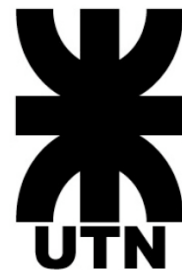


Trabajo Práctico Integrador: Virtualización con VirtualBox



Servidor hospedado en una VM para
almacenamiento, listado, eliminación y descarga de archivos.

Alumnos:

- Patricio Sussini Guanzioli – sussiniguanzioli@gmail.com
- Matias Ezequiel Vazquez – vazquez.matias.e@gmail.com

Materia: Arquitectura y Sistemas Operativos

Profesor: Osvaldo Falabella

Fecha de Entrega: *05/06/2025*

Índice

| | |
|---|-----------|
| Introducción: Elección de tópicos para elaborar el Trabajo Práctico Final. | 2 |
| Marco Teórico: La Virtualización como herramienta | 3 |
| Caso Práctico: MyCloud-VM-Server | 6 |
| Demostración práctica de la simulación | 7 |
| Metodología Utilizada: Marco teórico práctico | 15 |
| 1. Investigación previa: | 15 |
| 2. Diseño del proyecto: | 15 |
| 3. Preparación del entorno virtual: | 16 |
| 4. Instalación del Software necesario: | 16 |
| 5. Desarrollo de la interfaz y servidor: | 17 |
| 6. Pruebas: | 17 |
| 7. Errores y soluciones: | 17 |
| (figura 12) | 18 |
| Herramientas utilizadas: | 18 |
| Resultados Obtenidos | 19 |
| Bibliografía | 21 |
| Anexos | 21 |

Introducción: Elección de tópicos para elaborar el Trabajo Práctico Final.

La virtualización hoy en día es una tecnología clave, ya que permite correr varios sistemas operativos en una misma máquina física, lo cual optimiza recursos y facilita la labor de quienes desarrollan o administran sistemas. En este trabajo se decidió abordar el tema en cuestión, porque es algo que aparece con frecuencia tanto en entornos laborales (principalmente en roles como los de sysadmin o devops), como educativos. Conocer profundamente cómo funciona y sus principales aplicaciones ayuda a tener una visión más completa del rol del programador en la actualidad.

El desarrollo y diseño de este proyecto es una tarea desafiante, que lleva al límite la capacidad de análisis de un Técnico en programación en proceso de formación. Tareas claves como: configurar entornos de trabajo y desarrollo, establecer un diseño claro del alcance del proyecto y los sistemas a desarrollar, generar conocimiento sobre nuevas herramientas, ponen a prueba la capacidad de análisis individual de cada uno. Estas tareas exigentes le proveen al técnico en programación en desarrollo, un desafío y sobre todo un ambiente apto, como tierra fértil, para promover la auto-superación, un salto hacia adelante en calidad profesional y habilidades de toda índole. El objetivo principal de nuestro proyecto de Virtualización es mostrar, a través de una aplicación práctica, cómo se puede usar una máquina virtual para hospedar un servidor web. Para ello, se utilizará Oracle VirtualBox, donde mediante Ubuntu como sistema operativo se configurará un servidor Apache, el cual actuará como proxy inverso para conectar la lógica impulsada por Flask (Python) con un frontend construido en ReactJS. Toda la aplicación funcionará dentro de la máquina virtual y podrá ser accedida desde otra máquina en la red local, como si se tratara de un servidor real.

Esto permite no solo poner en práctica conceptos de programación, sino también aprender sobre redes, servidores, y cómo conectar distintos componentes de una aplicación web en un entorno virtualizado, reforzando así conceptos sobre infraestructura y despliegue de aplicaciones.

