Introducción a Terraform y Salt Project

Unidad 02. Terraform con Docker y Vagrant

Autor: Sergi García

Actualizado Noviembre 2025

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 - Terraform con Docker y Vagrant

1. 📘 Introducción: Terraform en entornos locales

1.1 ¿Por qué usar Terraform localmente?

Terraform es una herramienta ampliamente usada para gestionar infraestructura en la nube (AWS, Azure, GCP). Sin embargo, **no necesitas una cuenta en la nube para aprenderlo ni para probarlo**.

Gracias a proveedores como **Docker** y **Vagrant**, puedes usar Terraform para:

- Crear **infraestructura local automatizada.**
- Simular entornos reales para pruebas o desarrollo.
- Reforzar tus conocimientos de Infraestructura como Código (IaC).
- Probar configuraciones, redes o herramientas como Salt, Prometheus, etc.

💡 Aprender Terraform localmente con Docker o Vagrant te permite practicar sin costo ni riesgo, lo que es ideal para entornos educación.

1.2 Beneficios de Docker y Vagrant como entornos de pruebas

Característica	Docker 🐳	Vagrant 📦
Tipo de entorno	Contenedor (ligero)	Máquina virtual (completo)
Velocidad	Rápida (milisegundos)	Lenta (requiere arranque de VM)
Aislamiento	Medio	Alto (kernel separado)
Persistencia	Limitada (a menos que uses volúmenes)	Alta
Casos de uso	Microservicios, pruebas rápidas	Simular servidores reales
Requisitos	Docker Engine / Podman	Vagrant + VirtualBox / Libvirt

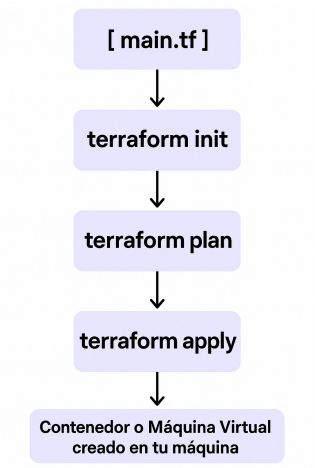
Ambas tecnologías permiten:

- Desplegar recursos de forma **repetible** y controlada
- Versionar y mantener configuración en **código**
- Integrar con otros sistemas, como **SaltStack, Ansible o Prometheus**

Y al combinarlas con Terraform, el flujo de despliegue queda unificado.

1.3 Flujo de trabajo típico: "Terraform + Contenedor o Máquina virtual"

Terraform actúa como **orquestador** que declara qué infraestructura quieres tener. Ya sea que trabajes con Docker o Vagrant, el flujo general es el mismo:



Este patrón funciona igual para cualquier caso, sea el que sea, por ejemplo:

- Un contenedor Nginx.
- Una VM Ubuntu 20.04.
- Una red local con varios recursos conectados.

🔁 Además, puedes destruir el entorno con terraform destroy y dejar tu equipo limpio.

2. 🧩TERRAFORM CON DOCKER: PRIMEROS PASOS

2.1 ¿Qué es Docker y cómo se integra con Terraform?

Docker es una plataforma que permite ejecutar aplicaciones dentro de **contenedores**: entornos ligeros, portables y aislados.

Terraform puede usar Docker como proveedor a través del plugin oficial kreuzwerker/docker, lo que permite declarar y crear contenedores con código como este:

```
resource "docker_container" "nginx" {
  name  = "my-nginx"
  image = "nginx:latest"
}
```

Terraform se convierte en una forma de **gestionar contenedores de forma declarativa**, permitiendo:

- Definir imágenes, puertos, volúmenes, redes, etc.
- Versionar tu entorno de contenedores.
- Automatizar laboratorios y entornos de desarrollo/testing.

2.2 Instalación y requisitos previos

Para poder usar Docker con Terraform, necesitas tener instalado:

Componente	Requisito
Docker Engine	https://docs.docker.com/get-docker/

Terraform CLI	https://developer.hashicorp.com/terraform/install

Una vez instalados:

```
docker version     # Verifica Docker
terraform version  # Verifica Terraform
```

🧪 También asegúrate de que Docker esté corriendo (docker ps debe funcionar sin error).

