Introducción a Terraform y Salt Project

Unidad 02. Caso práctico 02

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 02

🔥 Proyecto: Windows 10 VM local con Vagrant y Terraform

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

En este caso práctico aprenderemos a **provisionar una máquina virtual de Windows 10** de forma completamente **automatizada** utilizando **Terraform** junto con el *provider* de **Vagrant**, que a su vez empleará **VirtualBox** como hipervisor local.

Terraform actuará como **orquestador declarativo**, encargándose de crear y gestionar la máquina virtual a partir de los ficheros de configuración (main.tf y Vagrantfile). Gracias a esta integración, podremos desplegar un entorno Windows funcional con un solo comando (terraform apply) y eliminarlo fácilmente cuando ya no se necesite (terraform destroy).

La máquina virtual creada incluirá:

- 💻 **Sistema operativo:** Windows 10 (versión de evaluación).
- 🔌 **Acceso remoto habilitado (RDP)** en el puerto **3389**, permitiendo conectarse desde herramientas como **Remmina** o **mstsc**.
- ⚙️ **Configuración reproducible** de hardware (CPU, RAM, red) definida en el Vagrantfile.

Con este ejercicio consolidaremos conceptos clave sobre la **provisión de entornos locales con Terraform** y su integración con herramientas de virtualización. Además, aprenderemos buenas prácticas como el uso de *hashes* para detectar cambios en configuraciones y la lectura de *outputs* para obtener información útil (como los puertos abiertos o las credenciales de acceso).

El resultado será una **VM de Windows 10 totalmente funcional**, accesible mediante **Escritorio Remoto (RDP)**

🎯 2. Objetivo

Desplegar, mediante Terraform y el provider de Vagrant, una máquina virtual local con:

- 💻 **Sistema operativo**: Windows 10 (evaluación)
- 🔌 **Acceso por Escritorio Remoto (RDP)** en el puerto 3389
- 📦 **Provider**: Vagrant + VirtualBox
- 🔧 **Orquestación declarativa**: configuración del entorno desde Terraform

🗂️ 3. Estructura del proyecto

UD02CasoPractico02/
├── main.tf # Código Terraform con provider Vagrant
├── Vagrantfile # Configuración de la máquina virtual

🚀 4. DESPLIEGUE PASO A PASO

✅ Requisitos previos

Para ejecutar este caso práctico correctamente, debes contar con los siguientes componentes **instalados y configurados** en tu sistema:

- Terraform
- Vagrant
- VirtualBox (En el caso práctico se ha usado VirtualBox 7.2.2 con "Extension pack" instalado.

Además, tu sistema debe cumplir con los siguientes requisitos de hardware y virtualización:

- 🔧 **Soporte de virtualización activado** en BIOS/UEFI:
  - Intel VT-x (Virtualization Technology) o AMD-V (AMD Virtualization)
- 🧠 **Si estás dentro de una máquina virtual** (por ejemplo, usando VirtualBox, VMware, Hyper-V o Proxmox):
  - Asegúrate de que **la virtualización anidada esté activada** (nested virtualization)
  - De lo contrario, VirtualBox no podrá crear VMs dentro de la VM anfitriona ❌

ℹ️ Si usas Windows como host, puedes verificar que VT-x o AMD-V esté activo desde el **Administrador de tareas → Rendimiento → CPU**.
En Linux puedes usar:

```
egrep -wo 'vmx|svm' /proc/cpuinfo
```

💡 Verificación de herramientas

Asegúrate de que todo esté correctamente instalado ejecutando estos comandos en una terminal:

```
terraform version
vagrant --version
vboxmanage --version
```

🛠️ Si alguno de estos comandos falla o devuelve error, revisa la instalación o permisos del sistema.

