Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Interfaz REST para la gestión de un entorno seguro privado

Autor: Marcelino Canovaca Bravo
Tutor: Germán Madinabeitia Luque

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Interfaz REST para la gestión de un entorno seguro privado

Autor:
Marcelino Canovaca Bravo

Tutor:
Germán Madinabeitia Luque
Profesor colaborador

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Grado: Interfaz REST para la gestión de un entorno seguro privado

Autor: Marcelino Canovaca Bravo

Tutor: Germán Madinabeitia Luque

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

A lo largo de estos años de esfuerzo y dedicación, he tenido la fortuna de contar con el apoyo de personas que han hecho posible la culminación de este trabajo. Quiero expresar mi más sincero agradecimiento a todos aquellos que han estado a mi lado en este viaje académico y personal.

En primer lugar, a mi familia. Gracias a mi padre y a mi madre por creer siempre en mí y por enseñarme el valor del trabajo duro y la perseverancia. A mi hermana, por acompañarme durante este camino, por su constante apoyo y por ser una fuente inagotable de motivación y alegría. A mis abuelos, cuyo amor y sabiduría han sido una fuente de inspiración constante. Sin vuestra comprensión y paciencia, este logro no habría sido posible.

A todas las personas que he conocido durante estos años, muchos de ellos ahora amigos, gracias por los momentos compartidos, por los viajes realizados, las risas y las largas noches de estudio que me ayudaron a crecer tanto personal como académicamente.

A mis compañeros y amigos del fútbol y baloncesto, que tan buenos momentos me han brindado, gracias por ser una válvula de escape y una manera de despejarme en los momentos más intensos y exigentes de la carrera. Sin esos ratos, esto tampoco hubiera sido posible.

Por último, y no menos importante, a las personas con las que más horas he compartido durante esta etapa, a los seis compañeros de piso con los que he convivido en el día a día. Gracias por esas charlas, esas risas y esos momentos compartidos que han hecho de estos años una experiencia inolvidable que siempre recordaré.

Marcelino Canovaca Bravo

Sevilla, 2024

Resumen

En el entorno actual, las redes de computadoras son cada vez más complejas y extensas. Tradicionalmente, la solución más común implementada por las empresas consiste en disponer de equipos y servidores físicos. Sin embargo, esto puede representar una gran inversión, tanto en términos de dinero como de tiempo, ya que cada equipo debe configurarse de manera individual, lo que puede acabar resultando tedioso.

Una posible solución a este problema es el uso del concepto de virtualización. Basándonos en este concepto y en el de *cloud computing*, ampliamente utilizados en la actualidad, este documento tiene como objetivo presentar una alternativa y una solución para el despliegue de un entorno privado prácticamente a *golpe de clic*. Esto incluye la creación de un número adecuado de máquinas y redes virtuales, así como la configuración de roles de usuarios para gestionar el acceso a estas. De esta manera, podemos crear una solución simple sin invertir excesivamente en equipos físicos y sin la necesidad de configurar la gestión manualmente, ya que utilizaremos herramientas de automatización como Ansible.

Abstract

Nowadays, computer networks are becoming increasingly complex and extensive. Traditionally, he most common solution implemented by companies is to have physical servers and equipment. However, this can represent a significant investment, both in terms of money and time, as each piece of equipment needs to be configured individually, which can end up being tedious.

One possible solution to this problem is the use of the concept of virtualization. Building on this concept and that of *cloud computing*, widely used today, this document aims to present an alternative and solution for deploying a private environment almost at the click of a button. This includes creating an appropriate number of virtual machines and networks, as well as configuring user roles to manage access to these machines. This way, we can create a simple solution without excessively investing in physical equipment and without the need to manually configure management, as we will use automation tools such as Ansible.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

API	Interfaz de Programación de Aplicaciones
REST	Transferencia de Estado Representacional
TI	Tecnología de la Información
SSH	Secure Shell
YAML	Otro Lenguaje de Marcado Más
JSON	Notación de Objeto JavaScript
XML	Lenguaje de Marcado Extensible
IP	Protocolo de Internet
HTTP	Protocolo de Transferencia de Hipertexto
HTML	Lenguaje de marcado de hipertexto
PHP	Preprocesador de Hipertexto
MVC	Modelo-Vista-Controlador
VPS	Servidor Privado Virtual
KVM	Máquina Virtual basada en Kernel
QEMU	Quick EMUlator
PPA	Repositorio de Paquetes Personal
PIP	Instalador de Paquetes de Python
NAT	Traducción de Direcciones de Red
DHCP	Protocolo de Configuración de direcciones Dinámicas de Host
XML	Lenguaje de Marcado Extensible
FTP	Protocolo de Transferencia de Archivos
RAM	Memoria de Acceso Aleatorio
LDAP	Protocolo Ligero de Acceso a Directorios
DN	Nombre Distinguido
CPU	Unidad Central de Procesamiento
MB	Megabytes
GB	Gigabytes
MAC	Control de Acceso a Medios

1 INTRODUCCIÓN

El verdadero signo de la inteligencia no es el conocimiento sino la imaginación.

- Albert Einstein -

En la era de la computación en la nube y la virtualización, el despliegue y la gestión eficiente de entornos virtuales privados se ha vuelto esencial para numerosas organizaciones. Estos entornos, que comprenden redes y recursos virtuales, requieren de herramientas sólidas, accesibles y fáciles de usar para su administración. El despliegue suele ser una tarea repetitiva que puede generar problemas, una menor productividad y finalmente pérdida de tiempo. En este contexto, una Interfaz de Programación de Aplicaciones (API) basada en el estilo REST es la opción elegida a desarrollar en este trabajo para agilizar el despliegue y gestión de estos entornos virtuales, debido a su escalabilidad, flexibilidad, simplicidad, portabilidad e independencia [1].

Este proyecto se enfoca en el diseño y desarrollo de una interfaz REST para la administración de redes virtuales, máquinas virtuales y usuarios, automatizando todo el proceso mediante la herramienta **Ansible.**

1.1 Objetivos

Aunque existen herramientas gráficas como **VMware** o **Virtual Box** para la creación de máquinas virtuales, y herramientas de línea de comandos como **virsh**, su uso puede plantear desafíos para el administrador, ya que requiere un conocimiento previo de su funcionamiento y puede implicar configuraciones complicadas a través de varios archivos.

El objetivo de implementar la API REST es permitir al administrador realizar fácilmente diversas tareas con solo unos pocos clics y la introducción de parámetros, eliminando la necesidad de comprender en detalle el funcionamiento de las herramientas mencionadas anteriormente.

Estas tareas que el administrador debería poder realizar fácilmente al finalizar el proyecto son:

- **Definición de redes virtuales:** Creación, modificación y borrado de una red virtual.
- **Definición de máquinas virtuales:** Creación, modificación y borrado de una máquina virtual.
- **Definición de roles de usuarios:** Creación y borrado de roles en forma de Unidad Organizativa en un servidor LDAP.

Para automatizar estas tareas, se empleará la herramienta **Ansible**, lo que garantizará una administración eficiente de los entornos virtuales.

1.2 API REST

Las API son conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones. Suele considerarse como el contrato entre el proveedor de información y el usuario, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta). En otras palabras, las API permiten interactuar con una computadora o un sistema para obtener datos o ejecutar una función, de manera que el sistema comprenda la solicitud y la cumpla. Una de las ventajas de las API es que no se necesita saber cómo se recibe el recurso ni de dónde proviene.

REST no es ni un protocolo ni un estándar, sino más bien un conjunto de límites de arquitectura.

Cuando el cliente envía una solicitud a través de una API REST, esta transfiere una representación del estado del recurso requerido a quien haya solicitado o al extremo. La información se entrega por medio de HTTP en uno de estos formatos: JSON, HTML, Python, PHP o texto sin formato.

Para que una API se considere REST, debe cumplir los siguientes criterios:

- Arquitectura cliente-servidor compuesta de clientes, servidores y recursos, con la gestión de solicitudes a través de HTTP.
- Comunicación entre el cliente y el servidor *sin estado*, lo cual implica que cada solicitud es independiente y está desconectada del resto.
- Datos que pueden almacenarse en caché y optimizan las interacciones entre el cliente y el servidor.
- Una interfaz uniforme entre los elementos, para que la información se transfiera de forma estandarizada. [2]

1.3 Escenario inicial

Con el propósito de proporcionar un contexto adecuado para los siguientes capítulos, se presenta un breve resumen de los actores involucrados en el escenario (Figura 1-1) del proyecto, con el fin de ofrecer al lector una visión general de los objetivos a alcanzar:

- **Administrador:** Esta figura desempeña el papel de responsable en el despliegue y configuración de redes y máquinas virtuales. Accederá a través de una interfaz, la cual realiza las peticiones a la API[1].
- **Servidor REST:** Este componente del sistema aloja y ejecuta el servicio web que hospeda la API desarrollada en el proyecto.
- **Nodo administrador:** Este equipo está equipado con la herramienta Ansible y es el encargado de ejecutar las acciones sobres los servidores de virtualización. En este caso particular, el servidor REST se encuentra alojado en el mismo equipo que funciona como nodo administrador.
- **Servidores de virtualización:** Estos nodos son los destinados para el despliegue y gestión de las redes y máquinas virtuales del proyecto.

[1] Se hace notar que este trabajo no abarca la parte de *Front-End,* siendo simulado mediante herramientas como Postman.

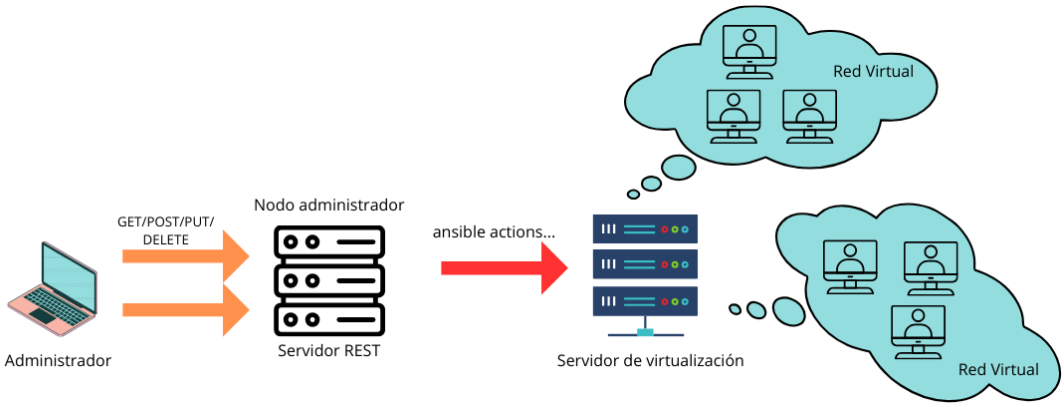


Figura 1-1. Escenario general del proyecto

1.4 Organización de la memoria

En este apartado se describe la estructura y distribución de la memoria, que consta de siete capítulos principales y un anexo final.

El **primer capítulo**, Introducción, presenta los objetivos del proyecto, proporciona una descripción general de lo que es una API REST, compara tecnologías similares a las del proyecto y describe el escenario inicial del trabajo.

El **segundo capítulo,** Tecnologías principales del proyecto**,** detalla las principales tecnologías utilizadas en el desarrollo del proyecto, destacando Ansible y Flask.

En el **tercer capítulo,** Configuraciones iniciales**,** se explican las primeras configuraciones necesarias para poner en marcha el proyecto. Esto incluye la instalación de Python, Ansible y otras herramientas, así como la configuración de roles, inventario y la generación de la clave pública SSH.

Los **capítulos cuatro**, **cinco** y **seis** están dedicados al desarrollo en sí del proyecto, cada uno correspondiente a los tres objetivos especificados en la sección Objetivos: Definición de redes virtuales, Definición de máquinas virtuales y Definición de roles de usuarios. Estos tres capítulos seguirán una estructura similar en sus subapartados.

Primero, se detallará la configuración del *playbook* correspondiente, luego se explicará el desarrollo del archivo principal de las tareas (*main.yml*), posteriormente se describirá la configuración de cualquier plantilla necesaria en caso de que así fuera y finalmente se explicará la configuración de las tareas de Ansible y el desarrollo en Python del código que atiende las solicitudes a la API para cada acción (creación, borrado y modificación).

En el **séptimo capítulo** se presentan los resultados obtenidos y se discuten las posibles mejoras que podrían implementarse de cara a un futuro, además de las conclusiones.

Asimismo, se incluye un Anexo al final de la memoria, donde se detallan los pasos a seguir para poner en marcha el proyecto, así como los requisitos mínimos de cada nodo involucrado.

2 TECNOLOGÍAS PRINCIPALES DEL PROYECTO

No es fracaso, son pasos hacia el éxito.

- Giannis Antetokounmpo -

En este capítulo se presenta un análisis detallado de las tecnologías fundamentales empleadas en el desarrollo y ejecución del presente trabajo. Se destacan especialmente dos: **Ansible** y la librería **ansible_runner** para Python, las cuales han sido piezas fundamentales de la automatización y gestión de los entornos virtuales. También se describe la sintaxis YAML, empleada en Ansible para configurar tareas y archivos necesarios. Asimismo, se aborda el uso del lenguaje de programación Python en combinación con el framework Flask para el desarrollo de la API REST, así como otro tipo de recursos usados durante las pruebas de ejecución.

Entre estos recursos o herramientas adicionales, se incluye Postman, una herramienta utilizada para agrupar y organizar las peticiones a la API, permitiendo su ejecución de manera estructurada. Además, se menciona la plataforma VPS Clouding, usada para configurar los servidores de virtualización y el servidor LDAP, sobre los cuales se ejecutarán las tareas lanzadas por Ansible.

También se proporciona una breve descripción del protocolo SSH, fundamental para que Ansible acceda de forma segura a los servidores remotos, y del protocolo OpenLDAP, empleado para crear el árbol de directorios de LDAP en el servidor remoto. Finalmente, se explican las tecnologías Libvirt y KVM/QEMU, elegidas para la creación y gestión de las máquinas virtuales en el proyecto.

2.1 Ansible

Ansible es una herramienta de automatización *open source* que automatiza el aprovisionamiento, la gestión de la configuración, la implementación de aplicaciones, la orquestación y muchos otros procesos de TI. Es de uso gratuito y el proyecto se beneficia de la experiencia y la inteligencia de sus miles de contribuyentes.

Una de las características clave de Ansible es su enfoque ***sin agente***. Esto significa que no requiere la instalación de software adicional en los nodos objetivo. En cambio, se basa en SSH[2] y utiliza conexiones seguras para comunicarse y ejecutar tareas en los sistemas remotos. Este diseño que presenta nos simplificará la configuración y el mantenimiento de nuestros sistemas, ya que no es necesario instalar y mantener agentes en cada nodo. [4]

En el núcleo de Ansible se encuentra una sintaxis sencilla y legible llamada **YAML.** Con archivos YAML, los

[2] SSH es un protocolo de red que tiene como función ofrecer acceso remoto a un servidor. La principal peculiaridad es que este acceso es seguro, ya que toda la información va cifrada.

usuarios pueden describir y definir la configuración deseada de los sistemas, servicios y aplicaciones en un entorno estructurado y fácil de entender. Esto permite a los equipos de TI definir su infraestructura y políticas operativas como código, lo que facilita la colaboración y el seguimiento de cambios. [4]

Como tecnología de automatización, Ansible está diseñado en torno a los siguientes principios: [5]

- **Arquitectura sin agentes:** La arquitectura sin agentes es una forma de automatizar y administrar los dispositivos de TI sin la necesidad de instalar agentes ni software adicional en los entornos gestionados.
- **Sencillez:** Los manuales de automatización o *playbooks* utilizan una sintaxis YAML sencilla. Ansible también está descentralizado y utiliza credenciales SSH del sistema operativo existente para acceder a máquinas remotas.
- **Escalabilidad y flexibilidad:** Escala fácil y rápidamente los sistemas que automatiza a través de un diseño modular que admite una amplia gama de sistemas operativos, plataformas en la nube y dispositivos de red.
- **Idempotencia y previsibilidad:** Cuando el sistema está en el estado que el *playbook* describe, Ansible no cambia nada, incluso si el *playbook* se ejecuta múltiples veces.

2.1.1 Sintaxis YAML

YAML es un lenguaje de serialización de datos que las personas pueden comprender y suele utilizarse en el diseño de archivos de configuración.

Es una sintaxis popular porque está diseñada para que sea fácil de leer y entender. Gracias a la flexibilidad y la accesibilidad que caracterizan a YAML, Ansible lo utiliza para crear procesos de automatización en forma de *playbook*s de Ansible.

YAML utiliza una extensión de archivos **.yml** o **.yaml** y sigue reglas de sintaxis específicas:

- No hay símbolos de formato habituales, como llaves, corchetes, etiquetas de cierre o comillas, y los archivos son más sencillos para su lectura, ya que utilizan la sangría y el número de espacios para determinar la estructura de los datos. Está diseñado para que no se admitan los caracteres de tabulación, por lo que los espacios en blanco son los que se usan para indicar jerarquía y anidamiento.
- Los comentarios en YAML se pueden definir con una almohadilla (#), no siendo compatibles los comentarios que tienen varias líneas.
- El inicio de un archivo YAML se señala con tres guiones (**- - -**) al inicio del documento y el final con tres puntos al final del documento (…).
- YAML emplea un formato sencillo de par clave-valor, separados por dos puntos.
- La estructura de un archivo YAML es un mapa (diccionario) o una lista y sigue una jerarquía según la sangría. Cada clave debe ser única y el orden no importa.
- Las listas incluyen valores en un orden específico y pueden contener cualquier cantidad de elementos según sea necesario. Una secuencia de lista se separa del elemento padre mediante sangría y cada elemento de la lista comienza con un guión (-) seguido de un espacio. Todos los elementos de la lista deben tener la misma distancia de sangría.

Uno de los usos más comunes de esta sintaxis es la creación de archivos de configuración. Se recomienda usar YAML en lugar de JSON[3] para escribir los archivos de configuración porque es un lenguaje más fácil de comprender. Podemos ver un ejemplo de las distintas sintaxis en la Figura 2-1.

Una de las ventajas de utilizarlo es que se pueden agregar los archivos a un control de versiones, como

[3] JSON es un formato ligero de intercambio de datos. Es de fácil lectura y escritura para los usuarios. Es fácil de analizar y generar por parte de las máquinas.

Github[4], para rastrear y auditar los cambios. [6]

XML	JSON	YAML

<Servers>
<Server>
<name>Server1</name>
<owner>John</owner>
<created>123456</created>
<status>active</status>
</Server>
</Servers>
	{
	Servers: [
	{
	name: Server1,
	owner: John,
	created: 123456,
	status: active
	}
	]
	}
		Servers:
		- name: Server1
		owner: John
		created: 123456
		status: active

Figura 2-1. Comparación YAML, XML y JSON [7]

2.1.2 Conceptos esenciales de Ansible

En esta sección se introducen una serie de conceptos fundamentales para comprender el funcionamiento y la estructura del proyecto, especialmente en el contexto de la automatización y gestión de configuraciones con Ansible. Entre esos conceptos, el **inventario** de Ansible organiza y define los hosts a gestionar, permitiendo una administración eficiente de múltiples sistemas. Los ***playbooks***, escritos en YAML, automatizan las tareas y configuran los sistemas al ejecutar secuencias de ***plays o jugadas***, que son conjuntos ordenados de tareas asignadas a nodos específicos. Cada **tarea** invoca un **módulo**, pequeñas unidades de código diseñadas para realizar acciones específicas en los hosts. Finalmente, los **roles** estructuran estos componentes en directorios organizados, mejorando la reutilización y la eficiencia en la gestión de las configuraciones. A continuación, se explican con más detalle estos elementos fundamentales de Ansible.

2.1.2.1 Inventario

Un inventario de Ansible no es más que una colección organizada de todas las máquinas que se quieren gestionar. Los inventarios organizan los nodos administrados en archivos centralizados que brindan a Ansible información del sistema y ubicaciones de red. Al utilizar un archivo de inventario, Ansible puede administrar una gran cantidad de hosts con un solo comando.

Este fichero es capaz de definir los hosts, organizarlos por grupos y generarlos por rangos.

Para crearlo se necesitará la dirección IP o el nombre de dominio completo de al menos un sistema host. También debe asegurarse de que su clave SSH pública se agregue al archivo *authorized_keys* en cada host.

Se puede definir utilizando dos formatos: por un lado, tenemos el texto plano (extensión **.ini**), y por el otro, el formato utilizado es el YAML. Aunque no se definan grupos en fichero del inventario, Ansible crea dos grupos por defecto: *all* y *ungrouped*. El grupo *all* contiene a cada host mientras que el grupo *ungrouped* contiene aquellos hosts que no pertenecen a ningún otro grupo más allá del *all*. Por ejemplo, en la Figura 2-2 se puede observar cómo el host *mail.example.com* pertenece al grupo *all* y a su vez al grupo *ungrouped*; el host *two.example.com* pertenece al grupo *all* y al grupo *dbservers*. Aunque los grupos *all* y *ungrouped* estén siempre presentes, pueden estar implícitos y no aparecer. [8] [9] [10]

[4] GitHub es un servicio basado en la nube que aloja un sistema de control de versiones llamado Git. Éste permite a los desarrolladores colaborar y realizar cambios en proyectos compartidos, a la vez que mantienen un seguimiento detallado del proceso.

```yaml
ungrouped:
  hosts:
    mail.example.com:
webservers:
  hosts:
    foo.example.com:
    bar.example.com:
dbservers:
  hosts:
    one.example.com:
    two.example.com:
    three.example.com:
```

Figura 2-2 . Ejemplo de inventario en formato YAML [10]

2.1.2.2 Playbooks

Ansible se basa en el concepto de *playbooks*, que son archivos de configuración (Figura 2-3) que describen el estado deseado del sistema y las tareas que deben ejecutarse para lograrlo. Es decir, son planos de automatización, escritos en formato YAML, que Ansible utiliza para implementar y configurar nodos administrados. Se pueden utilizar para configurar servidores, desplegar aplicaciones, realizar actualizaciones de software y mucho más. Esto permite a los administradores de sistemas automatizar de forma declarativa una amplia gama de tareas, lo que ahorrará tiempo y evitará la posibilidad de introducir errores dentro de nuestros sistemas. Básicamente, le indican a Ansible *lo que debe hacer y el dispositivo en el que debe hacerlo*.

En lugar de tener que aplicar la misma acción de manera manual a cientos o miles de tecnologías similares en todos los entornos de TI, se puede ejecutar un playbook y completar la acción de forma automática sobre el tipo de inventario que se especifique.

Los playbooks de Ansible son listas de **tareas** que se ejecutan automáticamente. Las tareas de Ansible se pueden combinar para crear un *play o **jugada***, un grupo ordenado de tareas que se asigna a hosts específicos, y las tareas se ejecutarán en el orden en el que se escriban.

Las tareas se ejecutan por ***módulos***, y cada uno de ellos lleva a cabo una de las tareas específicas que aparecen en el playbook. [4] [11] [12]

```yaml
- name: Instalar dependencias
  package:
    name: "{{ item }}"
    state: latest
  with_items:
    - libvirt-daemon-system
    - python3-libvirt
    - python3-lxml

- name: Obtener lista de las redes virtuales existentes
  virt_net:
    command: list_nets
  register: existing_nets

- name: Borrar la red virtual especificada
  virt_net:
    state: absent
    name: "{{ net_to_delete }}"
  when: "net_to_delete is in existing_nets.list_nets"
```

Figura 2-3. Ejemplo de playbook de Ansible

2.1.2.3 Plays

Las tareas de Ansible se pueden combinar para crear un *play*, un grupo ordenado de tareas que se asigna a hosts específicos. Es decir, una lista ordenada de tareas que se asigna a nodos administrados en un inventario.