Marco Teórico: La Virtualización como herramienta

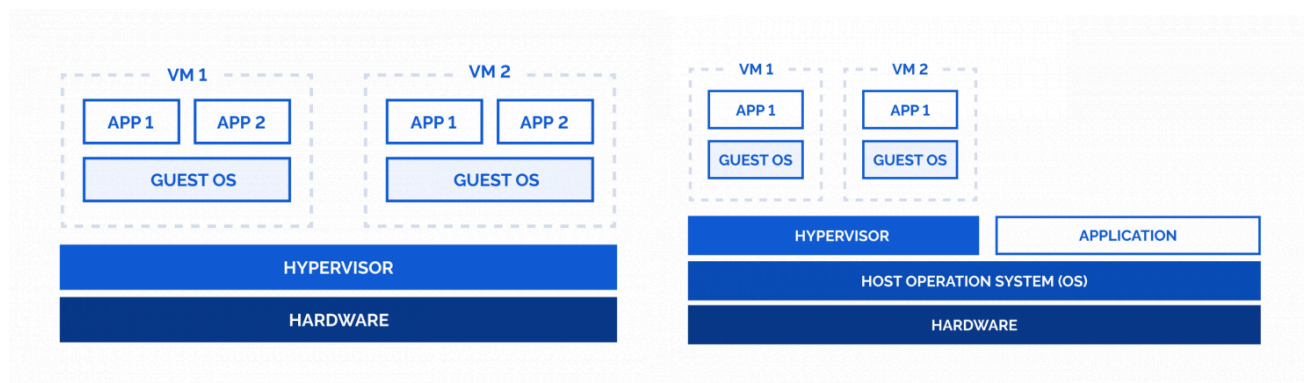
La virtualización es una tecnología que permite crear entornos informáticos virtuales sobre una infraestructura física. Mediante software especializado, se simulan hardware o servicios que funcionan de manera independiente del equipo físico subyacente. Esto permite ejecutar múltiples sistemas operativos o aplicaciones en un solo dispositivo físico, mejorando la eficiencia y el uso de recursos.

El concepto surgió en la década de 1960 con los mainframes de IBM, donde se buscaba optimizar el uso de equipos. Con el tiempo, y especialmente desde principios del siglo XXI, la virtualización se ha expandido ampliamente en entornos empresariales gracias a tecnologías como VMware, Hyper-V, y más recientemente los contenedores como Docker.

Los principales tipos de virtualización son:

- **Virtualización de hardware**, mediante hipervisores que permiten crear máquinas virtuales independientes.
- **Virtualización a nivel de sistema operativo**, que permite ejecutar múltiples contenedores aislados sobre un mismo kernel.
- **Virtualización de red y almacenamiento**, que abstraen estos recursos para una gestión más flexible.

Uno de los elementos centrales es el **hipervisor**, un software que permite la creación y gestión de máquinas virtuales. Existen dos tipos: los hipervisores de tipo 1 (montado), que corren directamente sobre el hardware, y los de tipo 2, que funcionan sobre un sistema operativo anfitrión.



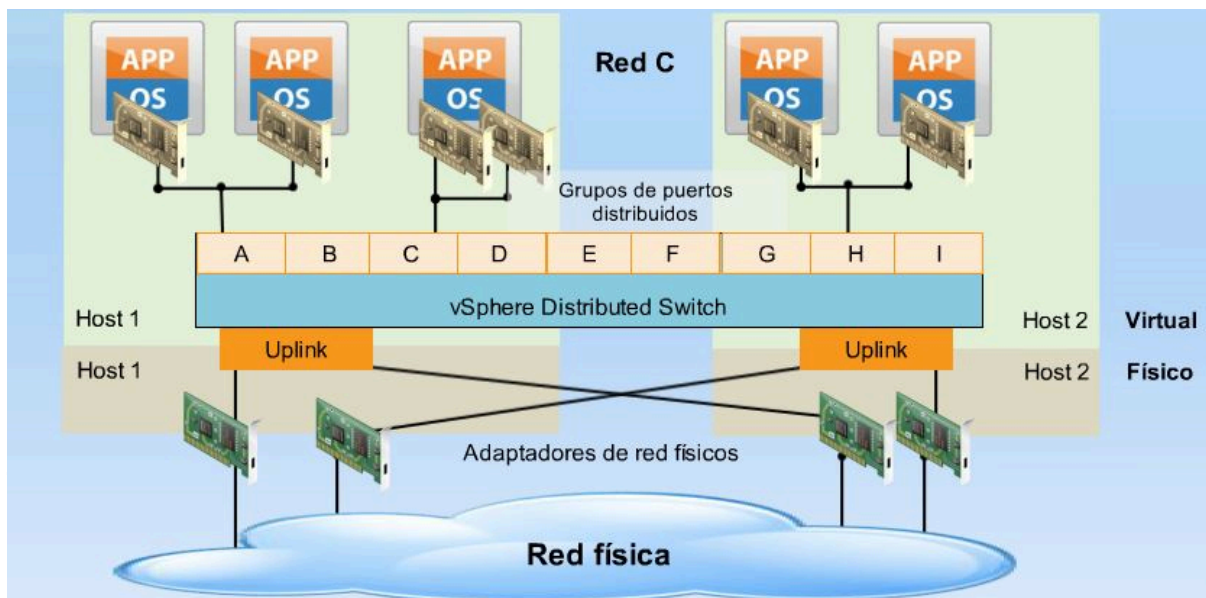
Componentes clave:

