue.

🎯 2. Objetivo

Desplegar con Terraform un entorno local compuesto por:

- 🗄️ **MySQL 5.7** (base de datos persistente)
- 🌐 **WordPress** (sitio web)
- ⚙️ **phpMyAdmin** (interfaz de administración de MySQL)

Todo conectado por una red Docker interna llamada wordpress_net, y totalmente parametrizable mediante variables.

🗂️ 3. Estructura del proyecto

UD02CasoPractico01/
├── main.tf
├── variables.tf
├── outputs.tf
└── terraform.tfvars (opcional, pero lo incluimos)

🚀 4. Despliegue paso a paso

✅ Requisitos previos

Para ejecutar este caso práctico, necesitas tener instalados en tu máquina:

- **Terraform**
- **Docker** (Docker Engine activo y funcionando)

💡 Puedes comprobar que están correctamente instalados ejecutando:

```
terraform version
docker version
```

📦 Descarga del proyecto

1. Descarga los archivos correspondientes a este caso práctico, asegurándote de mantener la **estructura de directorios descrita en el punto anterior**.
2. Una vez descargados, abre una terminal y sitúate en el directorio raíz del proyecto (donde se encuentra el archivo main.tf).

🚀 5. Ejecución paso a paso

Inicializa el proyecto mediante la orden:

```
terraform init
```

```
alumno@alumno-virtualbox:~/Desktop/UD02-Caso01$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding kreuzwerker/docker versions matching ">= 2.20.0"...
- Installing kreuzwerker/docker v3.6.2...
- Installed kreuzwerker/docker v3.6.2 (self-signed, key ID BD080C4571C6104C)
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://developer.hashicorp.com/terraform/cli/plugins/signing
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!
```

Tras ello aplicamos la infraestructura:

```
terraform apply
```

```
alumno@alumno-virtualbox:~/Desktop/UD02-Caso01$ 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.mysql will be created
  + resource "docker_container" "mysql" {
      + attach                                      = false
      + bridge                                      = (known after apply)
      + command                                     = (known after apply)
      + container_logs                              = (known after apply)
      + container_read_refresh_timeout_milliseconds = 15000
      + entrypoint                                  = (known after apply)
      + env                                         = [
          + "MYSQL_DATABASE=wpdb",
          + "MYSQL_ROOT_PASSWORD=admin123",
        ]
      + exit_code                                   = (known after apply)
```

Espera unos segundos mientras Docker descarga las imágenes y levanta los contenedores.

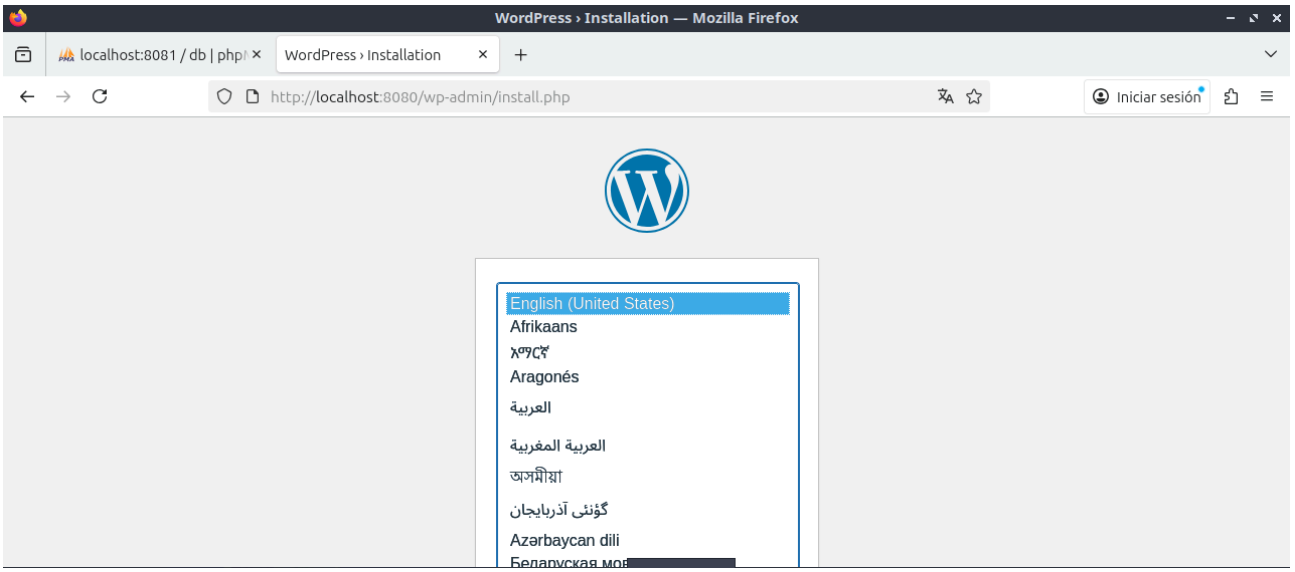
Una vez finalizado, puedes verificar el estado de los contenedores. Puedes hacerlo con la orden:

```
docker ps
```