📦 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@equipo:~/Escritorio/UD02-Caso02$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding bmatcuk/vagrant versions matching "4.1.0"...
- Installing bmatcuk/vagrant v4.1.0...
- Installed bmatcuk/vagrant v4.1.0 (self-signed, key ID 77AF9238E6DE2547)
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!
```

Nota importante antes de hacer "terraform apply": es posible durante el proceso de "terraform apply" haya problemas con la descarga de la "Vagrant box", especialmente en entornos educativos donde a veces el ancho de banda es limitado, está capado, etc.

Si quieres mitigarlos, una opción es descargar manualmente la "Vagrant box" antes de usar el comando "terraform apply".

Puedes descargar la "Vagran box" con el comando:

```
vagrant box add gusztavvargadr/windows-10 --provider virtualbox
```

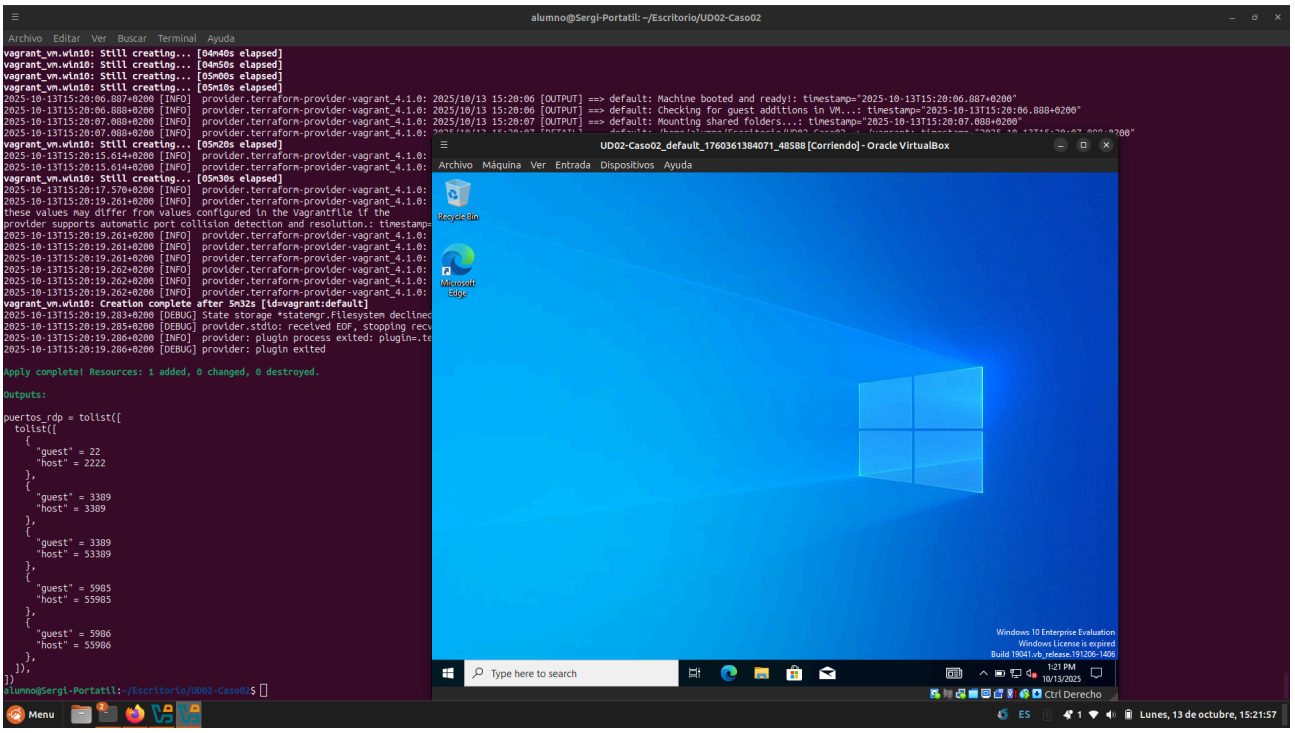
Tras ello y con la "Vagrant box" ya en nuestro sistema aplicamos la infraestructura:

```
terraform apply
```

```
alumno@equipo:~/Escritorio/UD02-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:

  # vagrant_vm.win10 will be created
  + resource "vagrant_vm" "win10" {
      + env             = {
          + "VAGRANTFILE_HASH" = "f78284dd529eb701c06270816ba71743"
        }
      + get_ports       = true
      + id              = (known after apply)
      + machine_names   = (known after apply)
      + name            = "win10-eval"
      + ports           = (known after apply)
      + ssh_config      = (known after apply)
      + vagrantfile_dir = "."
    }

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

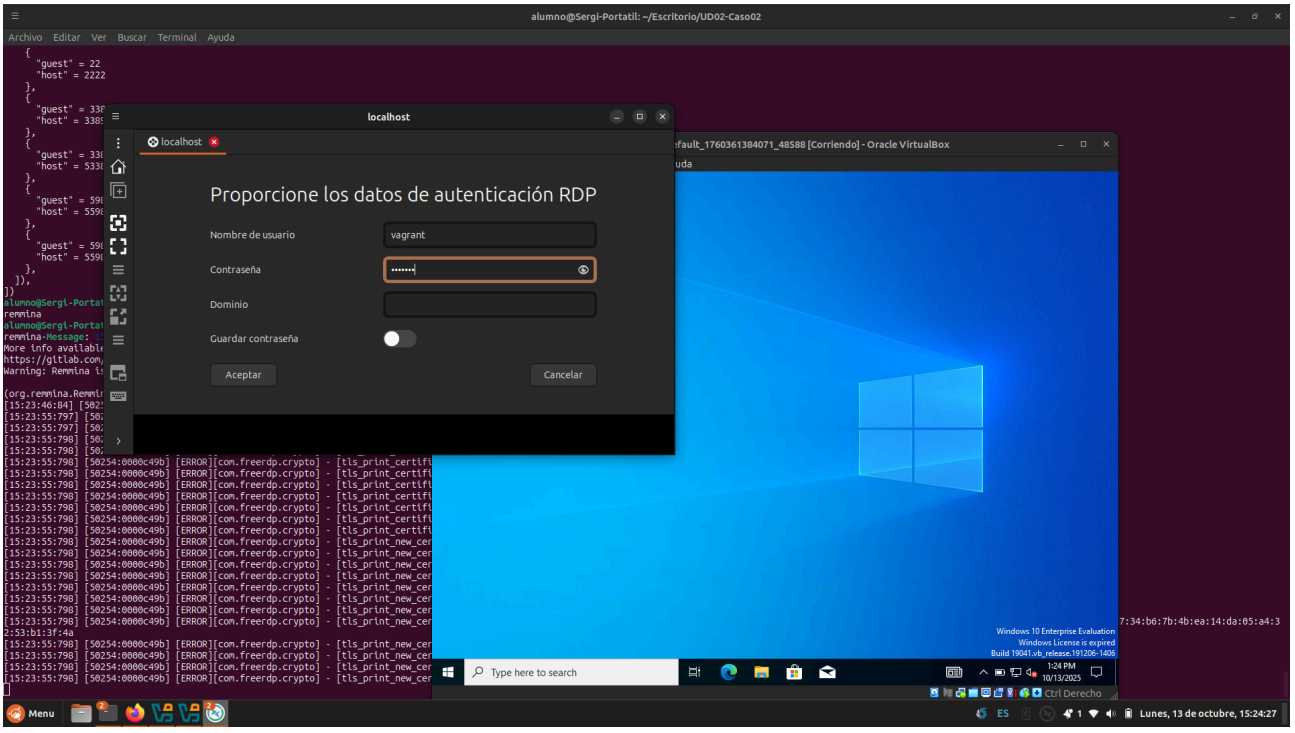
Si no has descargado antes la "Vagrant box", deberás esperar unos minutos mientras Vagrant descarga la Vagrant Box (pesa varios GB).