- Hipervisor
- Máquinas virtuales
- Contenedores
- Sistemas de orquestación
- **Imágenes ISO**
- **Redes Virtuales**

Una imagen ISO es un archivo que contiene la copia exacta del contenido de un disco óptico. En el entorno de la virtualización, las imágenes ISO son fundamentales ya que permiten instalar sistemas operativos o software en máquinas virtuales, desplazando la necesidad de utilizar medios físicos. En el contexto virtual, una imagen ISO se monta como si fuera un disco físico en el hipervisor, lo que permite iniciar la máquina virtual directamente desde ese archivo e iniciar la instalación del SO.



Comprendemos como Red Virtual a la configuración en “modo puente” que permite que una máquina virtual se muestre como un dispositivo físico más dentro de la red local. Con esta configuración, el sistema virtualizado obtiene una dirección IP del mismo rango que el anfitrión físico, permitiendo comunicación directa con los otros dispositivos, como si fuera



uno más de ellos. De esta forma puede ser accedida por los demás dispositivos y se comporta como un nodo independiente.

(imagen de referencia, [documentación de VMWare](#))

Las principales ventajas de la virtualización son: la reducción de costos operativos, el uso más eficiente del hardware, la escalabilidad rápida y la facilidad para crear entornos de prueba, entre otras. Sin embargo, también presenta desafíos como una posible degradación del rendimiento debido al sobrecarga del sistema, complejidad en la administración y mantenimiento además de vulnerabilidades de seguridad específicas.

Actualmente, la virtualización es una piedra angular de la computarización en la nube, del desarrollo ágil y de los entornos de alta disponibilidad, siendo indispensable en sectores que requieren infraestructura flexible, escalable y replicable.

Referencias:

[Timeline of Virtualization](#) - Wikipedia

[¿Qué es la Virtualización?](#) - Amazon AWS

[¿Qué es un hipervisor?](#) - [Cloudzy.com](#)

[Tipos de hipervisores](#) - StarWind Hyperconvergence

[Redes Virtuales](#) - Oracle VirtualBox Docs

Caso Práctico: *MyCloud-VM-Server*

La solución práctica simulada en este trabajo se desarrolló en un entorno virtual y controlado. El proyecto cuenta centralmente con un servidor HTTP, capaz de ofrecer funcionalidades básicas de almacenamiento en la *nube* para usuarios dentro de una red local. Entre las funciones se destacan: carga, descarga, visualización y eliminación de archivos almacenados. Esta etapa tiene como objetivo principal servir como fase de pruebas antes de realizar un despliegue definitivo en un servidor dedicado.

Para ello se utilizaron diversas herramientas, empezando por **VirtualBox** como hipervisor tipo 2, para crear una máquina virtual en base al sistema operativo **Ubuntu Desktop 24.04** la cual cumple el propósito de albergar la instalación y configuración de **Apache2** como servidor web. **Apache** cumple el rol de **proxy inverso**, redirigiendo el tráfico entrante hacia un servidor de aplicaciones desarrollado con **Flask** (Python), que trabaja en el ámbito local de la misma máquina virtual.

Flask (indicado previamente) se encarga de manejar la lógica del servidor *back-end*, ofreciendo una **API REST** que permite cargar, descargar, listar y eliminar archivos. A su vez, el desarrollo de la interfaz con la que interactúa el usuario fue mediante código Javascript en **ReactJS**. La misma en comunicación con esta API, brinda la información actualizada y se encarga de proveer una experiencia de usuario moderna y flexible en múltiples dispositivos.