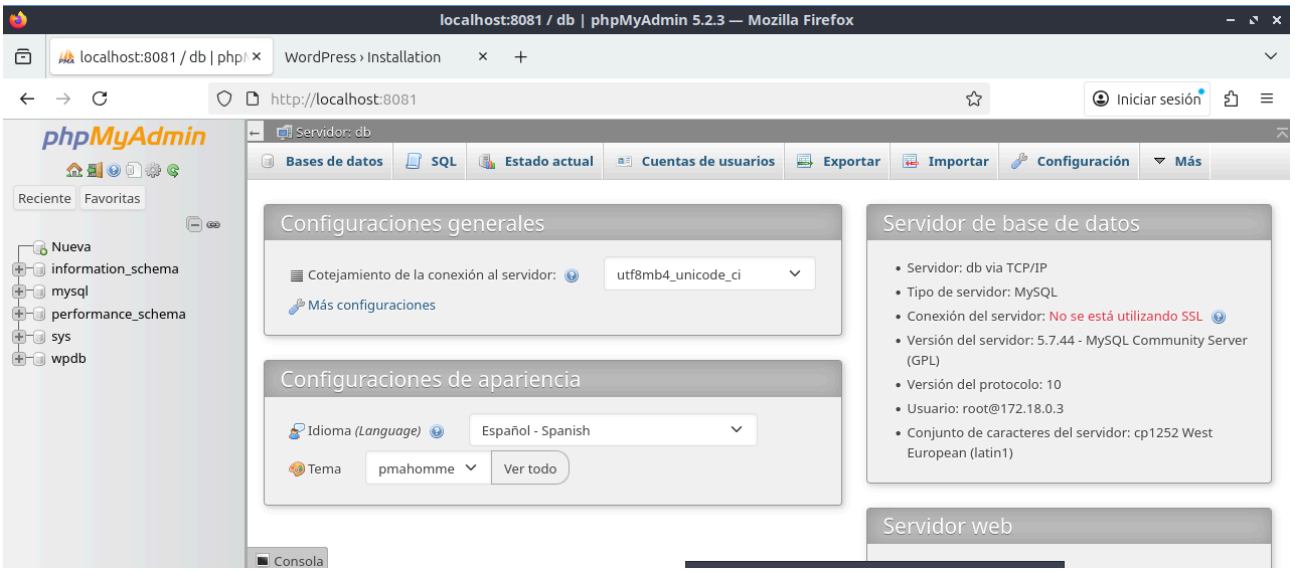
Deberías ver 3 contenedores:

```
Apply complete! Resources: 8 added, 0 changed, 0 destroyed.

Outputs:

database_info = {
  "database" = "wpdb"
  "password" = "admin123"
  "user" = "root"
}
phpmyadmin_url = "http://localhost:8081"
wordpress_url = "http://localhost:8080"
alumno@alumno-virtualbox:~/Desktop/UD02-Caso01$ docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED              STATUS              PORTS
                  NAMES
5d27afbd2900   wordpress:latest   "docker-entrypoint.s…"   47 seconds ago       Up 34 seconds       0.0.0.0:8080-
>80/tcp           wordpress_app
4fbb7f78fc4d   phpmyadmin:latest  "/docker-entrypoint.…"   About a minute ago   Up About a minute   0.0.0.0:8081-
>80/tcp           phpmyadmin_ui
514f26300e52   mysql:5.7          "docker-entrypoint.s…"   2 minutes ago        Up 2 minutes        0.0.0.0:3306-
>3306/tcp, 33060/tcp   mysql_db
alumno@alumno-virtualbox:~/Desktop/UD02-Caso01$
```

Ahora, está todo desplegado y puedes acceder a Wordpress y a PHPMyAdmin desde tu navegador:
🌐 **WordPress:** http://localhost:8080