El playbook puede incluir uno o más *plays*, así como funciones de Ansible.

En la Figura 2-4 podemos observar un playbook que actualiza de forma automática dos tipos de servidores. El playbook contiene dos *plays o jugadas:* [12] [11]

- El primero verifica si el servidor web está actualizado y, de lo contrario, ejecuta la actualización.
- El segundo verifica si el servidor de la base de datos está actualizado y, de lo contrario, ejecuta la actualización.

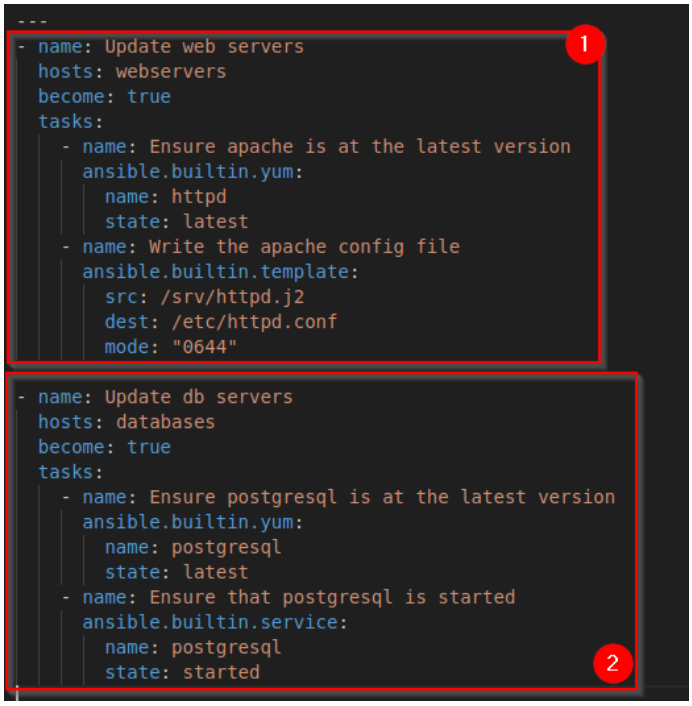


Figura 2-4. Ejemplo de plays o jugadas de Ansible

2.1.2.4 Tareas

Una tarea es una referencia a un único módulo que define las operaciones que realiza Ansible. Es decir, las tareas son los elementos que conforman los *plays* y realizan llamadas a los módulos de Ansible definiendo así la acción que se va a aplicar a un host gestionado. En un play, las tareas se ejecutan en el orden en que se escribieron. [11] [13].

En la Figura 2-4 podemos apreciar como cada *play* está compuesto por dos tareas, por tanto, el *play* está formado por 4 tareas en total:

- La primera tarea se asegura de que el servicio apache[5] se encuentre en su última versión, y en caso contrario, lo actualiza.
- La segunda tarea copia un archivo de configuración de apache en la ruta especificada.
- La tercera se asegura de que el servicio PostgreSQL[6] esté actualizado, en caso contrario, ejecuta la actualización.
- La cuarta tarea se asegura que el servicio PostgreSQL esté arrancado, y de lo contrario, lo arranca.

2.1.2.5 Módulos

Ansible se conecta a los nodos o hosts y les inserta pequeños programas denominados módulos. Cada play puede ejecutar una o más tareas, y cada tarea invoca un módulo de Ansible. Los módulos son una unidad de código binario que Ansible ejecuta en nodos administrados. Cada módulo utiliza un conjunto de parámetros

[5] Apache es un servidor web HTTP de código abierto.
[6] PostgreSQL es un Sistema de Gestión de Bases de Datos de código abierto.

configurables y una vez ejecutados Ansible los elimina al finalizar.

Analizándolo desde otro punto de vista: si las tareas son los trabajos que deben llevarse a cabo, los módulos son las herramientas que se necesitan para lograrlo. Una tarea define una acción que debe llevarse a cabo, el módulo se ejecuta en un host gestionado para realizar esa acción.

Sin los módulos se dependería de comandos específicos y de la creación de scripts para realizar cualquier tarea. Ansible contiene módulos integrados que pueden utilizare para automatizar las tareas.

Los módulos contienen metadatos que determinan el momento y el lugar en los que se ejecuta una tarea, así como el usuario que lo hace. [11] [13] [14]

En la Figura 2-4 podemos ver como se usa un módulo por cada tarea, siendo estos los módulos ***yum, template*** y ***service.***

2.1.2.6 Roles

Los roles son formas de cargar automáticamente variables, archivos, tareas, controladores y otros artefactos de Ansible relacionados en función de una estructura de archivos conocida. Permiten tener una estructura del proyecto más organizada y estructurada.

Un rol de Ansible tiene una estructura de directorios definida (Figura 2-5) con ocho directorios estándar principales. Se pueden omitir aquellos que no se utilicen.

De forma predeterminada, Ansible buscará roles en un directorio llamado *roles/*. Además, por defecto, Ansible también buscará en cada directorio dentro de una función un archivo *main.yml* para contenido relevante. [15]

- ***tasks/main.yml****:* Contiene la lista principal de tareas que ejecuta el rol.
- ***defaults/main.yml****:* Contiene variables predeterminadas para el rol. Estas variables tienen la prioridad más baja de todas las variables disponibles y pueden ser anuladas fácilmente por cualquier otra.
- ***templates/main.yml****:* Contiene plantillas que implementa el rol.

```
roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies
        library/          # roles can also include custom modules
        module_utils/     # roles can also include custom module_utils
        lookup_plugins/   # or other types of plugins, like lookup in this case

    webtier/              # same kind of structure as "common" was above, done for the webtier role
    monitoring/           # ""
    fooapp/               # ""
```

Figura 2-5. Estructura de directorios de un rol de Ansible

2.2 Ansible Runner

Ansible Runner es una herramienta y una biblioteca de Python que ayuda a la hora de interactuar con Ansible directamente o como parte de otro sistema, ya sea a través de una interfaz de imagen de contenedor, como una herramienta independiente o como un módulo de Python que se puede importar (nuestro caso). El objetivo es proporcionar una abstracción de interfaz estable y consistente para Ansible.

Parte de lo que hace que esta herramienta sea útil es que puede recopilar sus entradas de forma flexible. También tiene un sistema para almacenar la salida (*stdout*) y los artefactos (*datos de eventos a nivel de host, datos de hechos, etc)* de la ejecución del playbook.

La forma en la que estaremos usando Ansible Runner en este proyecto será a través de un módulo de Python, una interfaz de biblioteca. [16]

2.3 Python

Para el desarrollo de la API REST se usará el lenguaje de programación Python, principalmente porque Ansible está desarrollado en este lenguaje.

2.3.1 Flask

Flask es un *micro framework*[7] escrito en Python y concebido para facilitar el desarrollo de aplicaciones web bajo el patrón MVC. Al usar Flask tenemos las herramientas necesarias para crear una aplicación web funcional. Las ventajas de usar Flask antes que otros *frameworks* como Django son las siguientes: [17]

- Es un *micro framework*: Es útil para desarrollar de una forma ágil y rápida una aplicación básica.
- Incluye un servidor web de desarrollo: No se necesita una infraestructura con un servidor web.
- Tiene un depurador y soporte para pruebas unitarias.
- Es compatible con Python3.
- Buen manejo de rutas.
- Sirve para construir servicios web como APIs REST.
- Flask es Open Source y tiene buena documentación.

2.4 Postman

Postman es una herramienta utilizada para probar APIs, permitiendo a los desarrolladores enviar peticiones a servicios web y ver sus respuestas. Es útil ya que nos ofrece la posibilidad de guardar y agrupar conjuntos de solicitudes en las denominadas *colecciones*, es decir, simples carpetas en distintos niveles que organizarán nuestras peticiones de una manera más estructurada y ordenada. Podemos ver un ejemplo de esta estructura en la Figura 2-6.

[7] Los frameworks son herramientas que nos dan un esquema de trabajo y una serie de utilidades y funciones que nos facilita y nos abstrae de la construcción de páginas web dinámicas.

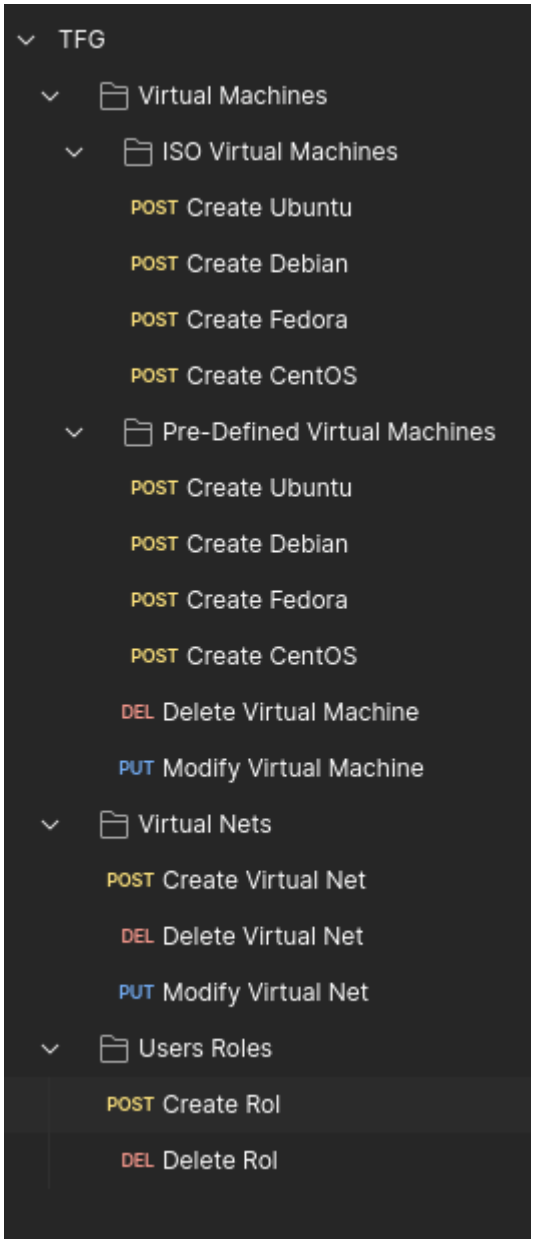


Figura 2-6. Interfaz gráfica de Postman

2.5 SSH

Es un protocolo para enviar comandos de forma segura a un equipo mediante una red no segura. SSH utiliza criptografía para autenticar y encriptar las conexiones entre dispositivos. Se utiliza a menudo para controlar los servidores a distancia, para gestionar infraestructuras y para transferir archivos. [18]

Ansible utiliza una arquitectura de servidor-cliente. Los nodos se conectan entre ellos gracias a este protocolo, lo que hace imprescindible su instalación en cada uno de ellos. Cada vez que se realiza una conexión remota a una máquina mediante SSH hay que introducir la contraseña de la máquina remota, por lo tanto, cada vez que se ejecute Ansible se debería introducir la contraseña de cada servidor. Esto puede ser una tarea pesada ya que cuanto mayor es el número de servidores a orquestar, mayor es el número de contraseñas que se deben introducir. La solución a este problema es añadir previamente la clave pública del servidor en los nodos clientes consiguiendo así que Ansible no requiera la introducción de la contraseña cada vez que sea ejecutado. La clave pública del servidor se debe de copiar en el fichero *authorized_keys*. [9]

2.6 OpenLDAP

Es un desarrollo del protocolo LDAP, implementado con la filosofía libre y código abierto. Está muy optimizado para ofrecer los mejores resultados en situaciones que requieran operaciones de lectura intensivas. De esta forma, un directorio OpenLDAP arrojará unos resultados muy superiores a los que ofrece una base de datos relacional optimizada, cuando realicemos operaciones de consulta intensivas sobre ambas. Lo usaremos para el capítulo de configurar los roles de los usuarios que podrán acceder a las máquinas virtuales que se creen.

2.7 VPS Clouding

Es una plataforma web (https://clouding.io/) que ofrece VPS. La usaremos para desplegar en ella el servidor de virtualización y el servidor LDAP sobre los que se realizarán todas las pruebas de las funciones de la API.

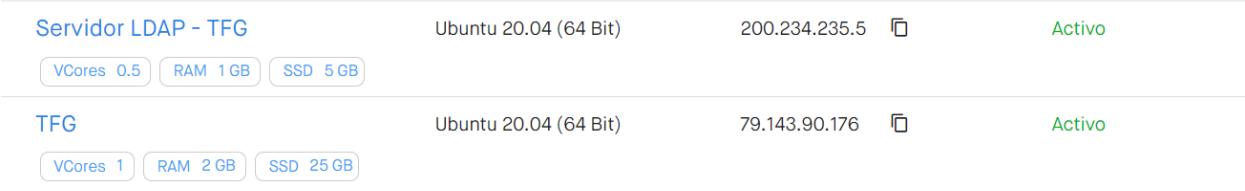


Figura 2-7. VPS Clouding

2.8 Libvirt y KVM/QEMU

Libvirt es un kit de herramientas para interactuar con las capacidades de virtualización de versiones recientes de Linux (y otros sistemas operativos). Proporciona gestión de máquinas virtuales, redes virtuales y almacenamiento; tanto locales como remotos. Dado que libvirt actúa como un intermediario entre un hipervisor y las aplicaciones cliente, debe tener instalado un hipervisor[8] compatible. En nuestro caso usaremos **KVM/QEMU.**

KVM es una solución de virtualización completa para Linux en hardware x86 que contiene extensiones de virtualización (Intel VT o AMD). Consiste en un módulo de kernel que se puede cargar, el cual proporciona la infraestructura de virtualización central y un módulo específico del procesador. [19]

QEMU es un emulador y virtualizador de máquinas genérico y de código abierto.

Cuando se utiliza como emulador de máquina, QEMU puede ejecutar sistemas operativos y programas creados para una máquina en una máquina diferente.

Cuando se utiliza como virtualizador, QEMU logra un rendimiento casi nativo al ejecutar el código invitado directamente en la CPU del host. QEMU admite la virtualización cuando se utiliza el módulo del kernel KVM en Linux. [20]

[8] Un hipervisor es un software que crea y ejecuta máquinas virtuales y que, además, aísla su sistema operativo y recursos de las máquinas virtuales y permite crearlas y gestionarlas.

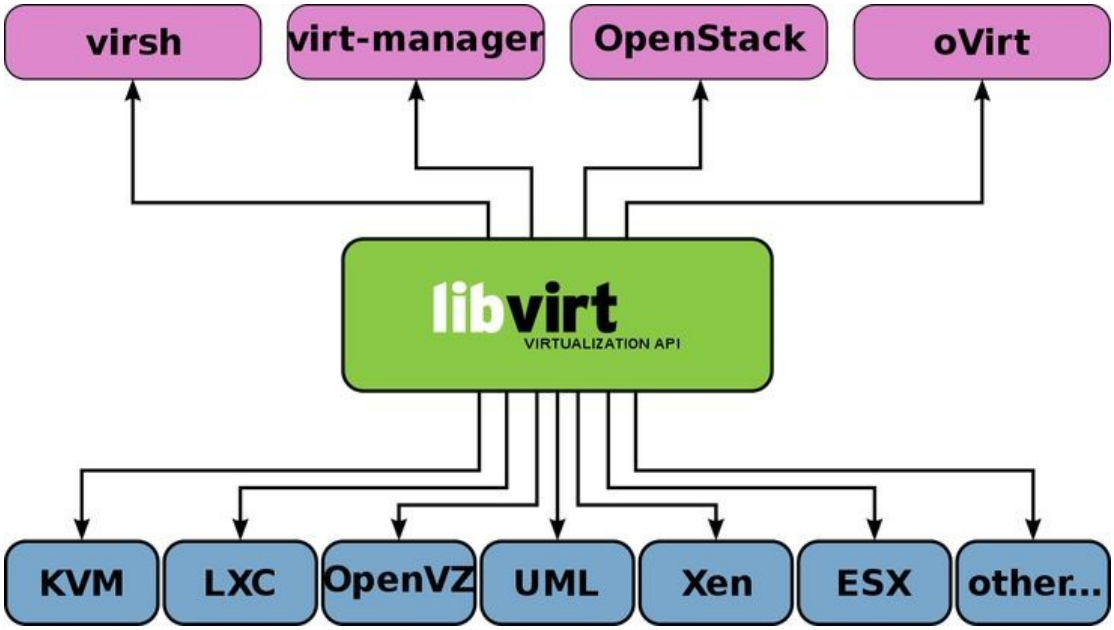


Figura 2-8. Esquema Libvirt [21]

2.9 Otras tecnologías: Contenedores vs Máquinas Virtuales

La virtualización surgió como una solución para mejorar la utilización de los recursos físicos y simplificar la gestión de sistemas heterogéneos mientras que los contenedores son una tecnología más reciente que ha ganado popularidad gracias a su eficiencia y portabilidad.

Como alternativa o complemento a este trabajo, se realizará una comparación (Tabla 2-1) entre la tecnología de los contenedores, una de las más destacadas en el contexto actual, y la de las máquinas virtuales, usada en el proyecto. Aunque a primera vista pueden parecer similares, ya que ambas ofrecen entornos computacionales aislados, presentan diferencias fundamentales en su arquitectura y aplicaciones.

Tabla 2-1. Diferencias entre contenedores y máquinas virtuales

Características	CONTENEDOR	MÁQUINA VIRTUAL
Definición	Un paquete de código de software que contiene el código de una aplicación y otras dependencias que conforman el entorno de ejecución de la aplicación.	Réplica digital de una máquina física. Divide el hardware físico en varios entornos.
Virtualización	Virtualiza el sistema operativo.	Virtualiza la infraestructura física subyacente.
Encapsulación	La capa de software por encima del sistema operativo necesaria para ejecutar la aplicación.	Sistema operativo, todas sus capas de software por encima, varias aplicaciones.
Tecnología	El motor de contenedores coordina los recursos con el sistema operativo subyacente.	El hipervisor se coordina con el sistema operativo subyacente o el hardware.
Tamaño	Más ligero (en términos de MB).	Más grande (en términos de GB)
Control	Menos control del entorno fuera del contenedor.	Más control sobre todo el entorno.

A modo de resumen, ambas tecnologías consisten en entornos informáticos empaquetados que combinan varios elementos de TI y los aísla del resto del sistema. La diferencia más importante radica en que ajustan su capacidad y en la portabilidad.

Los contenedores son conjuntos de uno o más procesos aislados del resto del sistema, y permiten que los procesos accedan solo a las solicitudes de recursos que se especificaron. Estas limitaciones de recursos garantizan que el contenedor pueda ejecutarse en un nodo con suficiente capacidad.

Las máquinas virtuales tienen su propio sistema operativo, lo que les permite realizar varias funciones con uso intensivo de los recursos al mismo tiempo. Las máquinas virtuales cuentan con una mayor cantidad de recursos disponibles, lo que les permite extraer, dividir, duplicar y simular sistemas operativos, escritorios, bases de datos, conexiones de red y servidores completos. [3]

3 CONFIGURACIONES INICIALES

Solo puede ser feliz siempre el que sepa ser feliz con todo.

- Confucio -

Este capítulo se adentra en las fases iniciales del proyecto, focalizándose en las configuraciones que influyen directamente en el **nodo administrador**, es decir, nuestro equipo donde desarrollaremos el trabajo. Abarcaremos desde la instalación de las herramientas esenciales como Ansible y Flask, hasta la meticulosa organización de directorios y roles y la generación del par de claves SSH, entre otras.

3.1 Instalación de Python

Dado que la API REST será desarrollada en Python, el servidor web de Flask lo lanzaremos con Python, siendo necesario garantizar que Python está instalado en el equipo. Generalmente, en sistemas Linux, viene preinstalado por defecto. Sin embargo, es importante tener en cuenta que *ansible_runner* requiere una versión de Python igual o superior a la 3.9 para funcionar correctamente. Por lo tanto, para asegurar la compatibilidad, podemos instalar la última versión disponible tanto de Python como de Flask con los comandos:

```
sudo apt install python3
pip install flask
```

3.2 Instalación de Ansible

Dependiendo del sistema operativo, la instalación de Ansible puede variar. En nuestro caso, al trabajar con Ubuntu, se requerirá realizar ciertos pasos específicos. Es importante tener en cuenta que será necesario agregar un PPA[9] al sistema para poder instalar Ansible en Ubuntu. [22]

```
sudo apt install software-properties-common
sudo add-apt-repository –yes –update ppa:ansible/ansible
sudo apt install ansible
```

[9] Un PPA es un repositorio de software diseñado para usuarios de Ubuntu y que es más fácil de instalar que otros repositorios de terceros.

3.3 Instalación de Ansible Runner

Una vez instalado Python y Ansible, existen varias formas de instalar Ansible Runner. En nuestro caso usaremos el administrador de paquetes de Python, PIP, para instalar la última versión. [23]

```
pip install ansible-runner
```

3.4 Instalación del cliente SSH

Para permitir que Ansible acceda a los servidores gestionados, se utilizará el cliente OpenSSH. Esto implica que los servidores gestionados deben contar también con el servicio instalado[10] y configurado correctamente. Al igual que Python, suele venir instalado por defecto en los sistemas Linux.

```
sudo apt install openssh-client
```

3.5 Generación de la clave pública

Como se ha mencionado en el capítulo 2.1, Ansible requiere acceso a los servidores sin la necesidad de ingresar una contraseña. Para lograr esto, es necesario que generemos una clave pública de nuestro equipo. Esta clave pública debe ser agregada al archivo *authorized_keys* (Figura 3-1) en cada servidor de virtualización a gestionar para permitir que Ansible acceda sin solicitar credenciales.

El comando utilizado para generar esta clave pública es `ssh-keygen`, el cual crea una clave pública y una privada. El contenido de la clave pública (*id_rsa.pub*) debe ser copiado y añadido al archivo *authorized_keys* dentro del directorio */home/usuario/.ssh* de cada servidor que se desea gestionar, siendo *usuario* el nombre de la cuenta de usuario del servidor de virtualización al cual queramos que Ansible pueda acceder. Una forma rápida de copiar las claves públicas generadas a los servidores remotos es usar el siguiente comando:

```
ssh-copy-id usuario@IP
```

Donde *usuario* hace referencia al usuario al que queremos poder acceder sin proporcionar credenciales e *IP* es la dirección del servidor remoto que se quiere gestionar.