2.3 Provider docker: configuración básica

Terraform necesita que declares el proveedor `docker` en tu bloque `terraform`:

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = ">= 2.20.0"
    }
  }
}

provider "docker" {}
```

Este bloque:

- Asegura que el proveedor sea instalado automáticamente.
- Usa la versión más reciente compatible.
- Se conecta por defecto al socket local de Docker (/var/run/docker.sock en Linux/macOS, Docker Desktop en Windows)

2.4 Crear un contenedor simple con Terraform

Veamos un ejemplo completo para crear un contenedor `nginx` expuesto en el puerto `8080`:

🗂️ **Estructura del proyecto**

/docker
├── main.tf
└── variables.tf (opcional)

📄 **main.tf**

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = ">= 2.20.0"
    }
  }
}

provider "docker" {}
```

```
resource "docker_image" "nginx" {
  name = "nginx:latest"
}

resource "docker_container" "nginx" {
  name  = "web-nginx"
  image = docker_image.nginx.name

  ports {
    internal = 80
    external = 8080
  }
}
```

🧪 **Instrucciones**

```
terraform init      # Inicializa el proyecto y descarga el provider
terraform apply     # Despliega el contenedor
```

Verifica el contenedor con:

```
docker ps           # Deberías ver "web-nginx"
```

Navega a http://localhost:8080 y deberías ver la página por defecto de Nginx 🟢

2.5 Exponer puertos y definir volúmenes

Puedes agregar más complejidad declarando:

- Puertos múltiples.
- Volúmenes para persistencia.
- Variables para parametrizar todo esto.

Ejemplo con volumen:

```
resource "docker_volume" "nginx_data" {
  name = "nginx_data"
}

resource "docker_container" "nginx" {
  name  = "web-nginx"
  image = docker_image.nginx.name

  ports {
    internal = 80
    external = 8080
  }

  volumes {
    volume_name    = docker_volume.nginx_data.name
    container_path = "/usr/share/nginx/html"
  }
}
```

2.6 Buenas prácticas y errores comunes

✅ **Buenas prácticas:**

- Usa nombres únicos para evitar conflictos (web-nginx-${count.index}, etc.).
- Parametriza puertos e imágenes con variables (nginx:alpine, port = 8080).
- Usa terraform destroy para limpiar fácilmente los recursos.

❌ **Errores comunes:**

Error	Causa
Error: Unable to connect to Docker	Docker no está corriendo
docker_container name already in use	Nombre duplicado
provider not found	No ejecutaste terraform init

3. 🖥️ Terraform con Vagrant: máquinas virtuales

3.1 ¿Qué es Vagrant y para qué se usa?

Vagrant es una herramienta que automatiza la creación y configuración de máquinas virtuales locales (VirtualBox, Hyper-V, libvirt, etc.). Se usa para crear entornos reproducibles de desarrollo y pruebas que reflejan servidores reales sin necesidad de cloud.

3.2 Requisitos previos

- **Vagrant** instalado (https://www.vagrantup.com/docs/installation).
- **VirtualBox** o el proveedor de VM elegido (VirtualBox es el más común) https://www.virtualbox.org/wiki/Downloads
- **Terraform CLI** instalado (https://developer.hashicorp.com/terraform).

Verifica instalaciones:

```
vagrant --version
vboxmanage --version   # VirtualBox
terraform version
```

3.3 Consideraciones

- Hay providers comunitarios para Vagrant — útiles, pero pueden variar en características y mantenimiento.
- Alternativa universal: generar un Vagrantfile y ejecutar vagrant up/vagrant destroy desde Terraform (null_resource + local-exec). Es más dependiente del sistema local, pero predecible.
- Para desarrollo y pruebas es común usar Vagrant para simular máquinas completas (kernel separado) y Docker para servicios ligeros.

3.4 Ejemplo — Usando el provider comunitario bmatcuk/vagrant

main.tf

```
terraform {
  required_providers {
    vagrant = {
      source  = "bmatcuk/vagrant"
      version = ">= 0.1.0"
    }
  }
  required_version = ">= 1.0"
}

provider "vagrant" {}

variable "vm_name" {
  type    = string
  default = "tf-vagrant-vm"
}

variable "box" {
  type    = string
  default = "ubuntu/bionic64"
}

variable "cpus" {
  type    = number
  default = 1
}

variable "memory" {
  type    = number
  default = 1024
}

resource "vagrant_vm" "default" {
  name   = var.vm_name
  box    = var.box
  cpus   = var.cpus
  memory = var.memory
}
```

outputs.tf

```
output "vm_name" {
  value = vagrant_vm.default.name
  description = "Nombre de la VM creada por Vagrant"
}
```

Uso

```
terraform init
terraform apply
```

Verificación

- vagrant global-status o vagrant status en el directorio donde el provider creó la VM.
- Conectar con vagrant ssh <name> si el provider deja la Vagrantfile visible.

Limitaciones

- El set exacto de atributos (redes, provisioners, IPs) depende del provider. Lee su documentación.
- Algunas implementaciones no exponen la IP o el SSH config como outputs directos.