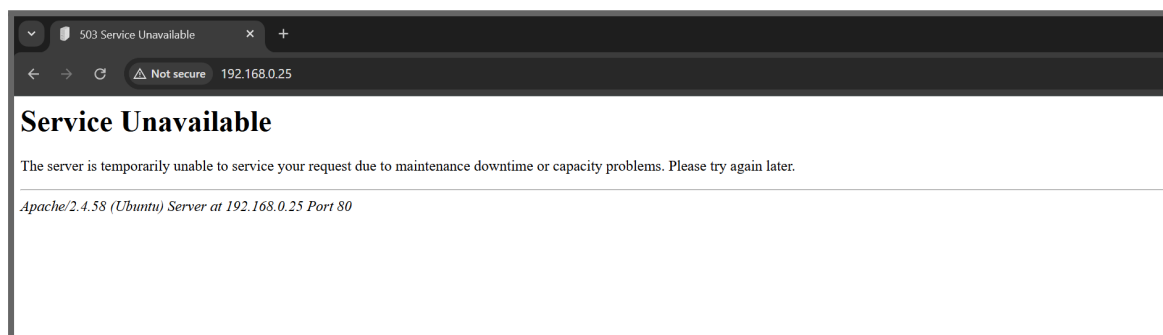
La elección de utilizar Apache como proxy inverso responde al interés de aplicar buenas prácticas ampliamente adoptadas en entornos de desarrollo y producción web, así como de experimentar con tecnologías comúnmente utilizadas para la gestión de servidores HTTP. Si bien era técnicamente posible acceder a la aplicación desde la máquina host mediante Flask ejecutándose de forma directa, se optó por la opción incorporar Apache para implementar y comprender el funcionamiento de un proxy inverso. Esta decisión permite no solo centralizar y controlar el acceso a los recursos, sino también sentar las bases para una arquitectura más escalable y modular. Gracias a esta configuración, cualquier dispositivo conectado a la misma red local puede acceder a la aplicación simplemente ingresando la dirección IP de la máquina virtual en el navegador, sin necesidad de especificar puertos ni detalles del backend.

El funcionamiento fue validado desde múltiples dispositivos dentro de la misma red local, accediendo cada uno correctamente a la aplicación web a través de Apache, interactuando con la interfaz de usuario y comprobando que las operaciones de subida y descarga de archivos se realizaron exitosamente. Se extendieron las pruebas mediante distintos tipos de archivos y fueron verificadas las rutas encontrándose correctamente gestionadas por el proxy inverso.

Este entorno virtual no solo permitió probar y validar los distintos componentes del sistema, sino que además ofreció una experiencia cercana a un entorno real de producción, facilitando la identificación y resolución de problemas antes de escalar el proyecto a un servidor dedicado.

Demostración práctica de la simulación

Comenzamos la simulación probando la gestión de rutas del servidor Apache2, deteniendo el servidor Flask e intentando acceder a él mediante la dirección IP de la máquina virtual. (figura 1)



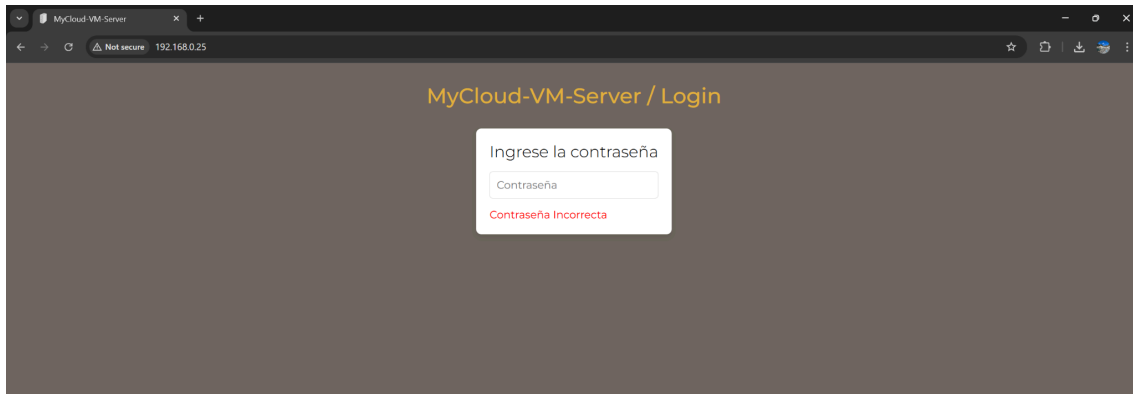
(figura 1)