Asimismo, debemos generar y copiar una clave pública de cada usuario de nuestro equipo que queramos que pueda acceder al servidor de virtualización. [4]

Es **importante** añadir las claves públicas al usuario *root* del servidor de virtualización ya que algunas tareas requieren privilegios y Ansible necesitará acceso a este usuario para ejecutarlas correctamente.

```
root@pruebas-tfg:~/.ssh# cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDGCPXum5/LQmZkMCMdS2Y3Hr52zCQ8hZFNjZQ8AxoJTNhw5NlNcY5/gTa22nu3eOWdIPch3Okstbi5gEBDLreTP3BeP2OMB
If9CNHVJg5qQX3fsYeYNYeNrtjQw58rRWj8WqJ+UJgEf+YrXyHbJyrJUHCxv5GaLnfmeixrIwtM+3yNjFUTl+Ib3vX8GO9LN/cX17DdOZXyDC2s5zoEhT/D9uE7u0vKfUlRbZ
ZGw0AkMmRUmW3oyE6YYLNhjqAnwpNNcv26ELWePaW0OM82G3uJ9J8ZQedhdkB6erQMjXbQCQ5Xhlno2sh0ohwwkTZZmLO93FoiOa/TzXnKflEjQUS1 Generated-by-Nova
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCqHoQOVGmxeICW5NK4kq6h3q9jeawppHjAmqCkvqDkPH/0Sn8BhFulFB+5Hqo8XkQ4NDT/Enom+I4T4Hu7q5l61o+gdxf/p
ZY07zAJ0ps75lc25cHV9jrIM/KeJnWmX/z6JYR2ORQSTrpcepShq4/FHYA//+hCBcVnSqVU7DJJdGOYzG7eR0JyCuEAxbrQJg0+qCI3i5jBLVIdpbH/25DwOXFr/K4o1bwTPk
bKlL4uuvYPTNkcSwHz3p3NB4dXwwUs1D9DshPRyIVG9o4uAH1Kw0RM/jOjBWDf6A7SN8XnNhLaYImk3lmQ0X2P/97S2l7T7lRQJjC3sTxVHk/uZ3hq0b8UTtTXbtNICAgBlyB
L8vb3v4ytj8TVfgvz7wvvS7GE7uaB5R6NxtWfNckAAWf78UM2P6n58sQskAGHPbs5HCib1qU4LJhCBVxnhhd1fXAO5Htk6qxeHAWSOQ6KWi/7eOHFZqoJhAI+/b+bJ2bhaz8r
WMHr5vjJMuEXLI6qZTk= marcelino@DESKTOP-0RG7KH0
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDieMYinY3406cHaUlDcNr28ctYWuroEExbSDVV7KsKC7lQbijKmV0wmP9dA1Fol4urZ9CZrE05HQDllqtc6Y+2aJrO2a6DQ
HenY+1ZY/pOTtbNPstWgnq8aPt7sk2HB7lTenD8QbVLBu6IXYV8ePXB6SDNrRjy62/sjBG2Shrrx1Z/9k5P2ZoZa3lx5IjZajpm+AUlnXUtRoQPVAoOSDn5kWI+Y08uyhwXhW
QLBYaEflvrtnhAOTvGX8ZbLC7F4Gqyakd8t/aVmCkN1VgDoaCynhfoQjKJDIZVRo2OCAXzuqIDwCvz3XVFe+Z1De5aeNw8eUGzxiTHzUmdkX8Fs/JZW4HUnfRqxfFUbxPDcHt
ZKx2p5o7Ic6WF9a14XhwQIeU7GyRcftNyszxYdy2RP0Vntj9bq1kKrfaPngA+FEwsCKK0FuEVqXyfRXKEUO/9DYJHZU0N3NVSLS1FVli8BTYkUhvm7B4paUUEu8YFEh9IcAP4
/iE1oKMAfnZhZpboSOc= dit@localhost
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCxfvq9LJfipnqnq41nJmk9rsweRIXjEYJMV2LHlKei9KiNvAOFIzhsU1ZD8mO22ZmPROKuBoLnDkH2hpYrMoTCWXP27R/fY
g2+XMJ1+isZ5khQe7ycABEa8JXf9hIYVvq5NezgGpqFWWuXOnuEciXRklJFnO4RsHY4tOHbhUxSJwgqneic3ZQAdkC7s4hMfL1vBdHym46UeJGmonOkftUevdAPJpS6bCozBC
rYVgooAGXMQX3/QVMPAkelh+sNN8GRTQcuk8aGJ8l5fUH05kPyXE8AutghZyllQo/2hz+303O/zGpQfBX1V40JbOuH/2kMh0Oc4I+CiVd34fZLTppXIA3oE3Hwqx/Gau5zO7p
ijvKQbC90Dsuj7WjOZ9uxqwOBHSh0WyKAuD/kHgW1swebsFECPcT97lsYXcj1gg/ygiR6yKBHlYbF96GB0QEokcH6B0vdcR9PGlUnvh86bunhDml7KdrQpGWtyqPvm+/UaPzY
9syc0kTdBY8CTOpOnc8= root@localhost
```

***Figura 3-1. Archivo authorized_keys del usuario root* del servidor de virtualización**

[10] La instalación y configuración del servicio SSH en los servidores de virtualización queda fuera del alcance del proyecto. Se presupone todo instalado y configurado correctamente.

3.6 Configuración de roles y estructura de directorios de Ansible

Conforme a los objetivos establecidos, se han definido tres roles que se reflejarán en la estructura de directorios del proyecto:

- Rol de **redes virtuales** (virt_nets)
- Rol de **máquinas virtuales** (virt_machines)
- Rol de **usuarios** (users)

Como se detalló anteriormente en **2.1.2.6 Roles**, un rol en Ansible sigue una estructura de directorios predefinida que incluye ocho directorios principales, mostrados en la Figura 2-5. No obstante, en nuestro proyecto solo definiremos tres de ellos, tal y como se puede ver en la siguiente figura:

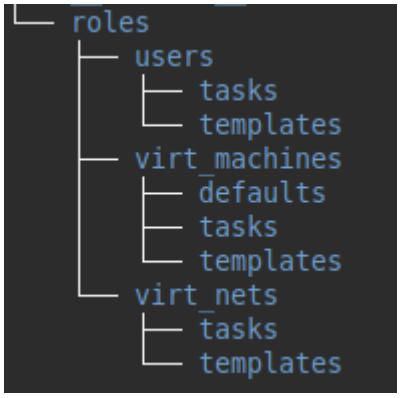


Figura 3-2. Estructura de directorio de los roles

Además, aparte de los roles, se encuentran otros directorios dentro del proyecto. Algunos son generados durante la ejecución, como *artifacts* y *env*, mientras que otros son creados por nosotros, como *inventory*. Por tanto, la estructura de directorios completa del proyecto se presenta en la siguiente figura:

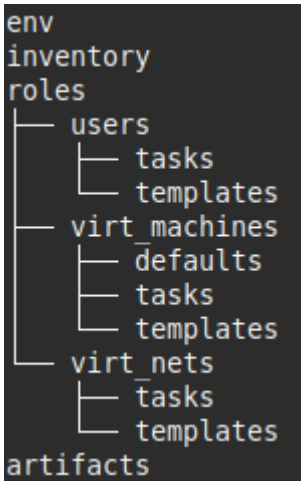


Figura 3-3. Estructura de directorios completa

3.7 Configuración del inventario

El archivo de configuración *hosts* (Figura 3-4) será ubicado dentro del directorio *inventory* del proyecto. Este archivo contendrá las direcciones IP o nombres de dominio de los servidores de virtualización y del servidor LDAP, permitiendo además la agrupación bajo nombres identificativos. En nuestro caso, como estamos usando un VPS en la nube asignamos el nombre identificativo *cloud* al grupo correspondiente. Este nombre podrá ser referenciado en el playbook para hacer referencia a todos los equipos agrupados dentro del grupo

cloud.

Adicionalmente, junto a la dirección IP del servidor a gestionar, se especificarán dos variables: [24]

- ***ansible_user:*** El nombre de usuario que se utilizará al conectarse al servidor.
- ***ansible_become_password:*** Define la contraseña[11] a ser utilizada cuando se ejecuten tareas con privilegios de superusuario mediante *sudo.*

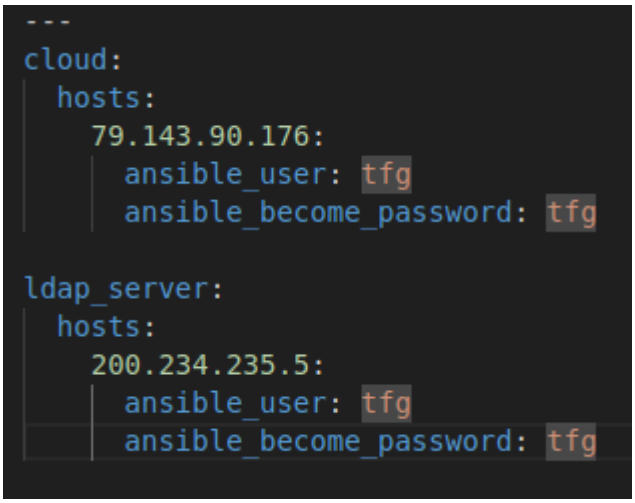


Figura 3-4. Archivo hosts del inventario

[11] Por motivos de seguridad, no es recomendable almacenar la contraseña en texto claro. Ansible proporciona métodos para cifrarla y almacenarla de manera segura, sin embargo, esto escapa del alcance de este proyecto.

4 DEFINICIÓN DE REDES VIRTUALES

La duda es la madre de la invención.

- Galileo Galilei -

Para que las máquinas virtuales se conecten a través de una red a su anfitrión, a otras máquinas virtuales en su anfitrión, y a ubicaciones en una red externa, la red de la máquina virtual debe ser configurada en consecuencia.

La red virtual utiliza el concepto de conmutador de red virtual. Un conmutador de red virtual es una construcción de software que opera en una máquina anfitriona. Las máquinas virtuales se conectan a la red a través del conmutador de red virtual.

La siguiente figura muestra un conmutador de red virtual que conecta dos máquinas virtuales a la red:

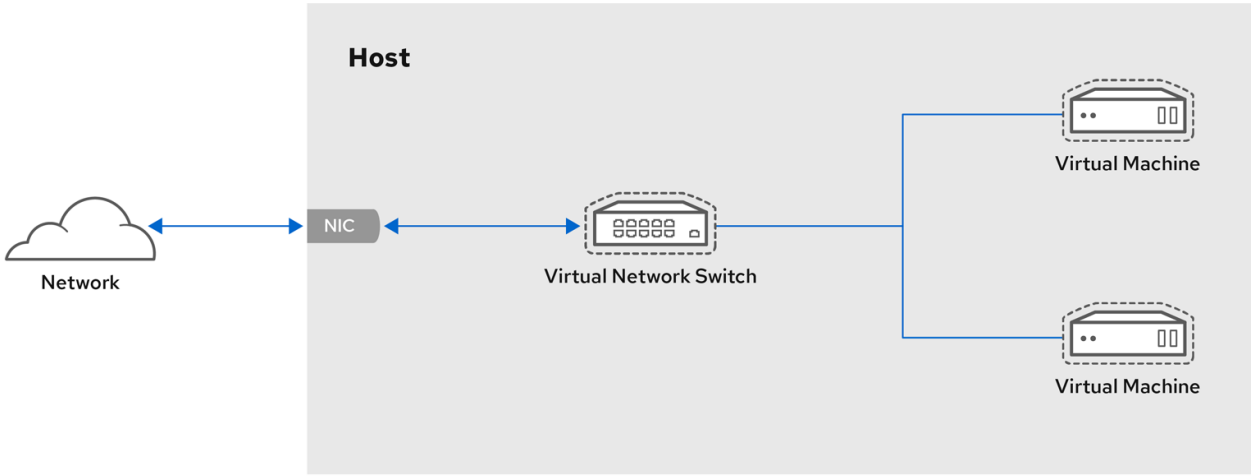


Figura 4-1. Conmutador de red virtual

Desde la perspectiva de un sistema operativo invitado, una conexión de red virtual es lo mismo que una conexión de red física. Las máquinas anfitrionas ven los conmutadores de red virtuales como interfaces de red. Cuando el servicio *libvirtd* se instala e inicia por primera vez, crea **virbr0**, la interfaz de red por defecto para las máquinas virtuales. De manera predeterminada, todas las máquinas virtuales de un mismo host están conectada a la misma red virtual de tipo NAT, denominada **default**, que utiliza la interfaz **virbr0**. Esta red por defecto se instala junto con el paquete *libvirt*, y se inicia automáticamente cuando se inicia el servicio *libvirtd.* [25]

El siguiente diagrama ilustra la configuración de red de la máquina virtual por defecto:

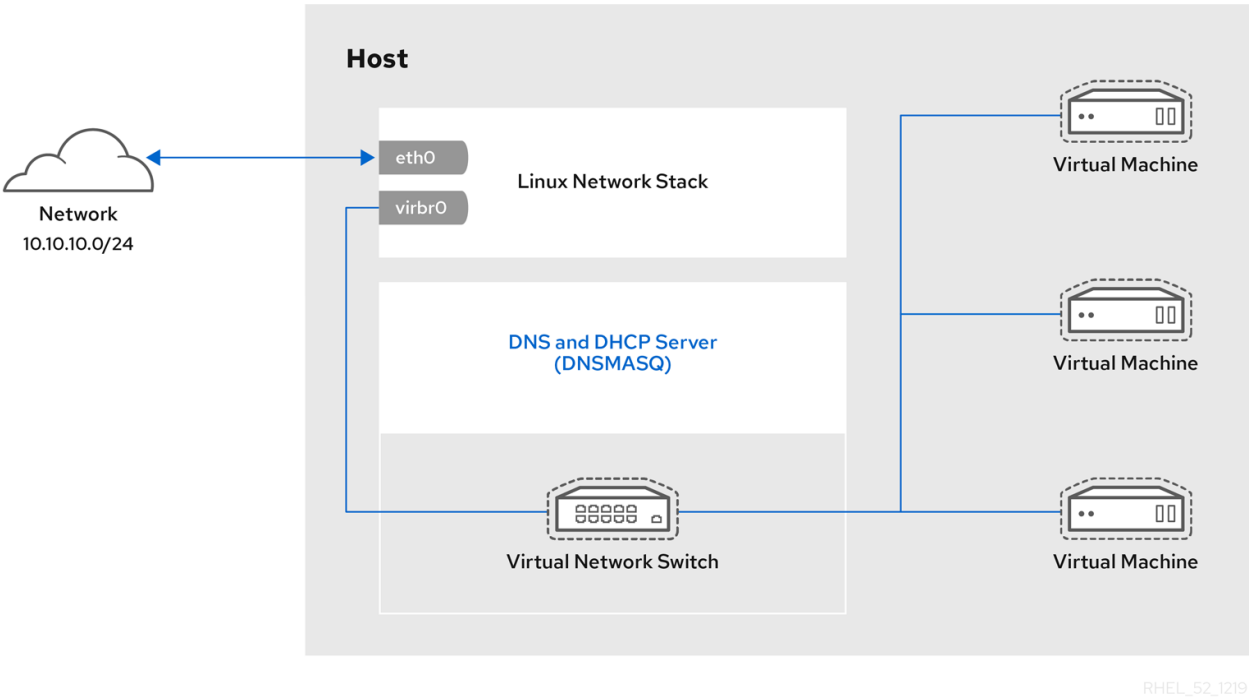


Figura 4-2. Configuración red por defecto libvirt

Se puede apreciar como el *switch virtual* dispone de un servidor DHCP (**dnsmasq**) que nos permite asignar IPs dentro del rango que configuremos a las máquinas invitadas que se conecten a esa red de forma automática. Para cada red se habilita una instancia de *dnsmasq* de forma que solo podrán acceder a ella las máquinas virtuales que estén configuradas para trabajar en esa red.

Además, al crear una red *libvirt* se crearán reglas *iptables* para permitir el tráfico al dispositivo virtual v*irbrX* y habilitará *ip_forward* en caso de que no se habilite. [26]

4.1 Configuración del playbook

Una vez estudiado el funcionamiento y comportamiento de las redes virtuales y cómo Libvirt facilita la gestión de estas, es momento de trasladar ese conocimiento en acciones concretas a través de Ansible.

Para gestionar de manera eficiente y estructurada las redes virtuales crearemos un playbook (Figura 4-3) que aprovecha el principio de reutilización de roles de Ansible, ya que crearemos uno por cada rol existente.

Para utilizar los roles usaremos la forma clásica según [15], usando la opción *roles* en el playbook. Cuando se usa esta opción a nivel de playbook, para cada rol que le indiquemos:

- Ansible agregará las tareas (*tasks/main.yml*) en ese archivo al playbook.
- Ansible agregará las variables por defecto (*defaults/main.yml*) en ese archivo.
- No será necesario indicar la ruta relativa o absoluta de las plantillas, Ansible las agregará de *templates.*

Además de esta opción, también añadimos las siguientes:

- ***name:*** Especifica el nombre identificativo del playbook.
- ***hosts:*** Indica los hosts sobre los que deseamos ejecutar las tareas que el playbook invoque. En este caso hacemos referencia al nombre identificativo *cloud*, correspondiente al grupo *cloud* del inventario. Ansible ejecutará las tareas correspondientes a todos los hosts que estén especificados dentro de este grupo en el inventario, en este caso, el servidor de virtualización.

- ***become:*** Al incluir esta opción como verdadera, nos aseguramos de que las tareas se ejecuten con privilegios de superusuarios.

```
---
- name: Configuración de las redes virtuales libvirt
  hosts: cloud
  become: true
  roles:
    - virt_nets
```

Figura 4-3. Playbook redes virtuales

4.2 Configuración del archivo *main.yml* de las tareas

A continuación, nos adentramos en la configuración específica del archivo **main.yml** de las tareas. Este archivo será buscado por Ansible una vez que se llame al rol de las redes virtuales en el playbook anterior. Estará localizado dentro del directorio de las tareas (*tasks*) e incluirá las tareas a ejecutar.

Sin embargo, orquestaremos las tareas en función de las variables *booleanas* ***create, modify*** y ***delete***. Estas variables, que se definen en tiempo de ejecución, guiarán la ejecución de las tareas para, como sus nombres indican, crear, modificar o eliminar redes virtuales según sea necesario.

Este archivo no contendrá en sí las tareas a realizar, si no que actuará como punto de entrada único y centralizado para todas las operaciones relacionadas con las redes virtuales. Esta estructura, nos permitirá mantener una organización clara de las tareas, distribuyéndolas en archivos específicos según su propósito. Es decir, en función del valor de las variables booleanas, se incluirán unas tareas u otras. Estas se dividirán en tres archivos distintos, agrupadas según la función a realizar:

- ***create.yml:*** Las tareas relacionadas con la creación de redes virtuales se lanzarán cuando la variable ***create*** sea **verdadera** y las otras dos falsas.
- ***delete.yml***: Las tareas relacionadas con el borrado de redes virtuales se lanzarán cuando la variable ***modify*** sea **verdadera** y el resto falsas.
- ***modify.yml***: Las tareas relacionadas con el modificado de una red virtual se lanzarán cuando la variable ***modify*** sea **verdadera** y las otras falsas.

Por tanto, la estructura de archivos dentro del directorio de las tareas de este rol queda definida de la siguiente manera:

```
virt_nets/
└── tasks
    ├── create.yml
    ├── delete.yml
    ├── main.yml
    └── modify.yml
```

Figura 4-4. Archivos de las tareas de las redes virtuales

Siendo el archivo *main.yml* el mostrado en la Figura 4-5:

```yaml
---
# Definimos las tareas relacionadas con las redes virtuales y las condiciones de su ejecución
- name: Creación de la red virtual libvirt
  include_tasks: create.yml
  when: create | bool and not modify and not delete

- name: Modificación de una red virtual
  include_tasks: modify.yml
  when: modify | bool and not create and not delete

- name: Borrado de una red virtual
  include_tasks: delete.yml
  when: delete | bool and not create and not modify
```

Figura 4-5. Contenido del archivo main.yml de las tareas del rol de redes virtuales

Con esta organización, garantizamos una gestión flexible y dinámica de las redes virtuales.

4.3 Configuración de la plantilla de red XML

En este proceso de configuración de redes virtuales en un entorno de virtualización, es necesario proporcionar a Ansibe un archivo XML (más adelante se desarrolla el porqué) que defina las características específicas de la red a crear en Libvirt. Sin embargo, hay elementos como el nombre de la red y el prefijo que pueden variar según lo proporcionado por el administrador de red, por tanto, es necesario usar un método dinámico y flexible que permita generar esta configuración.

En nuestro caso recurrimos a la combinación de **Jinja2**[12] y **XML** para abordar esta necesidad.

Esta plantilla (Figura 4-6), en formato **xml.j2** estará alojada dentro del directorio *templates* del rol de redes virtuales. Contendrá la estructura básica de un archivo XML para crear una red virtual, junto con expresiones Jinja2 para aquellos elementos que necesitan ser completados dinámicamente. Usando como base la plantilla del proyecto de [27] crearemos la siguiente plantilla para las redes que se vayan a crear:

```
<network>
    <name>{{ vnet_name }}</name>
    <forward mode='nat'>
        <nat>
            <port start='1024' end='65535'/>
        </nat>
    </forward>
    <bridge name="br-{{ vnet_name }}"/>
    <ip address="{{ vnet_prefix }}.1" netmask="255.255.255.0">
        <dhcp>
            <range start="{{ vnet_prefix }}.2" end="{{ vnet_prefix }}.254"/>
        </dhcp>
    </ip>
</network>
```

Figura 4-6. Plantilla xml.j2 de una red virtual

Donde:

- **vnet_name:** Hace referencia a una variable que se le pasará en tiempo de ejecución correspondiente al nombre de la red. Este nombre de la red también se usará para crear el nombre de la interfaz de red virtual de la red a crear (todas empezarán con *br-* seguido del nombre de la red). Además, el nombre

[12] Jinja2 es una poderosa biblioteca de plantillas para Python que permite la generación dinámica de texto. Al integrarlo con XML, podemos automatizar la creación de las redes de una manera dinámica mediante el uso de variables.

de la red debe ser único dentro de cada host para evitar conflictos en la infraestructura de virtualización, es decir, no puede haber dos redes con el mismo nombre.

- **vnet_prefix:** Hace referencia a una variable que se le pasará en tiempo de ejecución correspondiente al prefijo de la red a crear. Se usará para establecer la dirección IP de la puerta de enlace predeterminada y del servidor DHCP. Asimismo, se usará para establecer el rango de direcciones IPs que asignará el servidor DHCP a las máquinas que se conecten a la red. Para simplificar el proceso, establecemos que todas las máscaras serán **/24.**

4.4 Creación de redes virtuales

4.4.1 Desarrollo Python

Siguiendo las pautas y recomendaciones de [28] para la creación de una API REST con Flask, se ha implementado una función para cada acción relacionada con las redes virtuales. A continuación, se detallan las características de la solicitud (Tabla 4-1) y el código en Python (Figura 4-7, Figura 4-8, Figura 4-9 y Figura 4-10) del servidor encargado de su gestión:

Tabla 4-1. Características de la petición de crear una red virtual

Método HTTP	POST
Endpoint	/createVNET
Parámetros de la petición	**Nombre** de la red a crear
	Prefijo de la red a crear

Se establece que la longitud del nombre de la red no debe exceder los 13 caracteres debido a un límite en el nombre de la interfaz de red que se creará. Por lo tanto, se verifica este paso.

```python
# Petición POST para la creación de una red virtual
@app.post("/createVNET")
def createVNET():

    # Características de la red a crear (proporcionadas por el usuario)
    prefijo = request.form.get("prefijo")
    nombre = request.form.get("nombre")

    if prefijo is None or nombre is None:
        abort(400,"Faltan campos en la solicitud POST")
    elif len(nombre) >= 13:
        abort(400,"El nombre de la red debe tener 12 caracteres como máximo")
```

Figura 4-7. Código Python creación de redes virtuales

Creamos un diccionario (Figura 4-8) de variables que se le pasarán al playbook de ansible. En este caso la variable *create* será verdadera y el resto falsas. Además, introduciremos también el nombre de la red y el prefijo proporcionados en la petición.

```python
# Configuramos la variable de creación como verdadera en el diccionario
variables = {"create":True,"modify":False,"delete":False, "vnet_name":nombre, "vnet_prefix":prefijo}
```

Figura 4-8. Diccionario de variables Python creación de redes virtuales

Siguiendo los ejemplos proporcionados por [16], se utiliza la biblioteca importada de Ansible Runner para ejecutar Ansible, tal y como se puede observar en la Figura 4-9. Esto se logra mediante la función *run* de dicha

biblioteca, que toma las entradas y ejecuta Ansible en primer plano. Al finalizar, devuelve el objeto Runner. Al llamar a esta, se especifican las siguientes opciones:

- ***private_data_dir***: Especificamos el directorio que contiene los metadatos del runner necesarios para invocar el módulo de runner. En este caso, indicamos el directorio actual de trabajo desde donde se ejecuta el servidor.
- ***playbook***: Se especifica el playbook que será invocado por el runner al ejecutar Ansible.
- ***inventory***: Se proporciona la ruta absoluta al fichero de inventario.
- ***extravars***: Se pasa el diccionario de variables adicionales que se desean utilizar en la ejecución de Ansible.

```
# Ejecutamos el playbook que crea la red virtual con ansible runner
runner = ansible_runner.run(private_data_dir=".", playbook="virt_nets_playbook.yml",
                            extravars=variables, inventory="inventory/hosts")
```

Figura 4-9. Uso de Ansible Runner en Python para creación de redes virtuales

Una vez ejecutado, usamos el objeto Runner devuelto para acceder a los resultados de la ejecución de Ansible [29] y así gestionar los posibles errores (Figura 4-10) que puedan darse: nombre de red a crear ya en uso o prefijo de red ya en uso.

```
for events in runner.events:
    #print(events["event"] + ":" + events["event_data"]["task"])
    if(events["event"] == "runner_on_failed"):
        task_failed = events["event_data"]["task"]
        error_msg = events["event_data"]["res"]["msg"]
        print(error_msg)
        response = "Falló la tarea <<" + task_failed + ">> debido al siguiente error: " + error_msg
        abort(500,response)
    elif events["event"] == "runner_on_skipped" and events["event_data"]["task"] == "Definir red virtual si no existe":
        response = "Nombre de la red virtual ya existe en el servidor"
        abort(500,response)
    else:
        response = "Red " + str(nombre) + " creada correctamente con prefijo " + str(prefijo) + ".0/24"

return response
```

Figura 4-10. Gestión de errores en Python de la creación de redes virtuales

4.4.2 Configuración de las tareas de Ansible

Como se describió en capítulos anteriores, cuando el valor de la variable ***create*** es verdadero, se incluyen en la ejecución las tareas correspondientes a la creación de redes virtuales.