Espera a que se cree y levante la VM. Una vez finalizado, veremos la máquina lanzada por VirtualBox

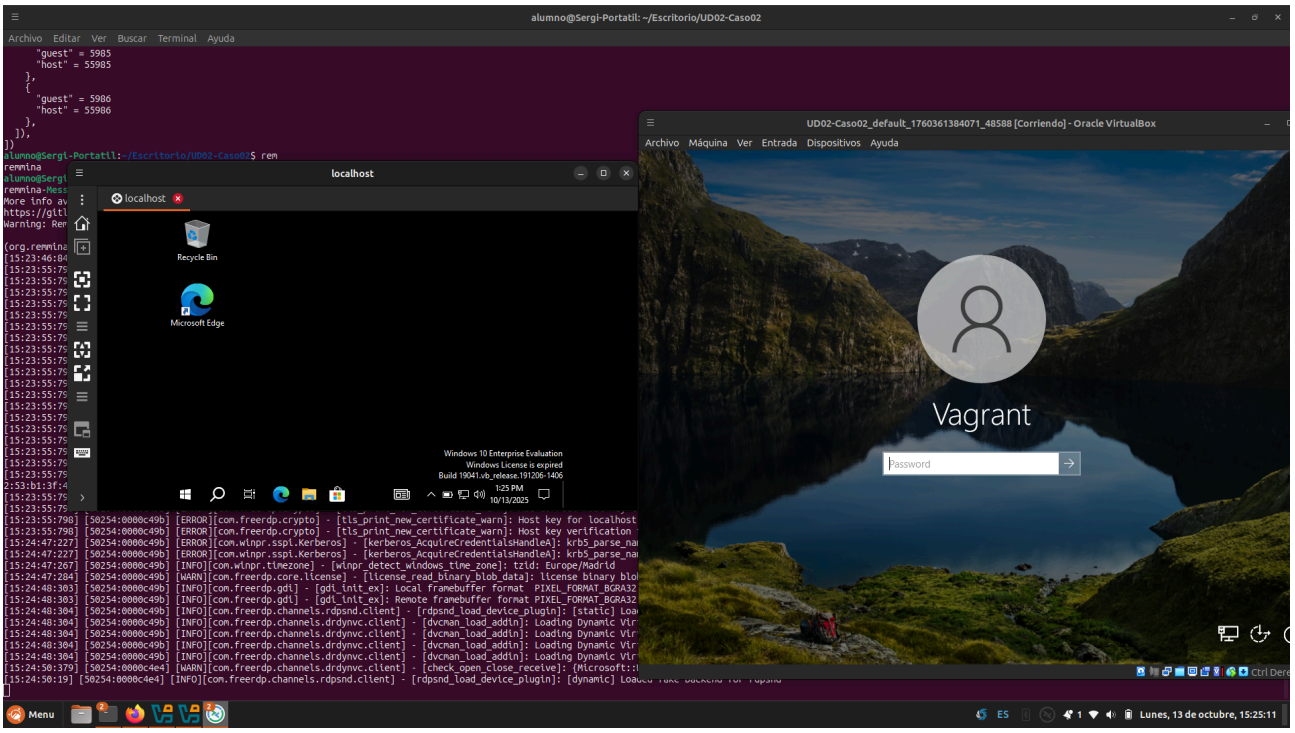


Una vez finalice, podrás conectarte a la máquina virtual por **Escritorio Remoto (RDP)** usando:

- Host: localhost
- Puerto: 3389
- Usuario: vagrant
- Contraseña: vagrant



y una vez conectado con Remmina vía RDP:



🧪 También puedes ver los puertos mapeados con:

```
terraform output puertos_rdp
```

🔐 Acceso a la máquina virtual vía “Escritorio remoto”

Una vez aplicada la infraestructura:

1. Abre tu cliente RDP (en Windows, usa mstsc, en Linux remmina o freerdp)
2. Conéctate a: localhost:3389
3. Credenciales:
    - Usuario: vagrant
    - Contraseña: vagrant

🧪 También puedes acceder a logs con:

```
vagrant global-status
```

🧼 Limpieza

Cuando termines el laboratorio, puedes destruir todos los recursos creados con la orden:

```
terraform destroy
```

```
alumno@Sergi-Portatil:~/Escritorio/UD02-Caso02$ terraform destroy
vagrant_vm.win10: Refreshing state... [id=vagrant:default]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  # vagrant_vm.win10 will be destroyed
  - resource "vagrant_vm" "win10" {
      - get_ports       = true -> null
      - id              = "vagrant:default" -> null
      - machine_names   = [
          - "default",
        ] -> null
      - name            = "win10-eval" -> null
      - ports           = [
          - [
              - {
                  - guest = 22
                  - host  = 2222
                },
              - {
                  - guest = 3389
                  - host  = 3389
                },
              - {
                  - guest = 3389
                  - host  = 53389
                },
              - {
                  - guest = 5985
                  - host  = 55985
                },
              - {
                  - guest = 5986
                  - host  = 55986
                },
            ],
        ] -> null
      - ssh_config      = [
          - {
              - agent       = "false"
              - host        = "127.0.0.1"
```

🧠 6. EXPLICACIÓN DEL FLUJO

- Terraform actúa como orquestador del provider Vagrant.
- Vagrant usa VirtualBox para levantar una VM con Windows 10.
- El archivo Vagrantfile define la configuración exacta de la VM.
- El output puertos_rdp permite inspeccionar cómo se accede a la máquina.

🧩 Resultado final

✅ Terraform despliega una VM Windows 10 completamente funcional.
✅ Puedes conectarte por RDP sin configurar nada más.
✅ Toda la infraestructura es reproducible con “terraform apply”.
✅ Se integra perfectamente con flujos declarativos y locales.

📄 7. DETALLES DEL CONTENIDO DE LOS FICHEROS

📄 main.tf

```
# 🎛️ Declaramos que se usará el provider "vagrant", publicado por "bmatcuk"
terraform {
 required_providers {
   vagrant = {
     source  = "bmatcuk/vagrant"      # Fuente del provider en el Registry
     version = "4.1.0"                # Versión fija del provider
   }
 }
}
```

```
# 🔌 Activamos el provider vagrant (no necesita configuración extra)
provider "vagrant" {}

# 💻 Recurso que representa la VM de Windows 10
resource "vagrant_vm" "win10" {
 name      = "win10-eval"  # Nombre visible en VirtualBox y vagrant global-status

 # 💡 Este valor hace que la VM se regenere si el archivo Vagrantfile cambia
 env = {
   VAGRANTFILE_HASH = md5(file("${path.module}/Vagrantfile"))
 }

 # ⚠️ Activamos la lectura de puertos para exponerlos como output
 get_ports = true
}

# 📤 Output que muestra los puertos disponibles (como el RDP 3389)
output "puertos_rdp" {
 value = vagrant_vm.win10.ports
}
```

📄 Vagrantfile

```
Vagrant.configure("2") do |config|
 # 💻 Box pública de Windows 10 (versión de evaluación)
 config.vm.box = "gusztavvargadr/windows-10"

 # 📡 Usamos WinRM como protocolo de comunicación (requerido por Windows)
 config.vm.communicator = "winrm"

 # 🔁 Mapeamos el puerto RDP 3389 de la VM al host local (mismo puerto)
 config.vm.network "forwarded_port", guest: 3389, host: 3389

 # 🧰 Configuramos recursos de hardware con VirtualBox
 config.vm.provider "virtualbox" do |vb|
   vb.gui = true        # Mostrar la GUI (entorno gráfico)
   vb.memory = 2048     # Asignar 2 GB de RAM
   vb.cpus = 1          # Asignar 1 CPU
 end
end
```