Una vez comprobada la gestión de rutas, inicializamos el servidor, ejecutando el comando: `python app.py` en la ruta directorio donde se alberga el servidor. (figura 2)

```
^Cdanbone@danbone-VirtualBox: ~/Documents/UTH/Integrador_Ay50/mycloud-vm-server/backend/flask_servidor$ python3 app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 294-441-380
```

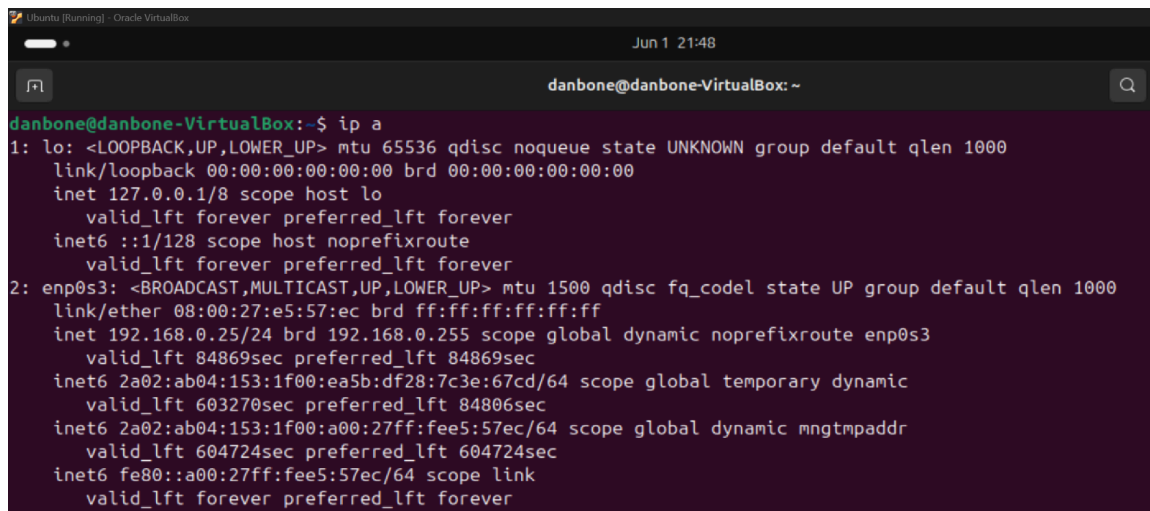
(figura 2)

Habiendo inicializado exitosamente el servidor, accedemos nuevamente a la IP de la máquina virtual en el navegador, donde esta vez, logramos cargar la interfaz de usuario correctamente. (figura 3)



(figura 3)

Ejecución del comando `ip a` en la terminal del servidor, con la finalidad de visualizar la dirección IP asignada a la VM. (figura 4)



```
danbone@danbone-VirtualBox:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:e5:57:ec brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.25/24 brd 192.168.0.255 scope global dynamic noprefixroute enp0s3
        valid_lft 84869sec preferred_lft 84869sec
    inet6 2a02:ab04:153:1f00:ea5b:df28:7c3e:67cd/64 scope global temporary dynamic
        valid_lft 603270sec preferred_lft 84806sec
    inet6 2a02:ab04:153:1f00:a00:27ff:fee5:57ec/64 scope global dynamic mngtmpaddr
        valid_lft 604724sec preferred_lft 604724sec
    inet6 fe80::a00:27ff:fee5:57ec/64 scope link
        valid_lft forever preferred_lft forever
```

(figura 4)