La creación de las redes virtuales implica un proceso más complejo que simplemente ejecutar un módulo de Ansible y proporcionar un archivo XML. Es un proceso que requiere la instalación de dependencias y requisitos indispensable para que no haya errores, así como diferentes comprobaciones adicionales durante la ejecución de las tareas para garantizar un despliegue sin errores. A continuación, detallamos las etapas clave de este proceso: [27]

1. Instalación de los requisitos necesarios para la creación de las redes virtuales en el servidor de virtualización (Figura 4-11):
    a. Instalación del demonio *libvirt*.
    b. Instalación del paquete *python3-libvirt*, que proporciona una API para interactuar con el hipervisor de virtualización Libvirt desde programas escritos en Python.
    c. Instalación del paquete *python3-lxml*, una biblioteca de Python que proporciona clases y métodos para analizar documentos XML y HTML, así como para navegar y manipular sus elementos, atributos y contenido.
    d. Módulo de Ansible empleado para esta tarea:
        i. ***Package***: Este módulo administra paquetes en un destino determinado sin necesidad de especificar un administrador de paquetes concreto, como *yum* o *apt*. Esto elimina

la necesidad de crear una tarea específica para cada administrador de paquetes ya que se adapta automáticamente al administrador de paquetes específico del sistema operativo donde se esté ejecutando. [30]

```
---
- name: Instalar dependencias
  package:
    name: "{{ item }}"
    state: latest
  with_items:
    - libvirt-daemon-system
    - python3-libvirt
    - python3-lxml
```

Figura 4-11. Tarea instalar dependencias al crear red virtual

2. Obtención de una lista de las redes virtuales existentes en el servidor de virtualización (Figura 4-12):
    a. Guardaremos la lista en una variable que será usada posteriormente.
    b. Módulo de Ansible empleado para esta tarea:
        i. ***Virt_net:*** Módulo usado para gestionar las redes virtuales libvirt. Comando *list_nets.* [31]

```
- name: Obtener lista de redes virtuales existentes
  virt_net:
    command: list_nets
  register: existing_nets
```

Figura 4-12. Tarea de obtención de las redes virtuales existentes

3. Definición de la red virtual **cuando** el nombre de la red a crear **no exista** previamente (Figura 4-13):
    a. Verificación de si el nombre de la red a crear ya existe en la lista de redes virtuales obtenida anteriormente, en caso afirmativo esta tarea no se ejecuta.
    b. Uso de la plantilla XML que define la red.
    c. Módulo de Ansible usado:
        i. ***Virt_net:*** Comando ***define*** y uso del parámetro ***xml*** donde se le indica la plantilla XML a usar para definir la red.

```
- name: Definir red virtual si no existe
  virt_net:
    name: "{{ vnet_name }}"
    command: define
    xml: "{{ lookup('template', 'virt_net.xml.j2') }}"
  when: "vnet_name not in existing_nets.list_nets"
```

Figura 4-13. Tarea definición red virtual

4. Activación de la nueva red creada (Figura 4-14):
    a. Registramos el resultado de esta tarea en una variable para su posterior uso.
    b. En caso de que falle la tarea, usamos el parámetro ***ignore_errors*** para permitir la continuación de la ejecución de tareas posteriores.
    c. Módulo de Ansible:

i. ***Virt_net:*** Parámetro ***state*** con el valor *active* para activar la red.

```
- name: Asegurar que la nueva red virtual está activa
  virt_net:
    name: "{{ vnet_name }}"
    state: active
  register: vnet_results
  until: "vnet_results is success"
  ignore_errors: true # Permitimos que siga la ejecución de las tareas aunque falle
  retries: 2
  delay: 2
```

Figura 4-14. Tarea activar red virtual

En este momento de la ejecución, se permite la continuación del proceso en caso de error, ya que si el administrador proporciona un prefijo de red que ya está en uso por otra red, surgirá un error al intentar activarla. A pesar de este fallo, la red quedará definida (creada) en el paso anterior, pero permanecerá inactiva debido a la invalidez del prefijo. Para evitar dejar esa red no válida definida, crearemos una tarea adicional que la elimine en el caso de que la activación falle. Por lo tanto, utilizamos una variable para almacenar el resultado de la tarea de activación y tomar en este momento la decisión correspondiente en consecuencia:

5. Borrado de la red definida en caso de fallo por uso de un prefijo en uso (Figura 4-15):
    a. Comprobamos si ha fallado la tarea anterior haciendo uso de la variable que contiene el resultado de la activación.
    b. Módulo de Ansible utilizado:
        i. ***Virt_net:*** Uso del parámetro ***state*** con el valor ***absent*** para eliminar la red.

```
- name: Borrar la red definida en caso de fallo por uso del mismo prefijo
  virt_net:
    name: "{{ vnet_name }}"
    state: absent
  when: "vnet_results is failed"
```

Figura 4-15. Tarea de borrado de la red virtual en caso de error

6. Configuración del arranque automático de la red (Figura 4-16):
    a. Asegurar de que la red se activará automáticamente en el arranque del servidor de virtualización, solo si la activación anterior ha sido exitosa.
    b. Módulo Ansible utilizado:
        i. ***Virt_net:*** Parámetro ***autostart*** con el valor ***yes***.

```
- name: Asegurar que la red se activará en el momento del arranque
  virt_net:
    name: "{{ vnet_name }}"
    autostart: yes
  when: "vnet_results is success"
```

Figura 4-16. Tarea de autoarranque de la red virtual creada

4.5 Borrado de redes virtuales

4.5.1 Desarrollo Python

Tabla 4-2. Características de la petición de borrar una red virtual

Método HTTP	DELETE
Endpoint	/deleteVNET
Parámetros de la petición	**Nombre** de la red a borrar

Se inicia el proceso con la identificación del nombre de la red que el administrador pretende eliminar, asegurándonos de que la longitud de dicho nombre no exceda los 13 caracteres.

```python
# Petición DELETE para el borrado de una red virtual
@app.delete("/deleteVNET")
def deleteVN():

    # El usuario nos pasa el nombre de la red a borrar
    nombre = request.form.get("nombre") # También inferior a 13 caracteres

    if nombre is None:
        abort(400,"Faltan parámetros en la solicitud POST")
    elif(len(nombre) > 13 ):
        abort(400,"El nombre de la red a borrar debe tener 12 caracteres como máximo")
```

Figura 4-17. Código Python borrado de redes virtuales

Posteriormente, se procede a la creación de un diccionario de variables (Figura 4-18) destinadas a ser utilizadas por Ansible durante la ejecución del proceso. En este caso, se establece el valor de la variable *delete* como verdadero, indicando la intención de eliminar la red proporcionada.

```python
# Configuramos las variables que deciden que se ejecute la tarea de borrado y el nombre de la red a borrar
variables = {"create": False,"modify": False,"delete": True, "net_to_delete": nombre}
```

Figura 4-18. Diccionario de variables Python borrado de redes virtuales

Una vez configuradas las variables necesarias, se ejecuta Ansible Runner con las entradas proporcionadas, y se almacena el objeto Runner en una variable para su posterior análisis. Mediante esta variable, se verifica la correcta ejecución de todas las operaciones planificadas. En caso de error, se devolverá un código de estado no exitoso en la respuesta a la petición.

```python
# Ejecutamos el playbook que elimina la red virtual con ansible runner
runner = ansible_runner.run(private_data_dir=".", playbook="virt_nets_playbook.yml",
                            extravars=variables, inventory="inventory/hosts")

for events in runner.events:
    if(events["event"] == "runner_on_failed"):
        task_failed = events["event_data"]["task"]
        error_msg = events["event_data"]["res"]["msg"]
        response = "Falló la tarea <<" + task_failed + ">> debido al siguiente error: " + error_msg
        abort(500,response)
    elif(events["event"] == "runner_on_skipped" and events["event_data"]["task"] == "Borrar la red virtual especificada"):
        abort(500,"Red virtual proporcionada no existe")
    else:
        response = "Red " + str(nombre) + " borrada correctamente"

return response
```

Figura 4-19. Gestión de errores en Python del borrado de redes virtuales

4.5.2 Configuración de las tareas de Ansible

El proceso de eliminación de una red virtual se desencadenará una vez que el archivo principal reciba la señal de que la variable ***delete*** ha sido establecida como verdadera. El procedimiento de borrado de una red virtual comprende una serie de tareas específicas, que incluyen:

1. Instalación de dependencias (las mismas que al crear la red virtual, Figura 4-11).
2. Obtención de una lista de las redes virtuales existentes, guardándola en una variable para su posterior

uso (exactamente igual que al crear la red virtual, Figura 4-12).

3. Borrado de la red virtual (Figura 4-20):
    a. Cuando la red a borrar esté dentro de la lista de redes existentes significará que existe y por tanto se puede borrar. En caso contrario, esta tarea será saltada por Ansible.
    b. Módulo de Ansible empleado:
        i. ***Virt_net:*** parámetro ***state*** y valor ***absent.***

```
- name: Borrar la red virtual especificada
  virt_net:
    state: absent
    name: "{{ net_to_delete }}"
  when: "net_to_delete is in existing_nets.list_nets"
```

Figura 4-20. Tarea de borrar una red virtual

4.6 Modificación de redes virtuales

4.6.1 Desarrollo Python

Tabla 4-3. Características de la petición de modificar red virtual

Método HTTP	PUT
Endpoint	/modifyVNET
Parámetros de la petición	**Nombre** de la red a modificar
	Nombre **nuevo**
	Prefijo nuevo

En este proceso de modificación, se han definido dos parámetros susceptibles de modificación en una red: el nombre y el prefijo. Por lo tanto, la solicitud (Tabla 4-3) debe contener el nombre de una red existente, el nuevo prefijo de red deseado y el nuevo nombre que se pretende asignar a la red existente. En caso de no proporcionar un nuevo nombre, esto indica la intención de no modificar este parámetro de la red, por lo que el valor de la variable asociada a nuevo nombre será el mismo que el nombre original de la red a modificar. Asimismo, se vuelve a verificar que la longitud de ambos nombres sea inferior a 13 caracteres.

```python
# Petición PUT para la modificación de una red virtual
@app.put("/modifyVNET")
def modifyVNET():

    # El usuario nos pasa el nombre de la red a modificar y los datos a modificar.
    new_name = request.form.get("nombre_nuevo")
    nombre = request.form.get("nombre")
    prefijo = request.form.get("prefijo")

    if nombre is None or prefijo is None:
        abort(400,"Faltan datos en la solicitud POST")
    elif new_name is None:
        new_name = nombre

    if(len(new_name) > 13 or len(nombre) > 13):
        abort(400,"El nombre de la red a borrar debe tener 12 caracteres como máximos")
```

Figura 4-21. Código Python modificar red virtual

Seguidamente, se procede a la creación del diccionario de variables (Figura 4-22), estableciendo el valor de la variable *modify* como verdadero, y se pasan las variables con los valores recibidos en la solicitud.

```python
# Variable que decide que acción se realiza
variables = {"create": False, "modify": True, "delete": False,
             "net_to_modify": nombre, "vnet_prefix":prefijo, "vnet_name": new_name}
```

Figura 4-22. Diccionario de variables modificar red virtual

Se utiliza nuevamente la función *run* de la biblioteca de Ansible Runner con las mismas entradas anteriores. Aprovechando el objeto Runner recibido, se realizan las verificaciones necesarias para comprobar si el nombre de la red proporcionado no existe, si el nuevo nombre ya está en uso o si el prefijo ya está asignado a otra red.

```python
# Ejecutamos el playbook
runner = ansible_runner.run(private_data_dir=".", playbook="virt_nets_playbook.yml", extravars=variables, inventory="inventory/hosts")

for events in runner.events:
    if(events["event"] == "runner_on_skipped" and events["event_data"]["task"] == "Borrar la red virtual a modificar"):
        #task_failed = events["event_data"]["task"]
        #error_msg = events["event_data"]["res"]["results"][0]["msg"]
        #response = "Falló la tarea <<" + task_failed + ">> debido al siguiente error: " + error_msg
        #response = "Red a modificar no existe"
        abort(500,"Red a modificar no existe o nueva red a crear ya tiene el nombre proporcinado asociado")
    elif(events["event"] == "runner_on_skipped" and events["event_data"]["task"] == "Definir red virtual con nuevo nombre si no existe"):
        abort(500,"Error: Ya existe una red con el nuevo nombre proporcionado")
    elif(events["event"] == "runner_on_failed" and events["event_data"]["task"] == "Asegurar que la nueva red virtual está activa"):
        abort(500,"Error: El nuevo prefijo solicitado ya está en uso en otra red")
    else:
        response = "Red " + str(nombre) + " modificada correctamente a " + str(new_name) + " y nuevo prefijo: " + str(prefijo) + ".0/24"

return response
```

Figura 4-23. Gestión de errores Python modificar red virtual

4.6.2 Configuración de las tareas de Ansible

La modificación de una red virtual se ejecutará bajo la premisa de interpretarla como una operación compuesta por la eliminación de la red existente y la creación de una nueva. Esta metodología se justifica por la limitación identificada en el comando ***modify*** del módulo **virt_net** [31] de Ansible, el cual no se encuentra completamente implementado en la herramienta utilizada. La falta de implementación de este comando ha generado inconvenientes, ya que no es reconocido como una instrucción válida, lo que genera la necesidad de abordar este proceso de manera alternativa.

Este procedimiento requiere la provisión de información específica por parte del administrador, como son el nombre de la red que se pretende modificar (y por ende eliminar en este contexto), el nuevo nombre de la red modificada (y, por ende, creada) y el prefijo de la red en el caso de que también se quiera modificar.

Por tanto, este proceso incluye la combinación de las tareas de borrado y creación de una red virtual para garantizar una ejecución eficiente y coherente del procedimiento de modificación:

1. Instalación de dependencias (las mismas que al crear la red virtual, Figura 4-11).
2. Obtención de una lista de las redes virtuales existentes, guardándola en una variable para su posterior uso (igual que en la creación de redes virtuales, Figura 4-12).

En este momento, creamos un bloque (*block,* Figura 4-24) denominado *Modificar Red Virtual*, el cual agrupará todas las tareas relativas al borrado y la creación. Este bloque de *modificación* se ejecutará cuando se den dos eventos a la vez:

- El nombre de la red a modificar (variable *net_to_modify*) debe estar en la lista de redes virtuales, es decir, la red debe existir.
- El nuevo nombre de la red (variable *vnet_name*) no debe existir para evitar así conflictos.

```
- name: Modificar red virtual
  block: …
  when: "net_to_modify in existing_nets.list_nets and vnet_name not in existing_nets.list_nets"
```

Figura 4-24. Bloque de las tareas de modificar red virtual

Si se cumplen las dos comprobaciones anteriores se ejecutarán las tareas (detalladas en los capítulos de creación y borrado) del bloque de modificación:

3. Borrado de la red virtual (Figura 4-20)
4. Definición de la red virtual con el nuevo nombre, proporcionado el archivo XML. (Figura 4-13)
5. Activación de la nueva red.(Figura 4-14)
6. Borrado de la red definida en caso de fallo por prefijo en uso.(Figura 4-15)
7. Configuración del arranque automático de la red. (Figura 4-16)

5 DEFINICIÓN DE MÁQUINAS VIRTUALES

Cuanto más grande es la dificultad, más gloria hay en superarla.

- Epicuro -

En este capítulo se abordarán las configuraciones y tareas necesarias para la gestión de máquinas virtuales en servidores de virtualización. Específicamente, nos centraremos en la creación de máquinas virtuales basadas en kernel (KVM).

KVM es una tecnología de virtualización *open source* integrada en Linux. Con ella, podemos transformar Linux en un hipervisor que permite que una máquina host ejecute varios entornos virtuales aislados llamados máquinas virtuales o *guests*. La arquitectura de esta tecnología es la mostrada en la Figura 5-1.

KVM forma parte de Linux. Por eso, si usamos como servidores de virtualización Linux con versiones iguales o posteriores a 2.6.20, ya estarán disponibles por defecto.

KVM convierte Linux en un hipervisor de tipo 1 (servidor dedicado). Todos los hipervisores necesitan algunos elementos del sistema operativo para ejecutar las máquinas virtuales. KVM incorpora todos estos elementos necesarios al formar parte del núcleo de Linux. Cada máquina virtual se implementa como un proceso habitual de Linux e incluye sistemas virtuales hardware exclusivos, como la tarjeta de red, el adaptador gráfico, las CPU, la memoria y los discos. [32]

Las ventajas asociadas al uso de KVM radican en sus características específicas, lo que la convierte en la opción preferida por parte de las empresas: [33]

- **Alto rendimiento**: KVM está diseñada para administrar aplicaciones de alta exigencia sin problemas. Todos los sistemas operativos invitados heredan el alto rendimiento del sistema operativo del host, es decir, Linux.
- **Seguridad:** Utilizan una combinación de SELinux y *Secure Virtualization* para mejorar la seguridad y el aislamiento de las máquinas virtuales, aplicando controles de acceso obligatorios (MAC) a las máquinas virtuales *guests* y previniendo los errores relacionados con el etiquetado manual.
- **Almacenamiento:** Pueden usar todos los tipos de almacenamiento compatibles con Linux, lo cual incluye algunos discos locales y el almacenamiento conectado en red.
- **Estabilidad**: Se utiliza de forma generalizada en aplicaciones empresariales desde hace más de una década. Goza del respaldo de una comunidad de código abierto muy activa. Además, el código fuente ha alcanzado la madurez necesaria y constituye una base estable para las aplicaciones empresariales.
- **Flexibilidad**: Pueden asignarse de forma eficiente CPU, almacenamiento o memoria adicionales a una máquina virtual.

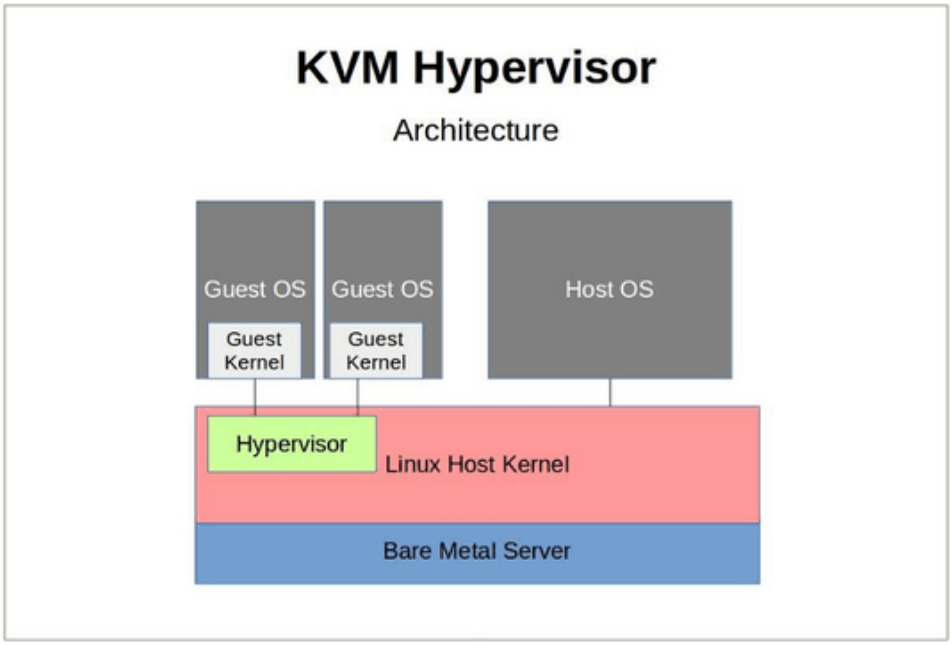


Figura 5-1. Arquitectura hipervisor KVM

5.1 Configuración del paybook

Siguiendo el mismo principio de reutilización de roles de Ansible utilizado en el playbook de las redes virtuales, procederemos de manera análoga en la creación del playbook (Figura 5-2) correspondiente a las máquinas virtuales. La única diferencia radicará en el rol invocado. En consecuencia, los demás parámetros se encuentran especificados en el capítulo 4.1 Configuración del playbook, por lo que no se detallarán en esta sección.

```
- name: Definición de máquinas virtuales
  hosts: cloud
  #become: false
  roles:
    - virt_machines
```

Figura 5-2. Playbook máquinas virtuales

5.2 Configuración del archivo *main.yml* de las tareas

De manera similar a lo expuesto en el capítulo relativo a las redes, este archivo (Figura 5-4) se encargará de incorporar las tareas relacionadas con las funciones principales que se pueden ejecutar en torno a las máquinas virtuales. En este contexto, es importante destacar la inclusión de dos archivos de tareas orientados a la creación de máquinas virtuales, es decir, tendremos un archivo para crear nuevas máquinas virtuales mediante la generación de discos virtuales específicos para cada máquina, así como otro archivo que incluirá las tareas para crear una máquina virtual a partir de discos virtuales preexistentes y descargados de los repositorios oficiales de las principales distribuciones de Linux. Estos discos contendrán el sistema operativo ya instalado.

Por lo tanto, esto introducirá una nueva variable (*createPreVM)* que, según su valor, incluirá las tareas relacionadas con las máquinas virtuales predefinidas.