⚙️ **phpMyAdmin:** http://localhost:8081



🧼 Limpieza

Cuando termines el laboratorio, puedes destruir todos los recursos creados (contenedores, redes, volúmenes, etc.) con la orden:

```
terraform destroy
```

```
alumno@alumno-virtualbox:~/Desktop/UD02-Caso01$ terraform destroy
docker_image.phpmyadmin: Refreshing state... [id=sha256:1dacec7c8bc93c7c76d3d3abc0ab3bf6c209d4d1405dfd8c85170740aa52feb5phpmyadmin:latest]
docker_image.mysql: Refreshing state... [id=sha256:5107333e08a87b836d48ff7528b1e84b9c86781cc9f1748bbc1b8c42a870d933mysql:5.7]
docker_volume.mysql_data: Refreshing state... [id=mysql_data]
docker_network.wp_net: Refreshing state... [id=34afe346a07be9085bfbc90cc6471cdfd5e518ebc9dcca3fcc564b6ed6ff4bd8]
docker_image.wordpress: Refreshing state... [id=sha256:1e1d6ce7c102e07cf622674deb89b0d75b35005cdb003d7427b4aa80c3e2cc7fwordpress:latest]
docker_container.mysql: Refreshing state... [id=514f26300e52399f4d8dbb31cb2780c4abdee6b73e18ec8f1ca4847c473b8754]
docker_container.phpmyadmin: Refreshing state... [id=4fbb7f78fc4d00b3bf735005b0ce2e34188daa5fc99d38c81906a6f0b1a4e96b]
```

Esto eliminará los contenedores, la red, **y el volumen de MySQL** (los datos no quedan persistentes).

🧠 6. Explicación del flujo

- Terraform → Provider Docker: se conecta al daemon local (por defecto /var/run/docker.sock)
- docker_network.wp_net: crea la red interna donde los contenedores se comunican por alias (db, wordpress_app, phpmyadmin_ui)
- MySQL arranca primero → expone puerto 3306 → usa un volumen persistente
- WordPress y phpMyAdmin se conectan al servicio MySQL a través del alias db
- depends_on garantiza el orden correcto de arranque
- Outputs imprimen las URLs al finalizar la ejecución

🧩 Resultado final

✅ Terraform crea toda la infraestructura local declarativamente.
✅ WordPress queda disponible en http://localhost:8080
✅ phpMyAdmin queda disponible en http://localhost:8081
✅ Los datos de la base se mantienen gracias al volumen mysql_data

📄 7. Detalles del contenido de los ficheros

📄 variables.tf

```
# Contraseña root para MySQL
variable "mysql_root_password" {
 description = "Contraseña root de MySQL"
 type        = string
 default     = "supersecurepass"
}

# Nombre de la base de datos donde WordPress guardará su contenido
variable "mysql_database" {
 description = "Nombre de la base de datos de WordPress"
 type        = string
 default     = "wordpress"
}

# Puerto en el que se expone el sitio WordPress (en tu máquina local)
variable "wordpress_port" {
 description = "Puerto local para WordPress"
 type        = number
 default     = 8080
}

# Puerto para acceder a phpMyAdmin desde el navegador
variable "phpmyadmin_port" {
 description = "Puerto local para phpMyAdmin"
```

```
 type        = number
 default     = 8081
}

# Imagen utilizada para el contenedor MySQL (se puede cambiar fácilmente)
variable "mysql_image" {
 description = "Imagen Docker de MySQL"
 type        = string
 default     = "mysql:5.7"
}

# Imagen utilizada para el contenedor WordPress
variable "wordpress_image" {
 description = "Imagen Docker de WordPress"
 type        = string
 default     = "wordpress:latest"
}

# Imagen utilizada para el contenedor phpMyAdmin
variable "phpmyadmin_image" {
 description = "Imagen Docker de phpMyAdmin"
 type        = string
 default     = "phpmyadmin:latest"
}
```