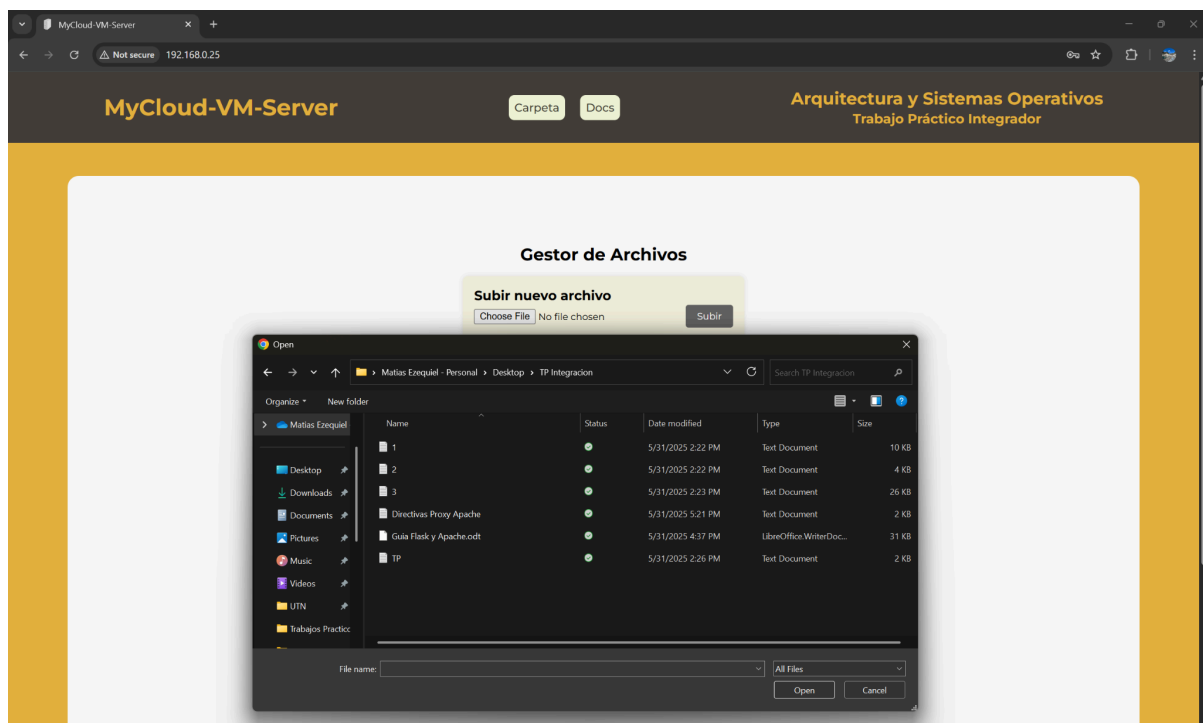
Como el gestor de archivos posee una contraseña como medida de seguridad, procedemos a dar su ingreso para acceder. Al momento del inicio de la prueba, el almacenamiento se encuentra vacío. (figura 5)



(figura 5)

A continuación se muestran las funcionalidades disponibles para el usuario mediante una rápida serie de pruebas de simulación de selección, carga, eliminación y descarga de archivos.

- **Selección de archivo a subir:** se genera un evento al hacer click en el botón “choose file” de la caja “subir un nuevo archivo” que una vez seleccionado en el explorador, la interfaz muestra el nombre del mismo. (figura 6 y 6.1)
- **Carga de archivo seleccionado:** se genera un evento click con el botón de “Subir”, que, como su nombre lo indica, carga el archivo en el almacenamiento, dando retroalimentación al usuario por medio de avisos dinámicos. (figura 7 y 7.1)
- **Eliminación de archivo:** una vez cargado un archivo al almacenamiento es listado, bajo el título de “archivos disponibles”. La misma lista agrega un botón a la izquierda de cada nombre de archivo que al generar el evento click sobre él, lo elimina del listado y por ende del almacenamiento global, no sin antes generar una ventana de confirmación, para prevenir eventos involuntarios. (figura 8 y 8.1)
- **Descarga de archivo almacenado:** haciendo uso de la lista “archivos disponibles”, cada nombre de archivo, vinculado a un *ancla* con su nombre, permite que al generar un evento de click sobre él, se dispare el método “GET” para descargar a la computadora del usuario dicho archivo. (figura 9)