Así, la estructura de los archivos de tareas de las máquinas virtuales queda organizada de la siguiente manera:

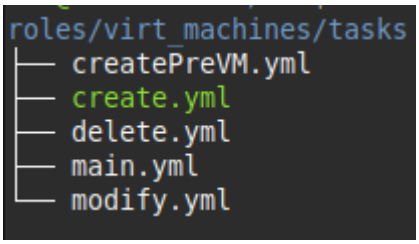


Figura 5-3. Archivos de las tareas de máquinas virtuales

El archivo *main.yml* queda definido de la siguiente manera:

```
---
# Definimos las tareas relacionadas con las máquinas virtuales virtuales y las condiciones de su ejecución
- name: Crear máquina virtual predefinida
  include_tasks: createPreVM.yml
  when: createPreVM | bool and not create and not modify and not delete

- name: Crear máquina virtual
  include_tasks: create.yml
  when: create | bool and not createPreVM and not modify and not delete

- name: Modificar máquina virtual
  include_tasks: modify.yml
  when: modify | bool and not createPreVM and not create and not delete

- name: Borrar máquina virtual
  include_tasks: delete.yml
  when: delete | bool and not createPreVM and not create and not modify
```

Figura 5-4. Contenido del archivo main.yml del rol de las máquinas virtuales

Las tareas se dividirán en cuatro archivos distintos, al igual que con las redes virtuales, incluyéndose cada archivo en función del valor de las variables booleanas. De esta manera conseguimos una gestión eficiente y estructurada de las máquinas virtuales.

5.3 Configuración de la plantilla XML de los Pools

Una máquina virtual, al igual que una máquina física, requiere almacenamiento de datos, programas y archivos del sistema.

Un pool o grupo de almacenamiento es un archivo, directorio o dispositivo de almacenamiento administrado por *libvirt* para proporcionar almacenamiento a máquinas virtuales. Los pools se dividen en volúmenes de almacenamiento, lo cuales contienen imágenes de disco de máquinas virtuales o se adjuntan a las máquinas como almacenamiento adicional.

Además, varias máquinas virtuales pueden compartir el mismo grupo de almacenamiento, lo que permite una mejor asignación de los recursos disponibles. [34]

Para crear un pool, es necesario proporcionar una plantilla (Figura 5-5) en XML que lo defina, la cual situamos dentro del directorio *templates* del rol de *virt_machines.* En esta plantilla, se vuelve a utilizar la sintaxis Jinja2 para permitir su completado de manera dinámica, similar a la plantilla utilizada para la creación de una red virtual.

```
<pool type='dir'>
  <name>{{ vm_disk_OS }}_Guests</name>
  <target>
    <path>/{{ vm_disk_OS }}_VMs</path>
  </target>
</pool>
```

Figura 5-5. Plantilla XML de un pool

Donde se detalla:

- **pool type='dir':** Especifica que el pool estará basado en directorio.
- **name:** Indica el nombre del pool. Todos los nombres finalizarán con *_Guests*, precedidos del nombre de la distribución de Linux de la máquina a crear. Esta distribución será proporcionada por el administrador en la solicitud al servidor REST. Por lo tanto, los 4 pools que se crearán son: *Ubuntu_Guests, Debian_Guests, Fedora_Guests y CentOS_Guests.*
- **path:** Especifica la ruta al directorio donde se ubicará el pool, el cual almacenará los volúmenes o discos virtuales de las máquinas que se creen. Se crearán, por tanto, cuatro directorios en la raíz del sistema: */Ubuntu_VMs, /Debian_VMs, /Fedora_VMs y /CentOS_VMs.*

En resumen, los pools proporcionan una manera estructurada y organizada de gestionar el almacenamiento de las máquinas que se vayan a crear.

5.4 Creación de máquinas virtuales

Para la creación de las máquinas virtuales, hemos definido dos posibilidades:

- **Crear una máquina virtual a partir de un disco *cloud* preexistente**, con el sistema operativo ya instalado.
- **Crear una máquina virtual generando un disco virtual nuevo** y asignándole la imagen ISO del sistema operativo que se instalará una vez sea iniciada.

5.4.1 Creación de máquinas virtuales a partir de discos preexistentes

5.4.1.1 Desarrollo Python

Tabla 5-1. Características de la petición de crear una máquina virtual a partir de un disco preexistente

Método HTTP	POST
EndPoint	/createPreVM
Parámetros de la petición	**Nombre** de la máquina virtual a crear
	Memoria **RAM** en MB
	Número de **vCPUs**
	Nombre de la **red** a la que se conectará la máquina
	Nombre de la **distribución de Linux** a instalar

En primer lugar, se verifica que la solicitud (Tabla 5-1) contiene cuatro de los cinco parámetros requeridos. Esto se debe a que, si el administrador no especifica el nombre de la red, la máquina se conectará por defecto a

la red *default*. Por tanto, el parámetro de la red no es obligatorio que viaje en una solicitud.

```
# Petición POST para la creación de una máquina virtual predefinida mediante un disc
@app.post("/createPreVM")
def createPreVM():

    # Parámetros: Cantidad de memoria RAM (MB), el número de virtual CPUs, la red vi
    RAM = request.form.get("ram")
    CPU = request.form.get("cpu")
    #tamaño = request.form.get("tam")
    virtual_net = request.form.get("virtual_net")
    vm_name = request.form.get("name")
    virtual_disk_OS = request.form.get("VdiskOS")

    # Variable usada en la ejecuciónde tareas
    #pool_name = virtual_disk_OS + "_Guests"

    # Comprobamos que se están recibiendo los tres parámetros, si no se especfica re
    if RAM is None or virtual_disk_OS is None or CPU is None or vm_name is None:
        abort(400,"Faltan campos en la solicitud POST")
    elif virtual_net is None:
        virtual_net = "default"
```

Figura 5-6. Código Python para crear máquina virtual a partir de disco preexistente.

Para poder ejecutar el comando de creación de una máquina virtual, es recomendable especificar el tipo de sistema operativo que se instalará. Por ello, se define una variable que se pasa a Ansible, la cual contiene el sistema operativo de la distribución que se desea instalar, según el nombre de la distribución proporcionado por el administrador. Si se proporciona una distribución distinta a las establecidas, se producirá un error y no se ejecutará ninguna tarea.

```
# Establecemos el tipo de Linux
if virtual_disk_OS == "Debian":
    os_type = "debian9"
elif virtual_disk_OS == "Ubuntu":
    os_type = "ubuntu20.04"
elif virtual_disk_OS == "CentOS":
    os_type = "centos7.0"
elif virtual_disk_OS == "Fedora":
    os_type = "fedora31"
else:
    abort(400,"Nombre de sistema operativo no válido")
```

Figura 5-7. Código Python definición de variable de sistema operativo

A continuación, se crea el diccionario de variables (Figura 5-8) que se pasará a Ansible y se ejecuta la función *run* de Ansible Runner:

```
# Establecemos la variable que define la acción a realizar, en este caso la creación
variables = {"createPreVM": True, "create": False, "delete": False,
             "modify": False, "vm_ram": RAM, "vm_disk_OS": virtual_disk_OS,
             "vm_cpu":CPU, "virtual_net": virtual_net, "vm_name":vm_name,
             "vm_os_type": os_type}
```

Figura 5-8. Código Python creación de diccionario de variables

Finalmente, se evalúa el objeto *runner* devuelto por la ejecución y se gestionan los posibles errores (Figura 5-9) que puedan haber ocurrido, tales como que el nombre de la máquina ya exista, que la red no exista o

cualquier otro error inesperado.

```
for events in runner.events:
    if(events["event"] == "runner_on_skipped" and events["event_data"]["task"] == "Crear máquina virtual predefinida"):
        abort(500,"Error: Nombre de la máquina virtual ya existente")
    elif(events["event"] == "runner_on_failed" and events["event_data"]["task"] == "Crear máquina virtual predefinida"):
        abort(500,"Error al instalar la máquina virtual " + str(events["event_data"]["res"]["stderr"]))
    elif(events["event"] == "runner_on_failed"):
        abort(500,"Error en la tarea " + str(events["event_data"]["task"]) + ": " + str(events["event_data"]["res"]["stderr"]))
    else:
        response = "Máquina predefinida " + str(vm_name) + " creada correctamente con " + str(RAM) + " de RAM, " + str(CPU)
        + " núcleos, sistema operativo " + str(virtual_disk_OS) + " y ligada a la red virtual " + str(virtual_net)

return response
```

Figura 5-9. Código Python gestión de errores al crear máquina virtual con disco preexistente

5.4.1.2 Configuración de las tareas de Ansible

Usando como guía los procedimientos descritos en [35] y [36], se definirán las tareas para la correcta creación de máquinas virtuales basadas en discos virtuales predefinidos.

1. Instalación de los paquetes necesarios para la correcta creación de máquinas virtuales en el servidor de virtualización (Figura 5-10):
    a. Instalación del demonio de *libvirt*.
    b. Instalación del paquete *python3-libvirt*.
    c. Instalación del paquete *python3-lxml*.
    d. Instalación del paquete *libguestfs-tools:* Un conjunto de herramientas para acceder y modificar imágenes de disco de máquinas virtuales.
    e. Instalación del paquete *virtinst*: Contiene una serie de herramientas auxiliares de más alto nivel que virsh y que facilitan la creación de máquinas virtuales, entre otras tareas.
    f. Módulo de Ansible utilizado para esta tarea: ***Package***

```
---
- name: Instalar dependencias
  become: true
  package:
    name: "{{ item }}"
    state: latest
  with_items:
    - python3-libvirt
    - libguestfs-tools # En Fedora es
    - virtinst
    - python3-lxml
    - libvirt-daemon-system
```

Figura 5-10. Tarea instalar dependencias al crear una máquina virtual

2. Asegurar que el demonio de *libvirt* está arrancado, ya que a veces no se inicia automáticamente (Figura 5-11):
    a. Solo el usuario *root* puede ejecutar esta tarea, por lo tanto, es importante indicarlo con ***become: true***.
    b. Módulo de Ansible empleado para esta tarea:
        i. ***Service:*** Este módulo controla los servicios en hosts remotos. [37]

```yaml
- name: Asegurar que el demonio libvirt está arrancado
  become: true
  service:
    name: libvirtd
    state: started
    enabled: true
```

Figura 5-11. Tarea arrancar el módulo de libvirt al crear máquinas virtuales con discos preexistentes

3. Obtención de una lista de las máquinas virtuales existentes (Figura 5-12):
    a. La lista se guardará en una variable que será utilizada posteriormente.
    b. Módulo de Ansible usado:
        i. ***Virt:*** Gestiona máquinas virtuales soportadas por *libvirt*. Parámetro ***command*** con opción ***list_vms***. [38]

```yaml
- name: Obtener lista de máquinas virtuales existentes
  become: true
  virt:
    command: list_vms
  register: existing_vms
```

Figura 5-12. Tarea obtener lista de máquinas virtuales existentes

A partir de aquí se crea el bloque que contiene las tareas para crear la máquina virtual. Este bloque incluye las siguientes tareas:

4. Descarga de los discos virtuales *cloud* de los repositorios de las distribuciones (Figura 5-13):
    a. Se descargará el disco virtual correspondiente a la distribución especificada en la solicitud del administrador.
    b. El disco descargado se guardará en la ruta ***/tmp*** **y** siempre tendrá el nombre de la máquina virtual seguido de la extensión **qcow2**[13]**.**
    c. Módulo de Ansible:
        i. ***Get_url:*** Descarga archivos desde HTTP, HTTPS o FTP al servidor remoto. Parámetro ***url*** desde donde descargará el disco virtual y ***dest*** donde se ubicará el archivo descargado. [39]

[13] Es el formato nativo de KVM. A diferencia de los discos *raw*, el espacio de los discos *qcow2* crece a medida que se necesita, mientras que en los discos *raw* todo el almacenamiento debe provisionarse de entrada.

```
- name: Crear máquina virtual si no existe
  block:
    - name: Descargar imagen base Fedora
      get_url:
        url: https://archives.fedoraproject.org/pub/archive/fedora/linux/releases/31/Cloud/x86_64/images/Fedora-Cloud-Base-31-1.9.x86_64.qcow2
        dest: "/tmp/{{ vm_name }}.qcow2"
      when: vm_disk_OS == "Fedora"

    - name: Descargar imagen base Ubuntu
      get_url:
        url: https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img
        dest: "/tmp/{{ vm_name }}.qcow2"
      when: vm_disk_OS == "Ubuntu"

    - name: Descargar imagen base Debian
      get_url:
        url: https://cloud.debian.org/images/cloud/OpenStack/current-9/debian-9-openstack-amd64.qcow2
        dest: "/tmp/{{ vm_name }}.qcow2"
      when: vm_disk_OS == "Debian"

    - name: Descargar imagen base CentOS
      get_url:
        url: https://cloud.centos.org/centos/7/images/CentOS-7-x86_64-GenericCloud.qcow2
        dest: "/tmp/{{ vm_name }}.qcow2"
      when: vm_disk_OS == "CentOS"
```

Figura 5-13. Tareas de descargar discos virtuales cloud de las distribuciones

5. Copiar el disco virtual al directorio por defecto de libvirt (***images***) (Figura 5-14):
    a. Esto convierte dicho directorio en un pool donde se almacenan los discos virtuales de las máquinas ya configurados con el sistema operativo instalado.
    b. Módulo de Ansible utilizado:
    c. ***Copy:*** Este módulo copia un archivo o una estructura de directorios desde la máquina local o remota a una ubicación en la máquina remota. Parámetro ***src*** con la ruta al archivo que se desea copiar y ***dest*** con la ruta de destino del archivo. En este caso el origen será */tmp* donde se descargó el disco virtual y el destino será el directorio ***images*** de libvirt. También se especifican los permisos del archivo y se almacena el resultado de esta tarea en una variable para su uso posterior. [40]

```
- name: Copiar imagen al directorio libvirt
  become: true
  copy:
    dest: "/var/lib/libvirt/images/{{ vm_name }}.qcow2"
    src: "/tmp/{{ vm_name }}.qcow2"
    force: false
    remote_src: true
    mode: 0660
  register: copy_results
```

Figura 5-14. Tarea copiar disco virtual descargado al directorio por defecto de libvirt

La siguiente tarea no es necesaria, es opcional y se realiza para demostrar las posibilidades que habría en caso de crear una máquina virtual de esta manera.

6. Configurar el disco de la máquina virtual a crear (Figura 5-15):
    a. Se ejecutará cuando el resultado de la tarea anterior haya cambiado, es decir, cuando se haya ejecutado la tarea anterior, ya que si el disco no se ha copiado correctamente en la ruta de libvirt no se podrá llevar a cabo la personalización.
    b. Se usará el comando ***virt-customize,*** que permite modificar la imagen de disco de una máquina. Los parámetros personalizados son los siguientes:
        i. Ruta donde se ubica el disco a configurar. En este caso, el directorio ***images*** de libvirt.
        ii. Establecemos el nombre del host al nombre de la máquina virtual.

iii. Establecemos una contraseña para el usuario root.

iv. Inyectamos la clave SSH pública del usuario root del servidor de virtualización para que pueda acceder a la máquina creada mediante SSH y sin proporcionar contraseña.

v. Desinstalamos *cloud-init* de la máquina para evitar demoras en el inicio, ya que espera los parámetros para la inicialización con los scripts de *cloud-init.*

```
- name: Configurar la imagen
  become: true
  command:
    virt-customize -a /var/lib/libvirt/images/{{ vm_name }}.qcow2
    --hostname {{ vm_name }} --root-password password:{{ vms_default_root_pass }}
    --ssh-inject "root:file:/root/.ssh/id_rsa.pub"
    --uninstall cloud-init --selinux-relabel
  when: copy_results is changed and vm_disk_OS != "Debian"
```

Figura 5-15. Tarea configurar disco de la máquina virtual

7. Crear máquina virtual (Figura 5-16):

   a. Se utilizará la herramienta de línea de comandos ***virt-install*** para crear la máquina, pasándole todos los parámetros necesarios.

      i. Nombre de la nueva instancia de la máquina virtual a crear.

      ii. Memoria RAM a asignar.

      iii. Red a la que conectarse.

      iv. Número de CPUs virtuales a configurar.

      v. Especifica los medios a utilizar como almacenamiento, en este caso, la ruta al disco virtual descargado anteriormente.

      vi. Opción ***–import*** para indicar que se creará la máquina a partir de un disco existente, omitiendo el proceso de instalación del sistema operativo y creándola alrededor de esta imagen de disco.

      vii. Tipo de sistema operativo, en nuestro caso siempre será Linux

      viii. Variante del sistema operativo, variará en función de la distribución especificada en la solicitud del administrador. Tanto la opción anterior como esta, aunque no es obligatorio, si es recomendable indicarlas ya que puede aumentar significativamente el rendimiento de la máquina.

      ix. Indicamos que se inicie automáticamente y que no intente conectarse a la consola de la máquina de manera automática ya que si no la ejecución de las tareas de Ansible no avanzará y se quedará en este punto.

   b. Se registrará el resultado de esta tarea en una variable e ignorarán los errores en caso de fallo para posteriormente llevar a cabo una serie de tareas adicionales como consecuencia del fallo.

```
- name: Crear máquina virtual predefinida
  become: true
  command: >
   virt-install --name {{ vm_name }} --memory {{ vm_ram }}
   --network network={{ virtual_net }} --vcpus {{ vm_cpu }}
   --disk /var/lib/libvirt/images/{{ vm_name }}.qcow2
   --import --os-type linux --os-variant {{ vm_os_type }}
   --autostart --noautoconsole
  register: status_create
  ignore_errors: true
when: "vm_name not in existing_vms.list_vms"
```

Figura 5-16. Tarea crear máquina virtual a partir de disco virtual existente

8. Borrar el archivo temporal del disco descargado en el directorio ***/tmp*** ya que se copió al directorio de libvirt y, por lo tanto, no es necesario ya (Figura 5-17):
    a. Módulo de Ansible:
        i. ***File:*** Permite eliminar archivos, directorios o enlaces simbólicos en el servidor remoto. Parámetro ***path*** con la ruta del archivo a eliminar y parámetro ***state*** con valor ***absent***, indicando que no se desea que el archivo esté presente y, por tanto, lo elimina en caso de que exista. [41]

```
- name: Borrar archivo temporal de la imagen
  file:
    path: "/tmp/{{ vm_name }}.qcow2"
    state: absent
  when: borrar_tmp | bool
```

Figura 5-17. Tarea borrar el archivo temporal de la imagen de disco

9. Borrar el disco virtual en caso de error al crear la máquina (Figura 5-18):
    a. Este es el motivo por el que se ignoran los errores en la tarea de crear la máquina virtual ya que, si da error, el disco permanece en el directorio de libvirt, pero nunca estará en uso ya que la máquina no se creó. En ese caso, se eliminará dicho disco si el resultado de la tarea de creación de la máquina falló:
    b. Módulo de Ansible: ***file***.

```
- name: Borrar disco duro virtual creado en caso de error al crear la máquina
  become: true
  file:
    path: "/var/lib/libvirt/images/{{ vm_name }}.qcow2"
    state: absent
  when: "status_create is failed"
```

Figura 5-18. Tarea borrar disco virtual en caso de error al crear la máquina

5.4.2 Creación de máquinas virtuales generando un disco virtual nuevo

5.4.2.1 Desarrollo Python

Tabla 5-2. Características de la petición de crear una máquina virtual generando un disco virtual nuevo

Método HTTP	POST
Endpoint	/createVM
Parámetros de la petición	**Nombre** de la máquina virtual a crear
	Memoria **RAM** en MB
	Número de **vCPUs**
	Nombre de la **red** a la que se conectará la máquina
	Nombre de la **distribución de Linux** a instalar
	Tamaño del disco en GB

En este escenario, la diferencia respecto a la creación de una máquina virtual a partir de una imagen de disco preexistente radica en que las imágenes de disco predefinidas tienen un tamaño de disco establecido y definido por su creador. Al crear una máquina virtual generando un disco nuevo, es necesario especificar el tamaño del disco virtual que se creará.

El resto de los parámetros y comprobaciones son los mismos. Se incluye tanto la creación de la variable que define la variante de la distribución de Linux en función de la distribución proporcionada por el administrador, como la asignación de la red por defecto en caso de que no se proporcione ninguna. No obstante, se creará una nueva variable (Figura 5-19) que se pasará a Ansible, utilizando el nombre de la distribución de Linux proporcionado en la solicitud para formar el nombre del pool que se creará.

```
# Variable usada para formar el nombre del pool que se creará
pool_name = virtual_disk_OS + "_Guests"
```

Figura 5-19. Variable del nombre del pool que se creará

De manera similar a lo visto anteriormente, se crea el diccionario de variables, con el valor verdadero de la variable *create.* Luego, se ejecuta Ansible Runner y, usando el objeto proporcionado, se realizan distintas verificaciones para gestionar posibles errores que puedan haber ocurrido, tales que el nombre de la máquina a crear ya exista, que el nombre de la red a asignar no exista o cualquier error no esperado.

```
# Establecemos la variable que define la acción a realizar, en este caso la creación, además de las caracteristicas de la máquina a crear
variables = {"createPreVM": False, "create": True, "delete": False, "modify": False, "vm_ram": RAM,
             "vm_disk_OS": virtual_disk_OS, "vm_cpu":CPU, "virtual_net": virtual_net,
             "vm_name":vm_name, "vm_os_type": os_type, "vm_tam": tamaño_disco, "pool_name": pool_name}

runner = ansible_runner.run(private_data_dir=".", playbook="virtual_machines_playbook.yml",
                            extravars=variables, inventory="inventory/hosts")

for events in runner.events:
    if(events["event"] == "runner_on_skipped" and events["event_data"]["task"] == "Obtener XML de la máquina a crear"):
        abort(500,"Error: Nombre de la máquina virtual ya existente o nombre de red no existe")
    elif(events["event"] == "runner_on_failed"):
        abort(500,"Error en la tarea " + str(events["event_data"]["task"]) + ": " + str(events["event_data"]["res"]["stderr"]))
    else:
        response = "Máquina " + str(vm_name) + " creada correctamente con " + str(RAM) + " de RAM, "
        + str(tamaño_disco) +  "GB de tamaño de disco, " + str(CPU) + " núcleos, sistema operativo "
        + str(virtual_disk_OS) + " y ligada a la red virtual " + str(virtual_net)

return response
```

Figura 5-20. Código Python creación de máquinas virtuales

5.4.2.2 Configuración de las tareas de Ansible

1. Instalación de los paquetes requeridos para la correcta ejecución de las tareas, de manera similar a los descrito en el capítulo 5.4.1 Creación de máquinas virtuales a partir de discos preexistentes, en la

Figura 5-10.

2. Asegurar que el demonio *libvirt* está en funcionamiento (Figura 5-11).
3. Obtener lista de máquinas, redes y pools existentes (Figura 5-21):
    a. Cada lista se almacenará en una variable para su uso posterior.
    b. Módulos de Ansible utilizados:
        i. ***Virt***
        ii. ***Virt_net***
        iii. ***Virt_pool:*** Gestiona grupos de almacenamiento/pools *libvirt*. Parámetro ***command*** con opción ***list_pools***. [42]

```
- name: Obtener lista de máquinas virtuales existentes
  become: true
  virt:
    command: list_vms
  register: existing_vms

- name: Obtener lista de Pools existentes
  become: true
  virt_pool:
    command: list_pools
  register: existing_pools

- name: Obtener lista de redes existentes
  become: true
  virt_net:
    command: list_nets
  register: existing_nets
```

Figura 5-21. Tareas obtener listas de máquinas, redes y pools existentes al crear una máquina virtual

A continuación, se crea un bloque que contiene las tareas de descarga de las imágenes ISO del sistema operativo que se instalará, en función de la distribución indicada por el administrador en la solicitud.