📄 main.tf

```
terraform {
 # Aquí definimos qué proveedores (providers) vamos a usar en este proyecto
 required_providers {
   # Declaramos que vamos a usar el provider llamado "docker"
   docker = {
     # Especificamos el origen del provider: "kreuzwerker/docker"
     source = "kreuzwerker/docker"

     # Indicamos que queremos usar la versión 2.20.0 o superior del provider Docker
     # Es buena práctica fijar versiones o rangos para evitar problemas con futuras
actualizaciones
     version = ">= 2.20.0"
   }
 }
}

# Proveedor Docker local
provider "docker" {}

# 📡 Creamos una red Docker interna para conectar todos los contenedores
resource "docker_network" "wp_net" {
 name = "wordpress_net"
}
```

```
# 💾 Creamos un volumen persistente para los datos de MySQL
resource "docker_volume" "mysql_data" {
 name = "mysql_data"
}

# 🐬 Imagen de MySQL
resource "docker_image" "mysql" {
 name = var.mysql_image
}

# 🐬 Contenedor de MySQL
resource "docker_container" "mysql" {
 name  = "mysql_db"
 image = docker_image.mysql.name

 # Variables de entorno necesarias para inicializar la base de datos
 env = [
   "MYSQL_ROOT_PASSWORD=${var.mysql_root_password}",
   "MYSQL_DATABASE=${var.mysql_database}"
 ]

 # Montamos el volumen para persistencia de datos
 volumes {
   volume_name    = docker_volume.mysql_data.name
   container_path = "/var/lib/mysql"
 }

 # Lo conectamos a la red interna y le damos el alias "db"
 networks_advanced {
   name    = docker_network.wp_net.name
   aliases = ["db"]
 }

 # Exponemos el puerto 3306 internamente (opcional)
 ports {
   internal = 3306
   external = 3306
 }
}

# 🌐 Imagen de WordPress
resource "docker_image" "wordpress" {
 name = var.wordpress_image
}

# 🌐 Contenedor WordPress
resource "docker_container" "wordpress" {
 name  = "wordpress_app"
 image = docker_image.wordpress.name

 # Variables de entorno para que WordPress se conecte a MySQL
 env = [
```

```
    "WORDPRESS_DB_HOST=db:3306",                # db = alias en la red Docker
    "WORDPRESS_DB_NAME=${var.mysql_database}",
    "WORDPRESS_DB_USER=root",
    "WORDPRESS_DB_PASSWORD=${var.mysql_root_password}"
  ]

  # Conectado a la misma red que MySQL
  networks_advanced {
    name = docker_network.wp_net.name
  }

  # Puerto 80 interno expuesto en el puerto local definido por el usuario
  ports {
    internal = 80
    external = var.wordpress_port
  }

  # Este contenedor depende del de MySQL para arrancar correctamente
  depends_on = [docker_container.mysql]
}

# ⚙️ Imagen de phpMyAdmin
resource "docker_image" "phpmyadmin" {
  name = var.phpmyadmin_image
}

# ⚙️ Contenedor phpMyAdmin
resource "docker_container" "phpmyadmin" {
  name  = "phpmyadmin_ui"
  image = docker_image.phpmyadmin.name

  # Variables de entorno para conectarse a MySQL
  env = [
    "PMA_HOST=db",                              # Usamos el alias de red
    "PMA_USER=root",
    "PMA_PASSWORD=${var.mysql_root_password}"
  ]

  # Conectado a la misma red que los demás
  networks_advanced {
    name = docker_network.wp_net.name
  }

  # Puerto 80 interno expuesto como 8081 localmente (por defecto)
  ports {
    internal = 80
    external = var.phpmyadmin_port
  }

  # También depende de que MySQL esté funcionando primero
  depends_on = [docker_container.mysql]
}
```

outputs.tf

```
# URL para acceder a WordPress desde el navegador
output "wordpress_url" {
 description = "URL de acceso a WordPress"
 value       = "http://localhost:${var.wordpress_port}"
}

# URL para acceder a phpMyAdmin desde el navegador
output "phpmyadmin_url" {
 description = "URL de acceso a phpMyAdmin"
 value       = "http://localhost:${var.phpmyadmin_port}"
}

# Información útil para conectarse a la base de datos manualmente
output "database_info" {
 description = "Información de la base de datos creada"
 value = {
   database = var.mysql_database
   user     = "root"
   password = var.mysql_root_password
 }
}
```

(Opcional) terraform.tfvars

```
# Contraseña del usuario root de MySQL
mysql_root_password = "admin123"

# Nombre de la base de datos que usará WordPress
mysql_database = "wpdb"

# Puerto externo en tu máquina local donde estará accesible WordPress
wordpress_port = 8080

# Puerto externo en tu máquina local donde estará accesible phpMyAdmin
phpmyadmin_port = 8081
```