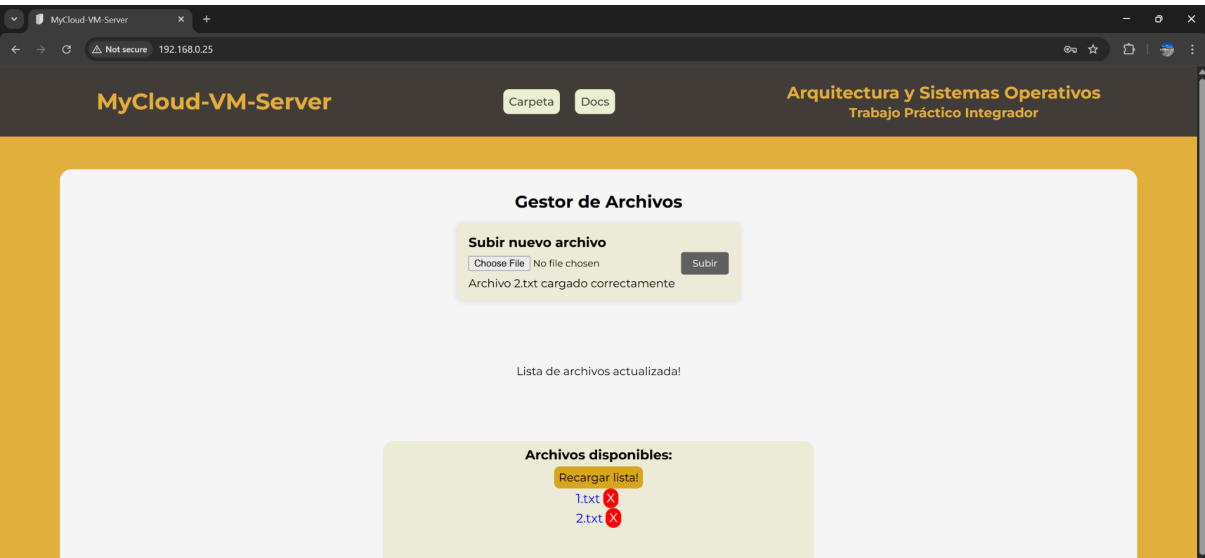
(figura 6)



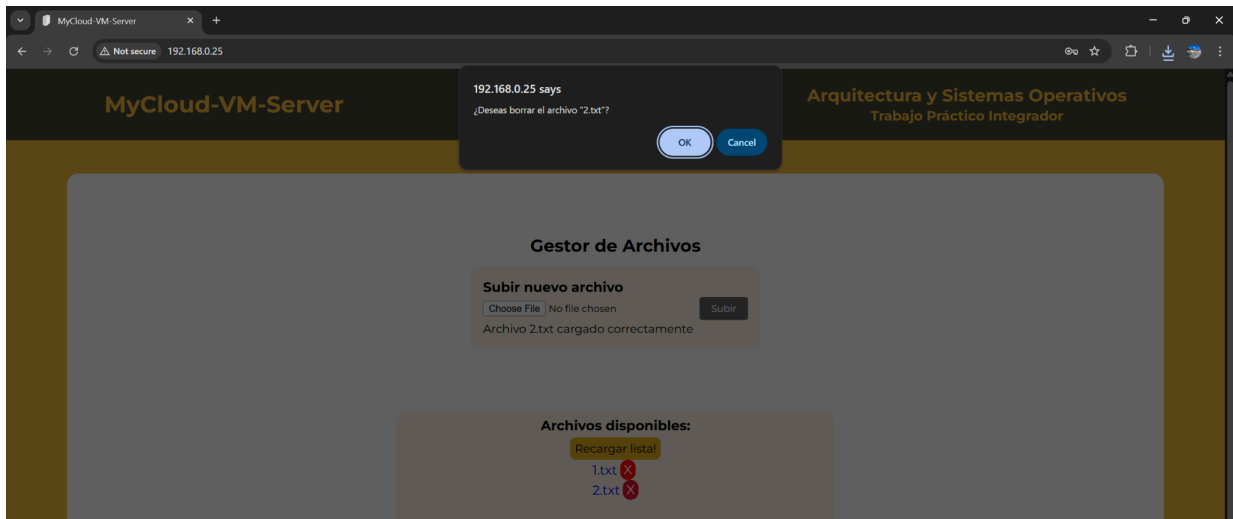
(figura 6.1)