4. Crear el directorio ***/ISOS*** en la raíz del sistema, donde se almacenarán las imágenes de los sistemas operativos descargados (Figura 5-22):
    a. Esto convertirá este directorio en un pool al usar la máquina virtual una de las imágenes ISO descargadas.
    b. Módulo de Ansible:
        i. ***File:*** Parámetro ***state*** con la opción ***directory*** para indicar la creación del directorio.

```
- name: Crear el directorio donde se guardaran las imágenes de los sistemas operativos
  file:
    path: /ISOS
    state: directory
    mode: '0755'
```

Figura 5-22. Tarea de crear el directorio /ISOS para guardar las imágenes descargadas

5. Descargar las imágenes ISO y guardarlas en el directorio ***/ISOS*** (Figura 5-23)***:***
    a. La imagen adecuada se descargará según el sistema operativo indicado en la solicitud al

servidor REST.

b. Módulo de Ansible:

i. ***Get_url***

```
- name: Descargar imagen Ubuntu ISO
  get_url:
    url: https://releases.ubuntu.com/20.04.6/ubuntu-20.04.6-live-server-amd64.iso?_gl=1*e3o3ba*_gcl_au*Njg3NDg0NDUyLjE3MTU2NzcxNjI.&_ga
    dest: "/ISOS/Ubuntu.iso"
  when: "vm_disk_OS == 'Ubuntu'"

- name: Descargar imagen Debian ISO
  get_url:
    url: https://cdimage.debian.org/debian-cd/current/amd64/iso-cd/debian-12.5.0-amd64-netinst.iso
    dest: "/ISOS/Debian.iso"
  when: "vm_disk_OS == 'Debian'"

- name: Descargar imagen CentOS ISO
  get_url:
    url: http://ftp.rediris.es/mirror/CentOS/7.9.2009/isos/x86_64/CentOS-7-x86_64-DVD-2009.iso
    dest: "/ISOS/CentOS.iso"
  when: "vm_disk_OS == 'CentOS'"

- name: Descargar imagen Fedora ISO
  get_url:
    url: https://download.fedoraproject.org/pub/fedora/linux/releases/40/Server/x86_64/iso/Fedora-Server-dvd-x86_64-40-1.14.iso
    dest: "/ISOS/Fedora.iso"
  when: "vm_disk_OS == 'Fedora'"
```

Figura 5-23. Tarea de descargar las imágenes ISO

En este punto, se crea un bloque encargado de crear un pool para cada distribución que se podrá instalar en la máquina virtual, es decir, Ubuntu, Debian, Fedora y CentOS. Solo se crearán la primera vez que se vaya a crear una máquina para la distribución indicada. A partir de ese momento, siempre estará presente en el servidor de virtualización. Los discos de las máquinas virtuales se organizarán, por tanto, en 4 pools:

- Ubuntu_Guests: Pool para discos de máquinas Ubuntu.
- Debian_Guests: Pool para discos de máquinas Debian.
- Fedora_ Guests: Pool para discos de máquinas Fedora.
- CentOS_Guests: Pool para discos de máquinas CentOS.

6. Crear el directorio del pool donde se almacenarán los discos de cada máquina virtual (Figura 5-24):
   a. Cada uno de los 4 directorios que se crearán comenzará por el nombre de la distribución seguido de ***_VMs.***
   b. Módulo de Ansible:
      i. ***File***

```
- name: Crear directorio del pool
  file:
    path: /{{ vm_disk_OS }}_VMs
    state: directory
    mode: '0755'
```

Figura 5-24. Tarea de crear directorio del pool al crear una máquina virtual

7. Crear pool usando la plantilla XML (Figura 5-25):
   a. Módulo de Ansible:
      i. ***Virt_pool:*** Parámetros ***command*** con el valor ***define,*** indicando la creación del pool sin iniciarlo. Se especifica el XML de la plantilla a usar para definirlo.

```
- name: Definir Pool
  virt_pool:
    command: define
    name: "{{ vm_disk_OS }}_Guests"
    xml: '{{ lookup("template", "pool.xml.j2") }}'
```

Figura 5-25. Tarea de definir pool al crear la máquina virtual

8. Iniciar el pool y configurar el autoarranque al iniciarse el servidor de virtualización:
    a. Módulo de Ansible:
        i. ***Virt_pool:*** Parámetro ***command*** con las opciones ***start*** y ***autostart*** con el valor *yes.*

Todas estas tareas conforman la creación del pool, realizándose la primera vez que se vaya a crear una máquina de una distribución Linux determinada en el servidor de virtualización. Si ya está creado, estas tareas se omitirán y se procederá directamente a la creación de la máquina virtual dentro del pool adecuado.

Este nuevo bloque de tareas se ejecutará cuando la máquina a crear no exista.

9. Generar el XML de la máquina virtual a crear (Figura 5-26):
    a. Usamos el comando ***virt-install*** indicando las siguientes opciones:
        i. Nombre.
        ii. Memoria RAM.
        iii. Red a la que se conectará.
        iv. Número de CPUs virtuales.
        v. Nombre del Pool donde se creará el disco de la máquina, usando el nombre de la distribución indicada en la solicitud.
        vi. Tamaño del disco.
        vii. Imagen ISO desde la que se instalará el sistema operativo en la máquina.
        viii. Tipo de sistema operativo, en nuestro caso siempre Linux.
        ix. Variante del sistema operativo.
        x. Autoarranque al iniciarse el servidor de virtualización.
        xi. Opción de imprimir el XML que define la máquina con las características especificadas. Este XML lo registramos en una variable llamada *vm_xml.*

```
- name: Crear máquina virtual
  become: true
  block:
    - name: Obtener XML de la máquina a crear
      command: >
        virt-install --name {{ vm_name }} --memory {{ vm_ram }}
        --network network={{ virtual_net }} --vcpus {{ vm_cpu }}
        --disk pool={{ vm_disk_OS }}_Guests,size={{ vm_tam }}
        --cdrom /ISOS/{{ vm_disk_OS }}.iso --os-type linux
        --os-variant {{ vm_os_type }} --autostart --noautoconsole
        --print-xml 1 #--autostart --noautoconsole
      register: vm_xml
```

Figura 5-26. Tarea de generar XML de la máquina virtual a crear

10. Generamos un archivo XML cuyo contenido sea el XML guardado en la variable anterior (Figura 5-27):
    a. Módulo de Ansible:
        i. ***Copy:*** Como contenido del archivo indicamos el nombre de la variable.

```
- name: Copiar el xml a una ruta temporal
  copy:
    content: "{{ vm_xml.stdout }}"
    dest: "/tmp/{{ vm_name }}.xml"
```

Figura 5-27. Tarea de generar archivo XML a partir de la variable que lo contiene

11. Traer el archivo XML al nodo administrador, ya que será necesario que esté en el directorio *templates* del rol para que Ansible pueda localizar la plantilla al momento de crear la máquina (Figura 5-28):
    a. Módulo de Ansible:
        i. ***Fetch:*** Este módulo funciona como el módulo *copy,* pero a la inversa, se utiliza para recuperar archivos de máquinas remotas. Se indican parámetros origen y destino, siendo origen, la ruta del archivo XML del servidor de virtualización y destino la ruta local donde se almacenará el archivo. [43]

```
- name: Traer XML a las plantillas de local
  fetch:
    src: /tmp/{{ vm_name }}.xml
    dest: /home/dit/TFG/roles/virt_machines/templates/
    flat: yes
```

Figura 5-28. Tarea de traer archivo XML al nodo administrador

12. Crear máquina virtual a partir del archivo XML (Figura 5-29):
    a. Módulo de Ansible:
        i. ***Virt:*** Comando ***define*** y parámetro ***xml*** con el nombre del archivo XML.
        ii. Ignoramos los errores de esta tarea para garantizar que, en caso de fallo al crear la máquina, se ejecuten siempre las dos tareas posteriores que incluyen la eliminación de los archivos XML, evitando así la acumulación de archivos innecesarios.

```
- name: Definir máquina virtual a partir del XML usando el módulo de Ansible
  become: true
  virt:
    command: define
    xml: "{{ lookup('template', '{{ vm_name }}.xml') }}"
  ignore_errors: true
```

Figura 5-29. Tarea de crear máquina virtual a partir de archivo XML

Las dos últimas tareas se ejecutarán siempre, independientemente de que haya algún error al crear la máquina virtual. La intención es asegurar que siempre se eliminen los archivos XML utilizados para definir la máquina, tanto a nivel local como en el servidor remoto.

13. Borrar archivos XML del servidor remoto y la plantilla XML del nodo local (Figura 5-30):
    a. Módulo de Ansible:
        i. ***File:*** Importante especificar en el borrado del archivo local el parámetro ***delegate_to: localhost***, ya que esa tarea debe ejecutarse en el nodo que lanza Ansible, no en el servidor remoto.

```
  - name: Borrar archivo temporal del XML del servidor
    file:
      state: absent
      path: /tmp/{{ vm_name }}.xml

  - name: Borrar archivo XML de la plantilla local
    become: false
    file:
      path: /home/dit/TFG/roles/virt_machines/templates/{{ vm_name }}.xml
      state: absent
    delegate_to: localhost
when: "vm_name not in existing_vms.list_vms and virtual_net in existing_nets.list_nets"
```

Figura 5-30. Tareas borrar archivos XML de los servidores local y remoto

5.5 Borrado de máquinas virtuales

5.5.1 Desarrollo Python

Tabla 5-3. Características de la solicitud para eliminar una máquina virtual

Método HTTP	DELETE
Endpoint	/deleteVM
Parámetros de la petición	**Nombre** de la máquina virtual a borrar

Para borrar una máquina virtual, el administrador solo necesita proporcionar el nombre de la máquina que desea borrar.

```
# Parámetros que recibimos de la petición
maquina_a_borrar = request.form.get("name")

# Comprobamos que no falta el parámetro
if maquina_a_borrar is None:
    abort(400,"Faltan parámetros en la petición POST")
```

Figura 5-31. Código Python para eliminar una máquina virtual

Una vez obtenido el nombre de la máquina de la solicitud, se procede a ejecutar Ansible, incluyendo el nombre en el diccionario de variables junto con otras variables, siendo la variable *delete* la única con el valor *true*.

Después de la ejecución de las tareas, se verifica si la tarea de eliminación de la máquina fue omitida por Ansible, lo que indicaría que la máquina virtual a borrar no existe. Además, se comprueba si alguna tarea falló, registrando el error en la respuesta a la solicitud.

```python
# Creamos el diccionario de variables que le pasaremos a Ansible
variables = {"create": False, "createPreVM": False, "delete": True, "modify": False, "vm_to_delete":maquina_a_borrar}

runner = ansible_runner.run(private_data_dir=".", playbook="virtual_machines_playbook.yml",
                            extravars=variables, inventory="inventory/hosts")

for events in runner.events:
    if(events["event"] == "runner_on_skipped" and events["event_data"]["task"] == "Borrar máquina virtual"):
        abort(500,"Error: Máquina virtual no existe")
    elif(events["event"] == "runner_on_failed"):
        abort(500,"Error en la tarea " + str(events["event_data"]["task"]) + ": " + str(events["event_data"]["res"]["stderr"]))
    else:
        response = "Máquina " + str(maquina_a_borrar) + " borrada correctamente"

return response
```

Figura 5-32. Código Python Borrar Máquina Virtual (2)

5.5.2 Configuración de las tareas de Ansible

1. Instalar dependencias (Figura 5-10).
2. Obtener una lista de las máquinas virtuales existentes y registrarla en una variable (Figura 5-12).
3. Obtener una lista de pools existentes y guardarla en una variable.

Se crea un bloque de tareas que se ejecuta únicamente si el nombre de la máquina virtual a eliminar no está en la lista de máquinas virtuales existentes. Dentro de este bloque se realizan las siguientes acciones:

4. Apagar la máquina virtual (Figura 5-33):
    a. Para poder eliminarla, es necesario que esté apagada.
    b. Módulo de Ansible:
        i. ***Virt:*** Parámetro ***state*** con valor ***destroyed***.

```yaml
- name: Eliminar máquina virtual si existe
  become: true
  block:
    - name: Apagar la máquina virtual a borrar
      virt:
        state: destroyed
        name: "{{ vm_to_delete }}"
```

Figura 5-33. Tarea de apagar la máquina virtual antes de eliminarla

5. Eliminar la máquina virtual, incluidos todos los dispositivos de almacenamiento asociados:
    a. Comando ***virsh undefine --remove-all-storage.*** Este comando no solo elimina la máquina virtual, sino también el disco asociado a ella.
6. Refrescar los pools (Figura 5-34):
    a. Esta tarea se incluye para evitar errores y mantener actualizados los pools y los discos contenidos en cada uno.
    b. Módulo de Ansible:
        i. ***Virt_pool:*** Comando ***refresh.***

```
  - name: Refrescar todos los pools para que actualicen capacidades
    virt_pool:
      command: refresh
      name: "{{ item }}"
    with_items: "{{ existing_pools.list_pools }}"
when: "vm_to_delete in existing_vms.list_vms"
```

Figura 5-34. Tarea refrescar pools una vez borrada una máquina virtual

5.6 Modificación de máquinas virtuales

Para modificar una máquina virtual, se han identificado las siguientes características como modificables, ya sea mediante un comando directo o modificando el archivo XML que define a dicha máquina:

- Memoria RAM.
- Tamaño del disco.
- Número de CPUs virtuales.
- Red a la que está conectada la máquina.

5.6.1 Desarrollo Python

Tabla 5-4. Características de la solicitud de modificar una máquina virtual

Método HTTP	PUT
Endpoint	/modifyVM
Parámetros de la petición	**Nombre** de la máquina virtual a modificar
	Nueva memoria **RAM** a asignar
	Nuevo **tamaño de disco** a asignar
	Nuevo número de **vCPUs** a asignar
	Nueva **red** a la que conectar la máquina

Para programar este apartado, es necesario indicar que el único parámetro obligatorio en la solicitud (Tabla 5-4) es el nombre de la máquina que se desea modificar. Esto se debe a que el administrador no siempre querrá modificar todas las características de una máquina al mismo tiempo. Además, si no se especifica una red a la que conectar, por defecto se usará la red *default.*

```python
# Petición PUT para modificar una máquina virtual
@app.put("/modifyVM")
def modifyVM():

    # Parámetros recibidos en la petición
    nombre = request.form.get("name")
    RAM = request.form.get("ram")
    tamaño_disco = request.form.get("tam")
    CPU = request.form.get("cpu")
    red = request.form.get("virtual_net")

    if nombre is None:
        abort(400,"Faltan parámetros obligatorios en la petición POST")
    if red is None:
        red = "default"
```

Figura 5-35. Código Python de modificar una máquina virtual

Se crea un diccionario de variables, estableciendo el valor de la variable *modify* en *true.* Posteriormente, se lanza Ansible y, usando el objeto *runner* devuelto, se verifican los posibles errores que puedan haber ocurrido, como la inexistencia de la máquina a modificar de la red indicada, así como otros errores adicionales que puedan surgir.

```python
# Creamos el diccionario de variables que le pasamos a Ansible
variables = {"create": False, "createPreVM": False, "delete": False, "modify": True,
             "vm_to_modify": nombre, "vm_ram": RAM, "vm_tam": tamaño_disco, "vm_cpus": CPU, "vm_net": red}

runner = ansible_runner.run(private_data_dir=".", playbook="virtual_machines_playbook.yml",
                            extravars=variables, inventory="inventory/hosts")

for events in runner.events:
    if(events["event"] == "runner_on_skipped" and events["event_data"]["task"] == "Parar máquina virtual a modificar"):
        abort(500,"Error: Máquina virtual a modificar no existe")
    if(events["event"] == "runner_on_skipped" and events["event_data"]["task"] == "Copiar XML a otra ubicación"):
        abort(500,"Red a modificar no existe")
    elif(events["event"] == "runner_on_failed"):
        abort(500,"Error en la tarea " + str(events["event_data"]["task"]) + ": " + str(events["event_data"]["res"]["stderr"]))
    else:
        response = "Máquina " + str(nombre) + " modificada correctamente"

return response
```

Figura 5-36. Código Python de modificar una máquina virtual (2)

5.6.2 Configuración de las tareas de Ansible

La modificación de cada una de las características mencionadas anteriormente se reflejará en la estructura de las tareas. Se creará un bloque global que incluye sub-bloques. Cada sub-bloque corresponderá a la modificación de una característica específica, siendo ejecutados solo cuando las variables asociadas a cada característica no estén vacías, es decir, cuando el administrador haya proporcionado un valor en la solicitud.

1. Instalar dependencias (Figura 5-10).
2. Obtener lista de máquinas virtuales existentes y guardarla en una variable (Figura 5-12).
3. Obtener lista de redes virtuales existentes y registrarla en una variable (Figura 4-12).
4. Descargar todas las imágenes ISO que no se encuentren en el directorio */ISOS* (Figura 5-23)*:*
    a. Esta tarea se incluye ya que, si se ha eliminado previamente una máquina virtual, es posible que la imagen ISO del sistema operativo a instalar se haya borrado. Para asegurarnos de su disponibilidad, se ejecutan nuevamente las tareas de descarga de las cuatro imágenes. Las imágenes ya existentes y no borradas no se volverán a descargar.

5. Parar la máquina virtual ya que para ciertas modificaciones es necesario que la máquina esté apagada.
6. Modificar memoria RAM cuando exista un valor para la variable asociada a la memoria RAM (Figura 5-37):
    a. Modificar RAM máxima: Se modifica la RAM máxima para evitar errores, asegurando que la cantidad de RAM asignada no sea inferior a la máxima especificada en la máquina. Por tanto, modificamos como máxima el nuevo valor:
        i. Usamos el comando ***virsh setmaxmem.***
    b. Modificar RAM en uso
        i. Usamos el comando ***virsh setmem.***

```
- name: Modificar RAM
  block:
  - name: Modificar RAM máxima a la máquina virtual
    command: virsh setmaxmem --domain {{ vm_to_modify }} --size {{ vm_ram }}M --config

  - name: Modificar RAM en uso de la máquina virtual
    command: virsh setmem --domain {{ vm_to_modify }} --size {{ vm_ram }}M --config
  when: "vm_ram is not none"
```

Figura 5-37. Tarea de modificar memoria RAM de una máquina virtual

7. Modificar tamaño del disco cuando sea necesario (Figura 5-38):
    a. Arrancar la máquina virtual para poder modificarlo, ya que de otra forma no es posible.
    b. Uso del comando ***virsh blockresize*** para llevar a cabo esta acción.

```
- name: Modificar tamaño del disco
  block:
    - name: Activar máquina virtual para poder modificar el tamaño del disco
      virt:
        name: "{{ vm_to_modify }}"
        state: running

    - name: Modificar tamaño del disco
      command: virsh blockresize {{ vm_to_modify }} --size {{ vm_tam }}G vda
  when: "vm_tam is not none"
```

Figura 5-38. Tarea de modificar tamaño del disco de una máquina virtual

8. Modificar número de CPUs virtuales (Figura 5-39):
    a. Apagar la máquina nuevamente para modificar esta característica.
    b. Modificar número máximo de CPUs virtuales:
        i. Uso del comando ***virsh setvcpus*** con la opción ***--maximum***
    c. Modificar número de CPUs virtuales en uso:
        i. Uso del comando ***virsh stvcpus*** con la opción ***–current***

```
- name: Modificar número de cpus
  block:
    - name: Modificar número máximo de vcpus
      command: virsh setvcpus --count {{ vm_cpus }} --maximum --domain {{ vm_to_modify }} --config

    - name: Modificar numero de vcpus en uso
      command: virsh setvcpus --domain {{ vm_to_modify }} --count {{ vm_cpus }} --current
  when: "vm_cpus is not none"
```

Figura 5-39. Tarea de modificar el número de CPUs virtuales

9. Modificar red a la que está conectada la máquina:
    a. Obtener la estructura XML que define la máquina virtual a modificar y guardarla en una variable (Figura 5-40):
        i. Uso del comando ***get_xml*** del módulo ***virt*** de Ansible.

```
- name: Modificar red a la que se conecta
  block:
    - name: Obtener XML de la máquina virtual a modificar
      virt:
        command: get_xml
        name: "{{ vm_to_modify }}"
      register: vm_xml
```

Figura 5-40. Tarea de obtener XML de la máquina virtual al modificarla

    b. Crear un archivo XML cuyo contenido sea la estructura XML de la máquina a modificar:
        i. Uso del módulo ***copy*** de Ansible, indicando la ruta donde de guardará el archivo nuevo.
    c. Modificar el atributo *network* de este archivo XML, estableciendo su valor al nombre de la red nueva a la que queremos conectar la máquina virtual (Figura 5-41):
        i. Uso del módulo ***xml*** de Ansible, el cual permite manejar archivos XML. [44]

```
- name: Modificar XML cambiando la red
  xml:
    path: /tmp/{{ vm_to_modify }}.xml
    xpath: /domain/devices/interface/source
    attribute: network
    value: "{{ vm_net }}"
```

Figura 5-41. Tarea de modificar archivo XML al cambiar la red de la máquina

    d. Eliminar la primera línea del archivo XML ya que puede causar errores durante la ejecución (Figura 5-42):
        i. Uso del módulo ***lineinfile*** de Ansible, el cual permite reemplazar una línea particular de un archivo. [45]

```
- name: Eliminar la primera línea del XML que da errores
  lineinfile:
    path: /tmp/{{ vm_to_modify }}.xml
    line: "<?xml version='1.0' encoding='UTF-8'?>"
    state: absent
```

Figura 5-42. Tarea de eliminar línea que da error del archivo XML

A partir de este punto, las tareas son las mismas que al crear una máquina virtual, utilizando el nuevo archivo XML para definir la máquina virtual con la red modificada:

e. Transferir el archivo XML al directorio *templates* del rol, en el nodo que ejecuta Ansible (Figura 5-28).

f. Crear la máquina virtual usando el nuevo archivo XML (Figura 5-29).

g. Borrar el archivo XML creado en el servidor remoto (Figura 5-30).

h. Borrar el archivo XML del directorio *templates* local, para evitar acumular archivos de máquinas modificadas (Figura 5-30).

6 DEFINICIÓN DE ROLES DE USUARIOS

Cada día sabemos más y entendemos menos.

- Albert Einstein -

En este capítulo, abordaremos la fase final del proyecto, que consiste en la creación de roles para los usuarios que se generarán con el fin de acceder a las máquinas virtuales. Estos roles se definirán en un servidor LDAP remoto, alojado en la nube, al cual se accederá mediante Ansible para configurar los roles o *entidades organizativas* necesarias. Para completar el escenario mostrado al principio de esta memoria, se incluye la siguiente imagen (Figura 6-1) que representa el escenario completo del proyecto realizado.

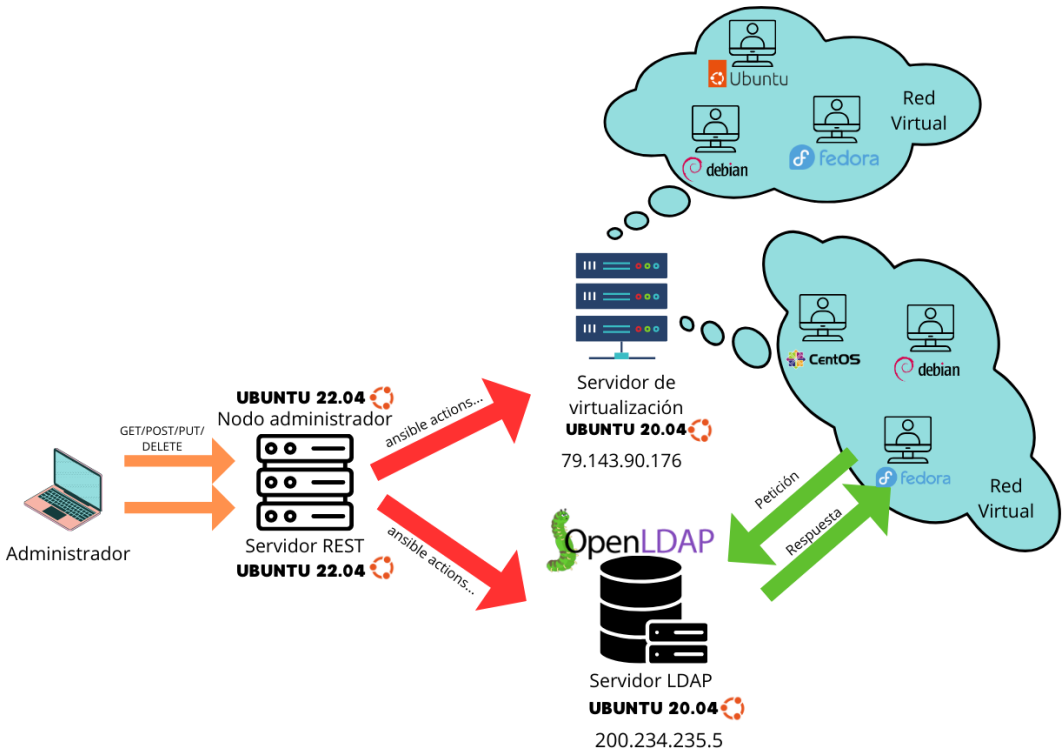


Figura 6-1. Escenario general completo del proyecto

Siguiendo las directrices proporcionadas en los vídeos del curso sobre OpenLDAP de [46] se procede a configurar Ansible para la instalación y configuración inicial del servicio OpenLDAP, en el caso de que sea el primer acceso al mismo.