3.5 Buenas prácticas y recomendaciones

- **Versiona** el Vagrantfile o el código Terraform en Git (incluye archivos de provisioning).
- No mezcles demasiadas cosas en un solo null_resource. Estructura el flujo: crear carpeta → escribir Vagrantfile → vagrant up → extraer ssh config.
- Evita hardcodear IPs que choquen con tu red local; usa rangos de host-only (ej. 192.168.56.x) o DHCP.
- Provisionamiento declarativo: para configuraciones complejas, usa scripts idempotentes o herramientas de CM (Salt/Ansible) en lugar de muchos comandos inline.
- Prueba en un entorno local limpio antes de usar en CI: la ejecución de vagrant up depende del estado local y VirtualBox.

3.6 ¿Qué son las Vagrant Boxes? + Ejemplo con Windows 10 🪟

📦 ¿Qué es una "Box" en Vagrant?

Una **box** en Vagrant (Vagran boxes) es una **plantilla preconfigurada de sistema operativo**. Es el equivalente a una "imagen base" que usas para crear nuevas máquinas virtuales de forma rápida y estandarizada.

Cada box incluye:

- Un sistema operativo (Ubuntu, Debian, Windows, etc.)
- Drivers y configuraciones necesarias para funcionar con el proveedor de virtualización (ej. VirtualBox)
- Configuración para integrarse con Vagrant (SSH o WinRM habilitado, red, etc.)

🧠 Las boxes se descargan automáticamente desde Vagrant Cloud cuando se ejecuta vagrant up.

🗃️ Tipos de Boxes comunes

Sistema Operativo	Box	Notas
Ubuntu	ubuntu/bionic64	LTS, común para labs
Debian	debian/bullseye64	Alternativa estable
CentOS/RHEL	centos/7	Usada en entornos empresariales
Windows	gusztavvargadr/windows-10	Box de Windows 10 Eval (ver más abajo)

🪟 Ejemplo: Box de Windows 10 en modo evaluación

Hay boxes mantenidas por la comunidad que ofrecen **Windows 10 con WinRM habilitado** (para acceso remoto). Estas versiones se pueden usar durante el período de evaluación de Windows (normalmente 90 días).

📦 Ejemplo de box: ➡️ gusztavvargadr/windows-10

⚠️ Requisitos importantes:

- Solo funciona con **VirtualBox** o **Hyper-V**
- Necesitas tener **Vagrant ≥ 2.2.10**
- Consume más recursos: mínimo 4 GB RAM y 2 CPUs

📄 Ejemplo de Vagrantfile (Windows 10)

Aquí tienes un ejemplo simple que podrías generar desde Terraform con local-exec, o escribir a mano para pruebas:

```
Vagrant.configure("2") do |config|
  config.vm.box = "gusztavvargadr/windows-10"
  config.vm.communicator = "winrm"

  config.vm.provider "virtualbox" do |vb|
    vb.name = "win10-eval"
    vb.memory = 4096
    vb.cpus = 2
  end

  config.vm.network "forwarded_port", guest: 3389, host: 3389, id: "rdp"

  config.winrm.username = "vagrant"
  config.winrm.password = "vagrant"
end
```

🧪 Conexión a la VM Windows

1. Ejecuta “vagrant up” (puede tardar unos minutos ⏳).
2. Abre la aplicación de Escritorio Remoto (RDP) en tu sistema operativo.
3. Conéctate a: localhost:3389
4. Usuario: vagrant | Contraseña: vagrant

🔐 También puedes usar herramientas de scripting compatibles con WinRM para ejecutar comandos desde fuera (ej. PowerShell remoto o pywinrm desde Python).

🧼 Limpieza y recursos

Estas VMs consumen bastantes recursos. Recuerda detenerlas desde Vagrant con:

```
vagrant halt
```

Y destruirlas cuando termines:

```
vagrant destroy -f
```

Pero lo más adecuado, es usando Terraform si está hecha la configuración adecuada con

```
terraform apply
```

✅ Consejos para usar Windows boxes con Terraform

Si quieres integrar esto con Terraform (como vimos en el ejemplo null_resource + local-exec), simplemente cambia la variable box en tu main.tf:

```
variable "box" {
  default = "gusztavvargadr/windows-10"
}
```

Y asegúrate de:

- Establecer config.vm.communicator = "winrm" en el contenido del Vagrantfile.
- Configurar los puertos necesarios (ej. RDP en 3389).

💡 Puedes incluso tener lógica condicional en Terraform para cambiar la configuración si `var.os_type == "windows"`.

4. ⚙️Comparación práctica: Docker vs Vagrant

Vamos a intentar comprender las **diferencias funcionales y técnicas** entre Docker y Vagrant, para elegir conscientemente en proyectos locales usando Terraform como orquestador.