El objetivo principal de este capítulo es establecer los roles que se asignarán a los usuarios para permitir su acceso a las máquinas virtuales creadas. Esto implica que la autenticación y el acceso a las máquinas virtuales se gestionarán a través del protocolo LDAP, por lo que dichas máquinas enviarán las solicitudes pertinentes a este servidor al iniciarse.

Adicionalmente, en caso de que sea la primera ocasión en la que se crea un rol en el servidor LDAP, será necesario instalar y configurar el servicio, dando lugar a la construcción de la siguiente estructura (Figura 6-2) en el directorio LDAP.

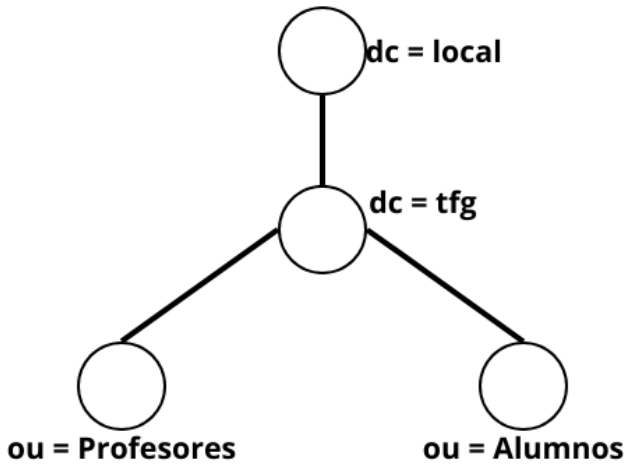


Figura 6-2. Estructura del directorio LDAP

En esta estructura, *Profesores* y *Alumnos* representarán los roles, y se añadirán los usuarios conforme al rol que deban desempeñar en el entorno. Además de estos roles, el administrador tendrá la capacidad de crear otros roles según sea necesario, sin estar limitado exclusivamente a estos mostrados.

6.1 Configuración del playbook

El playbook (Figura 6-3) se configura especificando el host en el cual se ejecutarán las tareas, que en este caso corresponde el servidor LDAP, y el rol que se llevará a cabo, siendo en este caso el rol *users*.

```
---
- name: Definición de roles LDAP para el acceso a las máquinas virtuales.
  hosts: ldap_server
  #become: true
  roles:
    - users
```

Figura 6-3. Playbook de roles de usuarios

6.2 Configuración del archivo *main.yml* de las tareas

Este archivo (Figura 6-4) se encargará de incluir las tareas relacionadas con la creación o el borrado de un rol, siempre en función de una variable *booleana.* Se incluirá el contenido del archivo *create.yml* si el valor de la variable *create* es verdadero y *delete* falso. De manera opuesta, se incluirá el contenido del archivo *delete.yml*:

```yaml
---
# Definimos las tareas relacionadas con la definición de roles en el servidor ldap y las condiciones de su ejecución
- name: Crear rol
  include_tasks: create.yml
  when: create | bool and not delete

- name: Borrar rol
  include_tasks: delete.yml
  when: delete | bool and not create
```

Figura 6-4. Archivo main.yml de las tareas de definición de roles de usuarios

6.3 Creación de roles de usuarios

6.3.1 Desarrollo Python

Tabla 6-1. Características de la solicitud de crear un rol en el servidor LDAP

Método HTTP	POST
Endpoint	/createROL
Parámetros de la petición	**Nombre** del rol a crear
	Contraseña del administrador de LDAP

La configuración de la función que gestiona las solicitudes para crear un rol será bastante sencilla. En primer lugar, se extraen de la petición el rol que el administrador desea crear y la contraseña de administrador del servidor OpenLDAP, necesaria para poder crear roles en el servidor. Esta contraseña debe ser conocida por el administrador que vaya a crear el rol. Se verifica que ambos parámetros están presentes en la solicitud y se añaden al diccionario de variables, junto con la variable *create* con valor verdadero. Posteriormente, se ejecuta la función *run* de Ansible Runner y se verifican los posibles errores que puedan ocurrir, a partir del objeto devuelto por esta función.

```python
# Petición POST para crear un rol de usuario en el servidor LDAP para acceder a las máquinas virtuales
@app.post("/createROL")
def createROL():

    # Parámetros recibidos en la petición
    rol = request.form.get("rol")
    ldap_admin_password = request.form.get("ldap_admin_passwd")

    if rol is None or ldap_admin_password is None:
        abort(400,"Faltan parámetros obligatorios en la petición POST")

    # Creamos el diccionario de variables que le pasamos a Ansible
    variables = {"create": True, "delete": False, "rol": rol, "ldap_admin_passwd": ldap_admin_password}

    runner = ansible_runner.run(private_data_dir=".", playbook="users_playbook.yml",
                                extravars=variables, inventory="inventory/hosts")

    for events in runner.events:
        if(events["event"] == "runner_on_failed"):
            abort(500,"Error en la tarea " + str(events["event_data"]["task"]) + ": " + str(events["event_data"]["res"]))
        else:
            response = "Rol " + str(rol) + " creado correctamente en el servidor LDAP"

    return response
```

Figura 6-5. Código Python de crear un rol

6.3.2 Configuración de las tareas de Ansible

1. Instalar los paquetes necesarios (Figura 6-6):
    a. Paquete *Python3-ldap:* Módulo de interfaz LDAP para Python3, necesario para ejecutar el módulo de Ansible *ldap_entry*.

b. Demonio *slapd*: Servidor LDAP.

c. Paquete *ldap-utils*: Incluye diversas herramientas para interactuar con el servidor LDAP.

d. Paquete *debconf-utils*: Contiene herramientas necesarias para la preconfiguración de paquetes, indispensable para preconfigurar el servidor LDAP.

e. Los resultados de esta tarea se registran en una variable llamada *status_ldap.*

```
---
- name: Instalar dependencias
  become: true
  package:
    name: "{{ item }}"
    state: latest
  with_items:
    - python3-ldap
    - slapd
    - ldap-utils
    - debconf-utils
  register: status_ldap
```

Figura 6-6. Tarea instalar dependencias al crear roles de usuarios

En caso de que sea la primera vez que se interactúa con el servidor, será necesario instalar y configurarlo inicialmente. Se puede determinar si es la primera vez si se instala el demonio *slapd*, es decir, si el estado de esa tarea ha cambiado. Este valor se puede consultar mediante la variable *status_ldap* anterior. De ser así, se procederá al siguiente bloque de tareas que instala el servicio.

2. Configurar las respuestas a las preguntas durante la instalación silenciosa del servicio *slapd* (Figura 6-8). Durante la instalación del servicio, se lanzan una serie de preguntas para su configuración inicial. Esto puede automatizarse con Ansible usando el módulo ***debconf*** [47] indicando:

    a. Nombre del paquete.

    b. Pregunta que se realiza durante la instalación.

    c. Valor de la respuesta a la pregunta.

    d. Tipo de dato de la respuesta. (*vtype*)

Para determinar el formato de la pregunta, se utiliza el comando ***debconf-show*** (Figura 6-7) en el servidor donde se instalará el servicio LDAP, mostrando las preguntas realizadas durante la instalación del paquete especificado:

```
root@ldap-server:~# debconf-show slapd
* slapd/password2: (password omitted)
  slapd/internal/adminpw: (password omitted)
* slapd/password1: (password omitted)
  slapd/internal/generated_adminpw: (password omitted)
* shared/organization: TFG
* slapd/move_old_database: true
  slapd/ppolicy_schema_needs_update: abort installation
* slapd/purge_database: true
* slapd/organization: TFG
  slapd/upgrade_slapcat_failure:
  slapd/invalid_config: true
* slapd/no_configuration: false
* slapd/domain: tfg.local
  slapd/unsafe_selfwrite_acl:
  slapd/dump_database: when needed
  slapd/dump_database_destdir: /var/backups/slapd-VERSION
  slapd/password_mismatch:
```

Figura 6-7. Ejecución del comando debconf-show slapd en el servidor LDAP

- Contraseña del administrador del servicio LDAP (proporcionada por el administrador en la solicitud). Debe proporcionarse dos veces, siendo el valor de la respuesta una cadena de texto.
- Nombre del dominio a crear. Tipo *string* (cadena de texto).
- Nombre de la organización. Tipo *string*.
- Configuración automática de *slapd.* Tipo *booleano*, que al establecerse en *false* indica que se desea una configuración automatizada.
- Mover base de datos antigua a una ubicación de respaldo. Tipo *booleano* con valor *true* para preservar datos antiguos, si los hubiera.
- Purgar base de datos antigua. Tipo *booleano* con valor *true* para indicar que se desea borrar la base de datos antigua para comenzar con una instalación limpia sin datos previos. No obstante, la base de datos antigua será movida previamente a una ubicación de respaldo según la anterior respuesta.

```yaml
- name: Configuración inicial del servidor LDAP cuando sea la primera vez que se inicia
  become: true
  block:
    - name: Configurar las respuestas a las preguntas durante la instalación silenciosa del servidor LDAP
      debconf:
        name: slapd
        question: "{{ item.question }}"
        value: "{{ item.value }}"
        vtype: "{{ item.vtype }}"
      with_items:
        - { question: "slapd/password1", value: "{{ ldap_admin_passwd }}", vtype: "string"}
        - { question: "slapd/password2", value: "{{ ldap_admin_passwd }}", vtype: "string"}
        - { question: "slapd/domain", value: "tfg.local", vtype: "string"}
        - { question: "shared/organization", value: "TFG", vtype: "string"}
        - { question: "slapd/no_configuration", value: "false", vtype: "boolean"}
        - { question: "slapd/move_old_database", value: "true", vtype: "boolean"}
        - { question: "slapd/purge_database", value: "true", vtype: "boolean"}
      no_log: True
```

Figura 6-8. Tarea de configurar las respuestas durante la instalación del servicio LDAP

3. Reconfigurar *slapd* para cargar la configuración con las respuestas anteriores (Figura 6-9):
   a. Se ejecuta el comando: ***dpkg-reconfigure*** según se indica en [46].

```
  - name: Reconfigurar slapd para cargar configuración
    become: true
    command: dpkg-reconfigure -f noninteractive slapd
when: status_ldap["results"][1]["changed"] == True
```

Figura 6-9. Tarea de ejecutar el comando dpkg-reconfigure para cargar la configuración inicial del servidor LDAP

4. Crear una entrada/rol en forma del tipo Unidad Organizativa (Figura 6-10):
    a. Se utiliza el módulo de Ansible ***ldap_entry*** [48], que permite añadir o borrar entradas LDAP.
    b. Es necesario especificar el *Distinguished Name* del administrador, así como la contraseña de este.
    c. Se especifica el DN del objeto a crear, que en este caso el rol, de clase Unidad Organizativa y dentro del dominio en cuestión.
    d. Finalmente, se indica la dirección IP del servidor LDAP y el estado del objeto a crear, que en este caso es *present* para crearlo.

```
- name: Crear entrada (Unidad Organizativa)
  ldap_entry:
    bind_dn: cn=admin,dc=tfg,dc=local
    bind_pw: "{{ ldap_admin_passwd }}"
    dn: "ou={{ rol }},dc=tfg,dc=local"
    objectClass: organizationalUnit #Añadir "top" pq here
    server_uri: "ldap://{{ ldap_server }}/"
    state: present
```

Figura 6-10. Tarea de crear entrada/rol en el servidor LDAP

6.4 Borrado de roles de usuarios

6.4.1 Desarrollo Python

Tabla 6-2. Características de la solicitud para eliminar un rol del servidor LDAP

Método HTTP	DELETE
Endpoint	/deleteROL
Parámetros de la petición	**Nombre** del rol a eliminar
	Contraseña del administrador de LDAP

El desarrollo de la función que gestiona las solicitudes para eliminar un rol del directorio LDAP es similar a la función que maneja las peticiones para su creación. Los parámetros recibidos son el nombre del rol a eliminar y la contraseña del administrador de LDAP. La única diferencia radica en el valor de las variables (Figura 6-11) que se pasan a Ansible. En este caso, la variable *delete* es verdadera y *create* es falsa, indicando así que se deben ejecutar las tareas para eliminar un rol del servidor LDAP.

```
# Creamos el diccionario de variables que le pasamos a Ansible
variables = {"create": False, "delete": True, "rol": rol, "ldap_admin_passwd": ldap_admin_password}
```

Figura 6-11. Código Python del diccionario de variables al borrar un rol

6.4.2 Configuración de las tareas de Ansible

1. Instalación del paquete *python3-ldap.*
2. Asegurar que el servicio *slapd* está activo.
3. Eliminación de la Unidad Organizativa/Rol (Figura 6-12).
    a. Se utiliza el módulo ***ldap_entry*** de Ansible, de forma similar a cuando se crea el rol. La única diferencia es el parámetro *state*, que en este caso tendrá el valor *absent*, indicando que se desea eliminar la entidad especificada.

```
- name: Eliminar entrada (Unidad Organizativa)
  ldap_entry:
    bind_dn: cn=admin,dc=tfg,dc=local
    bind_pw: "{{ ldap_admin_passwd }}"
    dn: "ou={{ rol }},dc=tfg,dc=local"
    objectClass: organizationalUnit #Añadir "top" pq hereda de esta y alomejor da fallo
    server_uri: ldap://{{ ldap_server }}/
    state: absent
```

Figura 6-12. Tarea eliminar rol del servidor LDAP

7 RESULTADOS Y CONCLUSIONES

Si no puedes hacer grandes cosas, prueba a hacer pequeñas cosas de forma grandiosa.

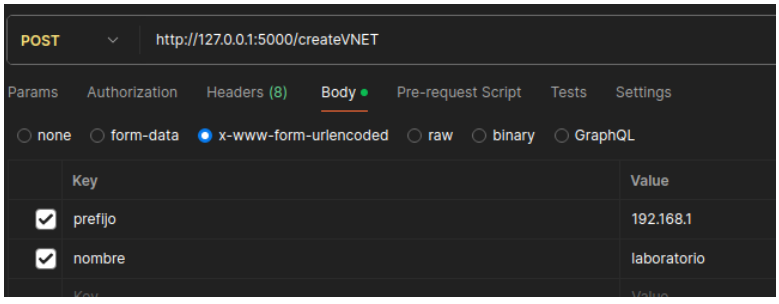
- Napoleón Hill -

Una vez configurado y desarrollado todo el proyecto, será necesario probar la ejecución de todas las funcionalidades posibles. En los capítulos siguientes se presentan los resultados obtenidos al ejecutar las distintas funcionalidades, además de proponer posibles mejoras y un apartado de conclusiones.

7.1 Resultados

Se detallan a continuación las funcionalidades más relevantes, tales como la ***creación*** de redes, máquinas virtuales y roles, así como la modificación de una máquina virtual. El proceso de borrado de estos elementos se considera trivial y no se expone en detalle.

- Creación de una red virtual (**¡Error! No se encuentra el origen de la referencia.**):
    - Nombre de la red: **laboratorio**
    - Prefijo de la red: **192.168.1.0/24**




```
<bridge name='br-laboratorio' stp='on' delay='0'/>
<mac address='52:54:00:9b:5c:f2'/>
<ip address='192.168.1.1' netmask='255.255.255.0'>
  <dhcp>
    <range start='192.168.1.2' end='192.168.1.254'/>
  </dhcp>
</ip>
```

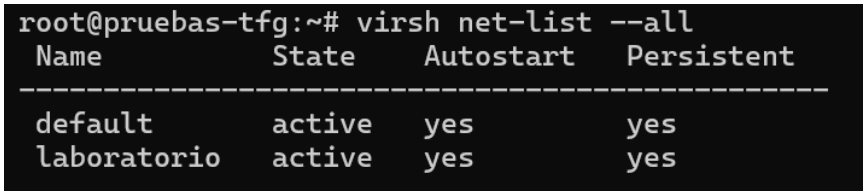


Figura 7-1. Resultados de crear una red virtual

- Creación de una máquina virtual predefinida en la red creada anteriormente (Figura 7-3):
  - Nombre de la máquina virtual: **marcanbra1**
  - Número de CPUs virtuales: **2**
  - Memoria RAM: **512 MB**
  - Nombre de la red virtual a la que conectarse: **laboratorio**
  - Sistema Operativo de la máquina: **Debian**

Al crear la máquina virtual a partir de un disco virtual predefinido, el disco tendrá un tamaño (Figura 7-3) predefinido de 2 GiB y estará ubicado en el pool *images* (Figura 7-2), tal y como fue configurado.

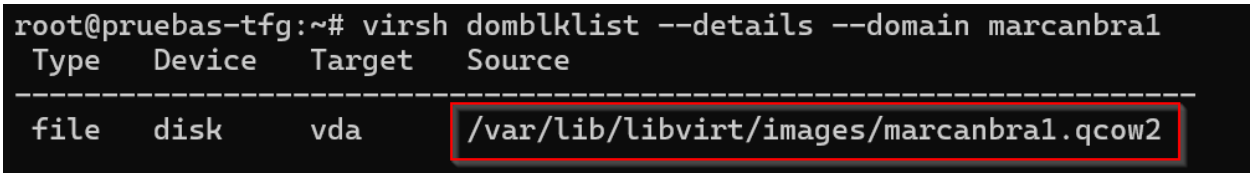


Figura 7-2. Ubicación del disco virtual de la máquina virtual creada

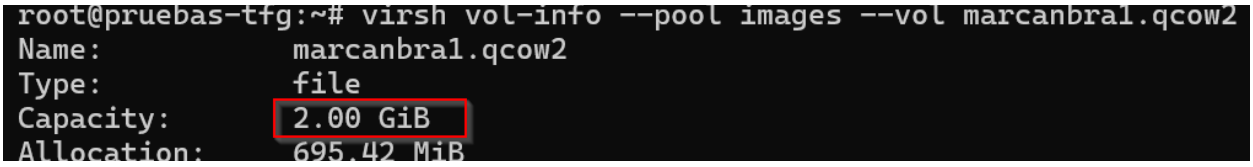


Figura 7-3. Verificación del tamaño correcto del disco de una máquina creada.

Además, se verifica que la máquina está efectivamente conectada a la red *laboratorio*, tal y como se puede observar en la siguiente imagen:

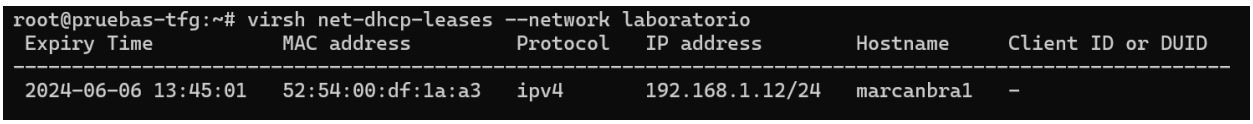


Figura 7-4. Resultado de crear una máquina virtual predefinida conectada a una red previamente creada

- Creación de una máquina virtual a partir de un disco nuevo:
  - Nombre de la máquina virtual: **marcanbra2**
  - Número de CPUs virtuales: **1**
  - Memoria RAM: **512 MB**
  - Red virtual a la que se conectará: **laboratorio**
  - Sistema Operativo a que se instalará: **CentOS**
  - Tamaño del disco que se creará: **6 GB**

En este caso (Figura 7-5), la máquina virtual creada tiene asociada una imagen ISO (*CentOS.iso*) del sistema operativo, la cual se utilizará al arrancar la máquina por primera vez. También se observa que el disco se ha creado dentro del pool para las máquinas CentOS (*CentOS_Guests*), tal y como se configuró, y que la máquina

está efectivamente conectada a la red *laboratorio*.

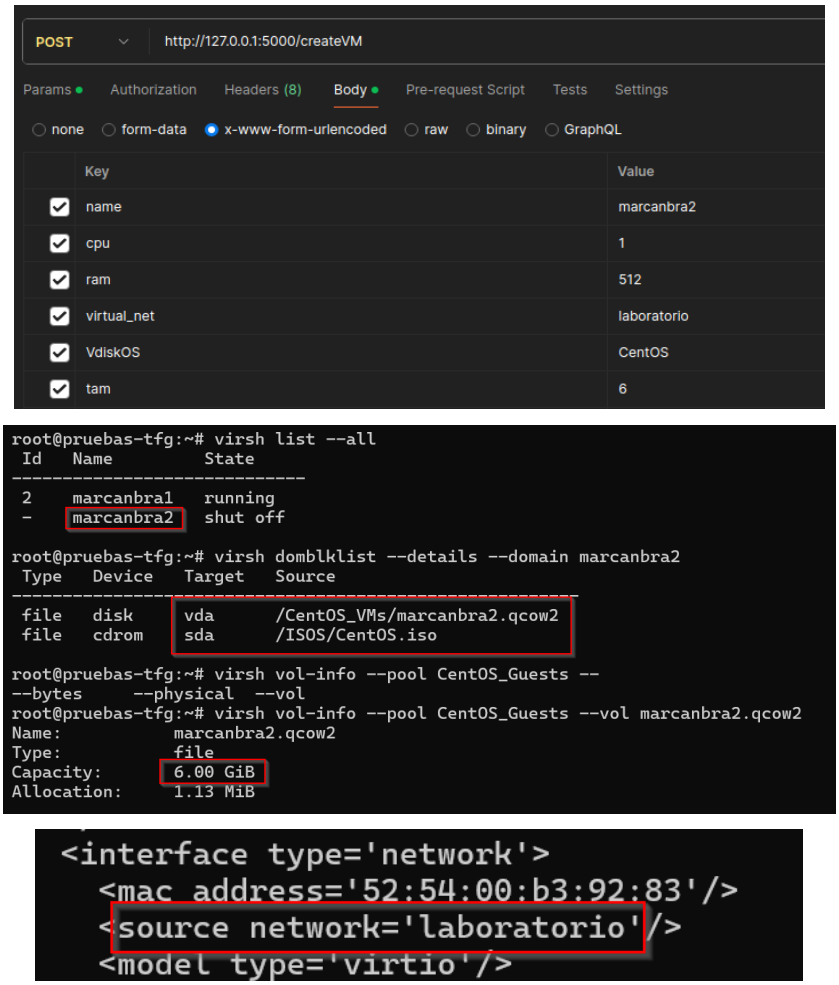


Figura 7-5. Resultados de crear una máquina virtual a partir de un disco nuevo

- Crear roles de *Profesor* y *Alumnos* en forma de objetos de tipo *Unidad Organizativa* en el servidor LDAP.

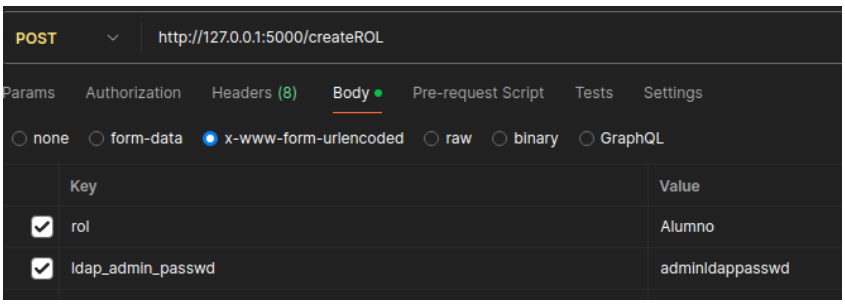


Figura 7-6. Uso de Postman para enviar una solicitud de crear un rol de alumno

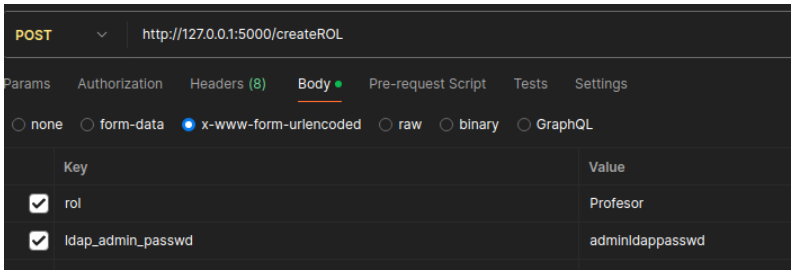


Figura 7-7. Uso de Postman para enviar una solicitud de crear un rol de profesor

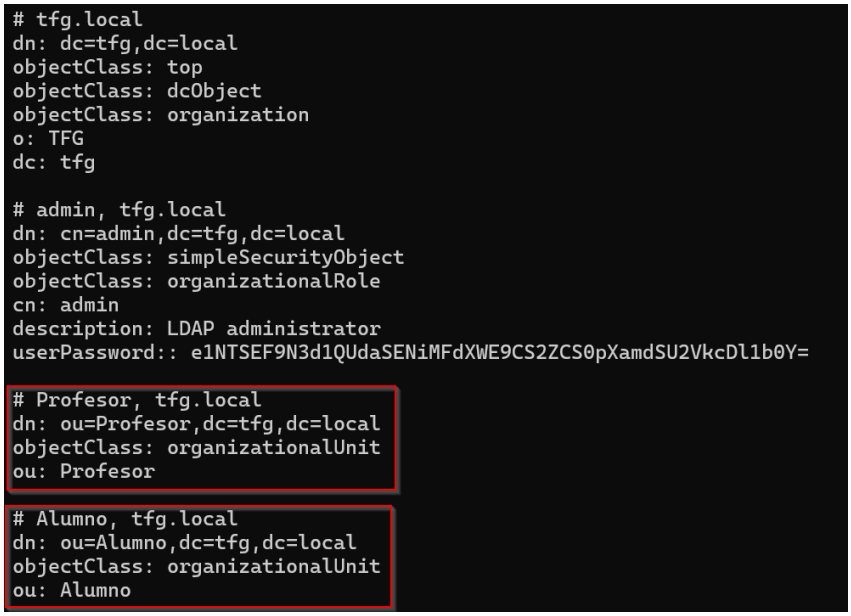
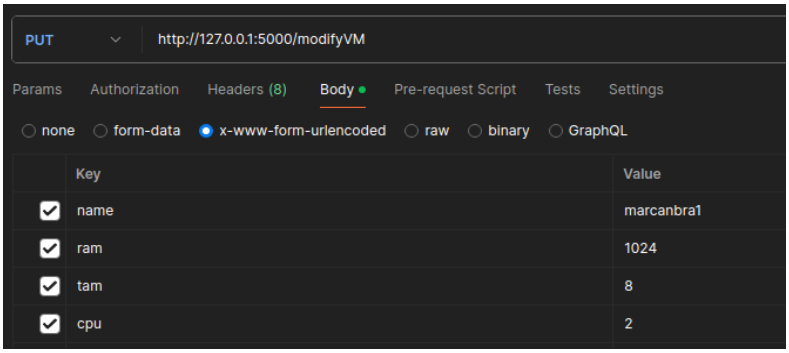
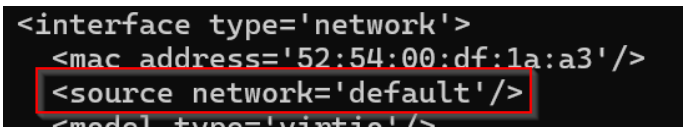


Figura 7-8. Resultados de crear los dos roles en el servidor LDAP

- Modificar máquina virtual predefinida *marcanbra1* (Figura 7-9):
  - Nueva cantidad de memoria RAM a asignar: **1024 MB**
  - Nuevo tamaño del disco: **8 GB**
  - Nuevo número de CPUs virtuales: **2**
  - Nueva red a la que conectarse: **default**





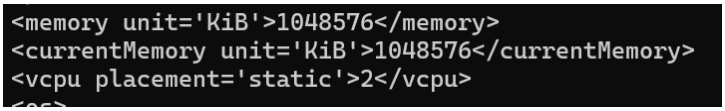


Figura 7-9. Resultados de modificar una máquina virtual

Al no indicar una red específica en la solicitud, se modificará automáticamente este apartado de la máquina, conectándola a la red por defecto.

7.2 Posibles mejoras

En este capítulo se exponen algunas mejoras que podrían implementarse en el futuro para el proyecto. La mayoría están relacionadas con aspectos de seguridad, un factor cada vez más importante en nuestra sociedad, pero que por no estar incluido en el alcance de este proyecto no se han implementado.

- **Cifrado de las contraseñas almacenadas en texto claro**: Utilizando *Ansible Vault* [49], una característica de Ansible que permite almacenar datos sensibles como contraseñas o claves en ficheros encriptados, en lugar de en texto plano en *playbooks* o roles. Como se observa en la Figura 3-4, en nuestro proyecto se indican las contraseñas de los usuarios de los servidores remotos en una variable en texto claro en el inventario, lo cual no es adecuado en términos de seguridad. Cualquiera que tenga acceso al código del proyecto podrá ver las contraseñas de esos servidores, permitiendo un acceso no autorizado a estos.
- **Validación de campos introducidos por el administrador:** Esto incluye validar los parámetros que el servidor REST recibe en cada solicitud, verificando que el tipo de dato sea el esperado y que no se han introducido caracteres que pudieran comprometer el servidor. Actualmente, se toma la entrada proporcionada por el administrador y se pasa a Ansible sin verificar su contenido, lo cual es una mala práctica. Por tanto, implementar una validación adecuada sería un ejercicio importante para la futura securización del código de la API REST.
- **Posibilidad de crear o eliminar varias redes, máquinas o roles a la vez:** El proyecto actual permite la creación o eliminación de una única red, máquina o rol por cada solicitud. La respuesta a cada solicitud suele ser lenta ya que Ansible debe ejecutar un conjunto de tareas por detrás. Para agilizar este proceso, una buena idea sería implementar la capacidad de incluir varias redes, máquinas o roles en una sola solicitud, permitiendo realizar múltiples acciones con un único envío.
- **Mostrar las redes, máquinas o roles existentes:** Debido a que el desarrollo de la parte *front-end* de la API no se ha abordado, se asume que el administrador que la use conoce las redes, máquinas y roles existentes en los servidores, y los proporciona en la solicitud. En un entorno real, sería útil mostrar esta información en la parte del *front-end,* implementando una opción que realizase una petición **GET** a la API y obtuviera todos los elementos existentes en el servidor. De esta manera, el administrador podría seleccionar sobre estos elementos las acciones a realizar. No obstante, en el código desarrollado, si se proporciona un elemento inexistente, se lanzará un error en la respuesta, intentando así hacerlo de la forma más realista posible.

7.3 Conclusiones

El proyecto ha logrado cumplir con éxito los objetivos planteados inicialmente, demostrando la efectividad y flexibilidad de las tecnologías empleadas. La implementación de una API REST ha permitido gestionar de manera eficiente un entorno virtualizado, utilizando herramientas como Ansible y desarrollos en Python.

En primer lugar, la creación y configuración de redes virtuales mediante Ansible han mostrado ser altamente efectivas. Las redes se configuraron correctamente. Además, el uso de plantillas y *playbooks* de Ansible facilitó la automatización de estas tareas, reduciendo el esfuerzo manual y minimizando los errores humanos.

La gestión de máquinas virtuales fue otro aspecto destacado del proyecto. Se logró crear máquinas virtuales tanto a partir de discos preexistentes como generando nuevos discos virtuales, lo que proporciona capacidad de elección en el tipo de máquina a crear. El sistema implementado permite no solo la creación sino también la eliminación y modificación de estas máquinas. Estas funcionalidades son fundamentales para la gestión dinámica y eficiente de los recursos de un entorno virtualizado.

Además, la integración con un servidor LDAP para la administración de roles de usuarios añade una capa adicional de organización. La capacidad de crear y eliminar roles de usuarios a través de la API REST asegura que solo los usuarios autorizados tengan acceso a las máquinas virtuales creadas, gestionándose así el acceso a estas.

El uso de tecnologías como Flask para la creación de la API REST y Postman para la prueba y validación de

las solicitudes ha sido esencial para el correcto funcionamiento de todo el proyecto. Flask proporcionó un marco ligero pero poderoso para desarrollar la API, mientras que Postman aseguró que todas las funcionalidades operaran correctamente antes de su despliegue final.

Por último, el alquiler de VPS en la nube ha sido fundamental para las pruebas del proyecto, dado que mi ordenador personal no permitía el uso simultáneo de varias máquinas debido a la limitada memoria RAM disponible. Afortunadamente, conocía **Clouding.io**, un proveedor de VPS que usé en trabajos anteriores durante la carrera, gracias a los 5€ que regalan al registrarse. No solo ofrecen precios asequibles, sino que también proporcionan virtualización anidada, lo cual era clave a la hora de buscar una plataforma de VPS para este proyecto. A pesar de tener que incurrir en gastos para alquilar los servidores privados, considero que fue una decisión e inversión acertada, ya que apenas tuve problemas y el servicio de https://clouding.io/ fue excelente en todo momento.

En resumen, la combinación de diferentes tecnologías empleadas ha permitido crear un sistema flexible, capaz de adaptarse a las necesidades dinámicas de un entorno de TI moderno. Este trabajo no solo representa un logro en términos técnicos, sino que también subraya la importancia de la automatización y la buena organización en el desarrollo de sistemas complejos, con espacio para mejora para funciones adicionales.

Anexo A: Puesta en marcha del proyecto

En este anexo se muestran los pasos a seguir para poner en marcha la API REST y los servidores que se gestionarán. Este proyecto se ha probado en sistemas Linux, específicamente en las distribuciones y versiones mostradas en la Figura 6-1 para cada actor implicado. Por lo tanto, se recomienda probarlo en las mismas versiones para evitar problemas. No obstante, probarlo en otras distribuciones y versiones será una opción para desarrollar en el futuro.

1. **Instalar Ansible** (3.2 Instalación de Ansible) y **Ansible Runner** (3.3 Instalación de Ansible Runner) en el nodo administrador.
2. **Incluir** en el **inventario** las **direcciones IP** de los servidores a gestionar (Figura 3-4).
3. **Indicar** en los *playbooks* el **grupo de hosts** o direcciones IP sobre los que se ejecutarán las tareas correspondientes (Figura 4-3, Figura 5-2 y Figura 6-3).
4. **Generar** las **claves públicas de SSH** de los usuarios *root* y del usuario que ejecute el servicio en el servidor remoto, y copiarlas al archivo *authorized_keys* de los servidores remotos a gestionar (Capítulo 3.5 Generación de la clave pública).
5. Asegurarse de que los **usuarios** de los servidores remotos con los que Ansible accede están **incluidos** en el archivo */etc/sudoers* de cada servidor, para que puedan **ejecutar tareas** con **privilegios** de administrador si es necesario.
6. Asegurarse de que los servidores remotos a gestionar tienen el **servicio SSH activo**.
7. **Lanzar** el **servidor REST** con el siguiente comando (por defecto escuchará las peticiones en el puerto 5000):

```
python3 -m http.server -app Server_REST run
```

8. Lanzar la solicitud con los parámetros adecuados y esperar a que termine la ejecución.

A continuación, se describen los requisitos mínimos en cada en cada equipo que interviene en el proyecto, según la Figura 6-1:

- **Nodo administrador y servidor REST:**
  - Sistema Operativo: **Ubuntu 22.04**
  - Python 3.10.12 (3.1 Instalación de Python)
  - Flask 3.0.2 (3.1 Instalación de Python)
  - Ansible 2.16.5 (3.2 Instalación de Ansible)
  - Ansible Runner 2.3.5 (3.3 Instalación de Ansible Runner)
  - Cliente OpenSSH_8.9p1 Ubuntu-3ubuntu0.6, OpenSSL 3.0.2 15 Mar 2022 (3.4 Instalación del cliente SSH)

- **Servidor de virtualización:**
  - Sistema Operativo: **Ubuntu 20.04**
  - Servidor OpenSSH_8.2p1 Ubuntu-4ubuntu0.5, OpenSSL 1.1.1f 31 Mar 2020
  - Python 3.8.10
  - Libvirtd 6.0.0 (Figura 4-11)
  - Python3-libvirt (Figura 4-11)
  - Python3-lxml (Figura 4-11)
  - Libguestfs-tools 1:1.40.2-7ubuntu5 (Figura 5-10)
  - Virt-install 2.2.1 (Figura 5-10)
- **Servidor LDAP:**
  - Sistema Operativo: **Ubuntu 20.04**
  - Python3-ldap 3.2.0 (Figura 6-6)
  - OpenLDAP: slapd (Ubuntu) (Figura 6-6)
  - Ldap-utils 2.4.49+dfsg-2ubuntu1.10 (Figura 6-6)
  - Debconf-utils 1.5.73 (Figura 6-6)

REFERENCIAS

[1] BBVA, «API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos,» 23 Marzo 2016. [En línea]. Available: https://www.bbvaapimarket.com/es/mundo-api/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos/.

[2] Red Hat, «¿Qué es una API REST?,» 31 Julio 2023. [En línea]. Available: https://www.redhat.com/es/topics/api/what-is-a-rest-api.

[3] Red Hat, «¿Qué es un hipervisor?,» 23 Marzo 2023. [En línea]. Available: https://www.redhat.com/es/topics/virtualization/what-is-a-hypervisor.

[4] A. J. C. Maestre, «Repositorio Institucional UOC,» 5 Enero 2024. [En línea]. Available: https://openaccess.uoc.edu/bitstream/10609/149530/2/acorderomaTFG0124memoria.pdf.

[5] Ansible, «Introducción a Ansible - Documentación de la comunidad de Ansible,» 24 Abril 2024. [En línea]. Available: https://docs.ansible.com/ansible/latest/getting_started/introduction.html.

[6] Red Hat, «¿Qué es YAML?,» 3 Marzo 2023. [En línea]. Available: https://www.redhat.com/es/topics/automation/what-is-yaml.

[7] R. U. Pulido, «Kubernetes: Crear un fichero YAML - Blog Virtualizacion,» 3 Febrero 2020. [En línea]. Available: https://www.maquinasvirtuales.eu/kubernetes-crear-un-fichero-yaml/.

[8] atareao, «El inventario de ansible,» 12 Junio 2020. [En línea]. Available: https://atareao.es/tutorial/ansible/el-inventario-de-ansible/.

[9] I. J. Valencia, «TFG-Final,» Julio 2022. [En línea]. Available: https://ddd.uab.cat/pub/tfg/2021/tfg_129630/TFG-Final.pdf.

[10] Ansible, «How to build your inventory - Ansible Community Documentation,» 21 Junio 2024. [En línea]. Available: https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html.

[11] Ansible, «Creating a Playbook - Ansible Community Documentation,» 24 Abril 2024. [En línea]. Available: https://docs.ansible.com/ansible/latest/getting_started/get_started_playbook.html.

[12] Red Hat, «¿Qué son los playbooks de Ansible?,» 1 Agosto 2023. [En línea]. Available: https://www.redhat.com/es/topics/automation/what-is-an-ansible-playbook#:~:text=Los%20playbooks%20de%20Ansible%C2%AE,dispositivo%20en%20el%20quedebe%20hacerlo..

[13] Red Hat, «Conceptos básicos de la automatización con Ansible,» 21 Junio 2022. [En línea]. Available: https://www.redhat.com/es/topics/automation/learning-ansible-tutorial#:~:text=Las%20tareas%20son%20los%20elementos,seguimiento%20del%20estado%20del%20sistema..

[14] Red Hat, «Los módulos de Ansible y su funcionamiento,» 30 Abril 2024. [En línea]. Available: https://www.redhat.com/es/topics/automation/what-is-an-ansible-module.

[15] Ansible, «Roles - Ansible Community Documentation,» 24 Abril 2024. [En línea]. Available: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_reuse_roles.html.

[16] Ansible Runner, «Ansible Runner Documentation,» [En línea]. Available: https://ansible.readthedocs.io/projects/runner/en/latest/.

[17] J. D. Muñoz, «Que es Flask y ventajas que ofrece,» 17 Noviembre 2017. [En línea]. Available: https://openwebinars.net/blog/que-es-flask/.

[18] Cloudfare, «¿Qué es SSH?,» [En línea]. Available: https://www.cloudflare.com/es-es/learning/access-management/what-is-ssh/.

[19] F. Donkers, «KVM y Libvirt,» 9 Febrero 2019. [En línea]. Available: https://docs.slackware.com/es:howtos:general_admin:kvm_libvirt?do=export_pdf.

[20] QEMU, «QEMU,» 9 Julio 2020. [En línea]. Available: https://wiki.qemu.org/Main_Page.

[21] «Introducción a Libvirt + QEMU/KVM,» 28 Septiembre 2022. [En línea]. Available: https://fp.josedomingo.org/hlc2122/u01/introduccion.html.

[22] Ansible, "Installing Ansible on specific operating systems - Ansible Community Documentation," 24 Abril 2024. [Online]. Available: https://docs.ansible.com/ansible/latest/installation_guide/installation_distros.html#installing-ansible-on-ubuntu.

[23] Ansible Runner, «Installing Ansible Runner - Ansible Runner Documentation,» [En línea]. Available: https://ansible.readthedocs.io/projects/runner/en/latest/install/#using-pip.

[24] Ansible, «How to build your inventory - Ansible Community Documentation,» 24 Abril 2024. [En línea]. Available: https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html#adding-variables-to-inventory.

[25] Red Hat, «Configuración de las conexines de red de las máquinas virtuales,» [En línea]. Available: https://access.redhat.com/documentation/es-es/red_hat_enterprise_linux/8/html/configuring_and_managing_virtualization/configuring-virtual-machine-network-connections_configuring-and-managing-virtualization.

[26] «Usando QEMU/KVM con libvirt - Redes Virtuales,» 13 Agosto 2021. [En línea]. Available: https://www.pinguytaz.net/index.php/2021/08/13/usando-qemu-kvm-con-libvirt-3-x-redes-virtuales/.

[27] [En línea]. Available: https://gerrit.opencord.org/plugins/gitiles/cord/+/a1f5508898a44aaf3c6dbd0e1134fe65dc25af5f/ansible/roles/virt-nets/templates/virt_net.xml.j2.

[28] «Web Services REST con Python y REST,» 28 Enero 2023. [En línea]. Available: https://chuidiang.org/index.php?title=Web_Services_REST_con_Python_y_Flask.

[29] Ansible Runner, «Introduction to Ansible Runner - Ansible Runner Documentation,» [En línea]. Available: https://ansible.readthedocs.io/projects/runner/en/latest/intro/#inputdir.

[30] Ansible, «package - Generic OS packet manager - Ansible Documentation,» 27 Mayo 2022. [En línea]. Available: https://docs.ansible.com/ansible/2.9/modules/package_module.html.

[31] Ansible, «virt_net – Manage libvirt network configuration - Ansible Documentation,» 27 Mayo 2022. [En línea]. Available: https://docs.ansible.com/ansible/2.9/modules/virt_net_module.html.

[32] Red Hat, «¿Qué es KVM?,» 11 Mayo 2022. [En línea]. Available: https://www.redhat.com/es/topics/virtualization/what-is-KVM.

[33] Amazon Web Services, «¿Qué es una máquina virtual basada en kernel?,» [En línea]. Available: https://aws.amazon.com/es/what-is/kvm/.

[34] Red Hat, «Gestión del Almacenamiento para máquinas virtuales,» [En línea]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_virtualization/managing-storage-for-virtual-machines_configuring-and-managing-virtualization#understanding-storage-pools_understanding-virtual-machine-s.

[35] Red Hat, «Build a lab in 36 seconds with Ansible,» 22 Octubre 2021. [En línea]. Available: https://www.redhat.com/sysadmin/build-VM-fast-ansible.

[36] Red Hat, «Build a lab in 5 minutes with three commands,» 20 Agosto 2021. [En línea]. Available: https://www.redhat.com/sysadmin/build-lab-quickly.

[37] Ansible, «Service modulo - Manage Services - Ansible Community Documentation,» 16 Mayo 2024. [En línea]. Available: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/service_module.html.

[38] Ansible, «Virt modulo - Manages Virtual Machines suported by libvirt,» 16 Mayo 2024. [En línea]. Available: https://docs.ansible.com/ansible/latest/collections/community/libvirt/virt_module.html.

[39] Ansible, «get_url module - Download files from HTTP, HTTPS and FTP - Ansible Community Documentation,» 16 Mayo 2024. [En línea]. Available: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/get_url_module.html.

[40] Ansible, «Copy module - Copy files to remote locations - Ansible Documentation,» 16 Mayo 2024. [En línea]. Available: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/copy_module.html.

[41] Ansible, «File module - Manage files and file properties - Ansible Community Documentation,» 16 Mayo 2024. [En línea]. Available: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/file_module.html.

[42] Ansible, «Virt_pool - Manage libvirt storage pools - Ansible Documentation,» 27 Mayo 2022. [En línea]. Available: https://docs.ansible.com/ansible/2.9/modules/virt_pool_module.html.

[43] Ansible, «Fetch module - Fetch files from remote nodes - Ansible Community Documentation,» 23 Mayo 2024. [En línea]. Available: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/fetch_module.html.

[44] Ansible, «XML Module - Manage bits and pieces of XML files or strings,» 23 Mayo 2024. [En línea]. Available: https://docs.ansible.com/ansible/latest/collections/community/general/xml_module.html.

[45] Ansible, «lineinfile Module - Manage lines in text files - Ansible Community Documentation,» [En línea]. Available: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/lineinfile_module.html.

[46] Merkasys, «Curso OpenLDAP,» 14 Diciembre 2014. [En línea]. Available: https://www.youtube.com/watch?v=Zmj6A5ggcgg.

[47] Ansible, «debconf - Configure a .deb package - Ansible Documentation,» 27 Mayo 2022. [En línea]. Available: https://docs.ansible.com/ansible/2.9/modules/debconf_module.html.

[48] Ansible, «ldap_entry - Add or remove LDAP entries - Ansible Documentation,» 22 Septiembre 2020. [En línea]. Available: https://docs.ansible.com/ansible/2.9_ja/modules/ldap_entry_module.html.

[49] Ansible, «Protecting Sensitive Data with Ansible Vault - Ansible Community Documentation,» 4 Junio 2024. [En línea]. Available: https://docs.ansible.com/ansible/latest/vault_guide/index.html.

[50] Ansible, «Virt pool - Manage libvirt storage pool - Ansible Documentation,» 27 Mayo 2022. [En línea]. Available: https://docs.ansible.com/ansible/2.9/modules/virt_pool_module.html.