4.1 Casos de uso recomendados para cada uno

Docker y Vagrant tienen propósitos similares (simular infraestructura local), pero enfoques distintos.

Necesitas…	Usa Docker 🐳	Usa Vagrant 📦
Un entorno ligero y rápido	✔️	✖️ (mucho más lento)
Aislamiento de procesos	✔️	✔️ (más fuerte, VM completa)
Simular un servidor físico real	✖️	✔️ (kernel separado, OS completo)
Probar servicios (web, DB, cache, etc.)	✔️	✖️
Entorno con GUI (Windows, escritorios)	✖️	✔️ (puede levantar GUI, RDP)
Reproducir problemas de compatibilidad SO	✖️ (usa kernel del host)	✔️ (kernel propio)
Ejecutar en equipos con pocos recursos	✔️ (bajo consumo)	✖️ (alto consumo)
Ejecución CI/CD en entornos controlados	✔️	✖️ (no apto para pipelines rápidos)
Probar herramientas de configuración (Salt, etc.)	✔️ (ideal para testing rápido)	✔️ (entornos más realistas)

📌 **Conclusión**:

- Docker = **ideal para microservicios, APIs, dev rápido**
- Vagrant = **ideal para labs, entornos completos o SOs distintos (Windows, RHEL, etc.)**

4.2 Tiempos de arranque, consumo de recursos y velocidad

Métrica	Docker 🐳	Vagrant (VirtualBox) 📦
Tiempo de arranque	⚡ Milisegundos - segundos	🐢 30-90 segundos (depende del SO)
Uso de CPU y RAM	Muy bajo (por contenedor)	Alto (por VM completa)
Persistencia por defecto	Efímera (requiere volúmenes)	Persistente
Peso de la imagen	20-200 MB	1-4 GB
Rendimiento	Alto (nativo, compartido)	Medio (virtualización completa)

🔍 Docker comparte el kernel del host, mientras que Vagrant crea una VM con kernel separado (más realista pero pesado).

4.3 ¿Puedo usar ambos en un mismo proyecto Terraform?

¡Sí! Terraform lo permite perfectamente 💥

Escenario combinado realista:

- **Docker**: simulas microservicios como Nginx, Prometheus, Salt Minions.
- **Vagrant**: simulas un servidor Windows o una VM central (ej. Salt Master o backend con sistema operativo diferente).

Ejemplo de estructura:

/infra
├── main.tf
├── variables.tf
├── /docker
│ └── nginx_container.tf
└── /vagrant
└── null_resource_windows_vm.tf

Con esto puedes:

1. Levantar un laboratorio completo sin cloud
2. Simular una arquitectura híbrida
3. Mantener todo versionado y orquestado con terraform apply

🧠 Tips de integración

Recomendación	¿Por qué?
Usar redes privadas comunes	Para que Docker y Vagrant puedan comunicarse
Usar outputs de IPs desde Vagrant	Para configurar targets de Prometheus o Salt
Separar recursos por archivo o módulo	Mejora organización y escalabilidad
Correr todo desde un solo terraform apply	Facilita testing y automatización

5. 📚 Webgrafía

Documentación oficial

- **HashiCorp Terraform Documentation** 🔗 https://developer.hashicorp.com/terraform/docs
  - Guía oficial completa sobre instalación, comandos, lenguaje HCL, módulos, providers y buenas prácticas en Terraform.
- **HashiCorp Vagrant Documentation** 🔗 https://developer.hashicorp.com/vagrant/docs
  - Documentación oficial sobre Vagrant, herramienta de HashiCorp para crear y gestionar entornos virtualizados reproducibles mediante archivos de configuración. Incluye instalación, comandos CLI y ejemplos prácticos.
- **Vagrant Cloud** 🔗 https://app.vagrantup.com/boxes/search
  - Plataforma oficial para buscar y compartir *boxes* de Vagrant (imágenes base preconfiguradas para distintos sistemas operativos).
- **Docker Documentation** 🔗 https://docs.docker.com/
  - Documentación oficial sobre instalación, comandos, imágenes, contenedores, volúmenes, redes y *Docker Compose*.
- **Docker Hub** 🔗 https://hub.docker.com/
  - Repositorio oficial de imágenes Docker listas para usar en entornos de desarrollo y producción.
- **Curso de Introducción a Docker** 🔗 https://sergarb1.github.io/CursoIntroduccionADocker/
  - Curso práctico en español que explica los fundamentos de Docker: conceptos de contenedores, imágenes, volúmenes, comandos esenciales y ejemplos aplicados en proyectos reales.