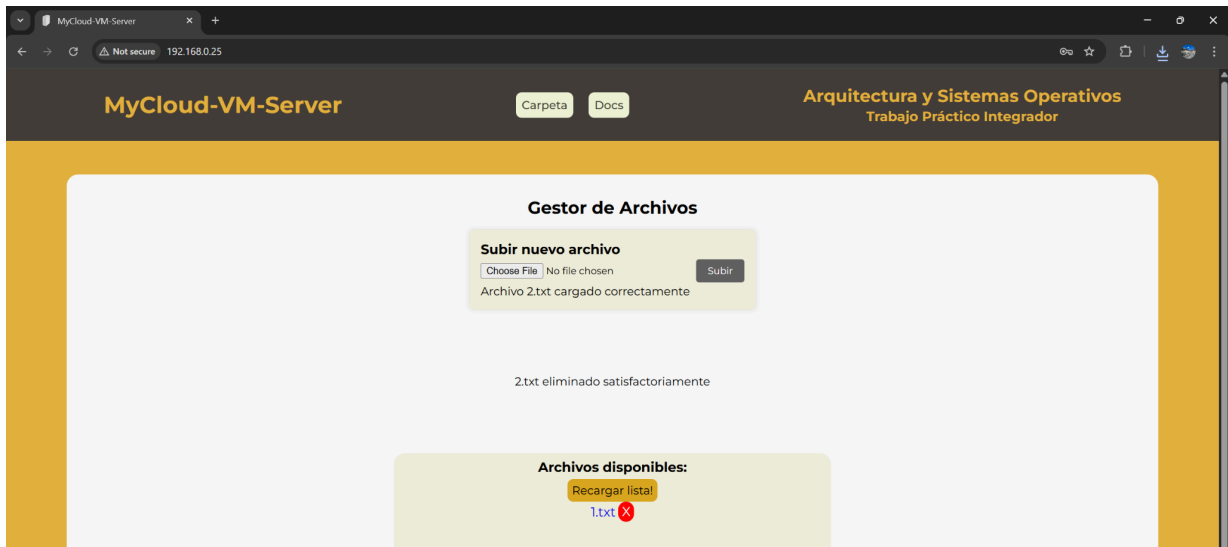
(figura 7)



(figura 7.1)



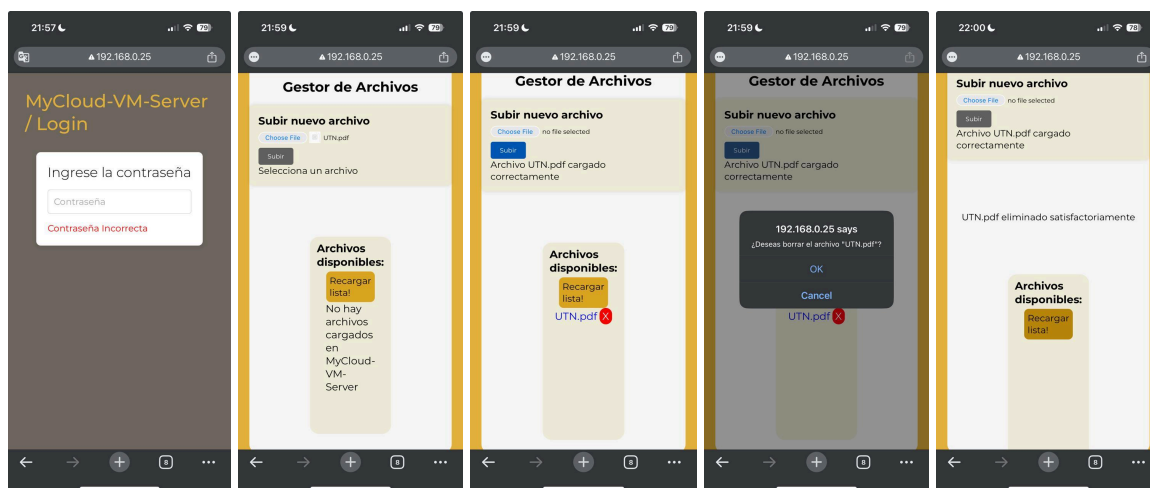
(figura 8)



(figura 8.1)



(figura 9)

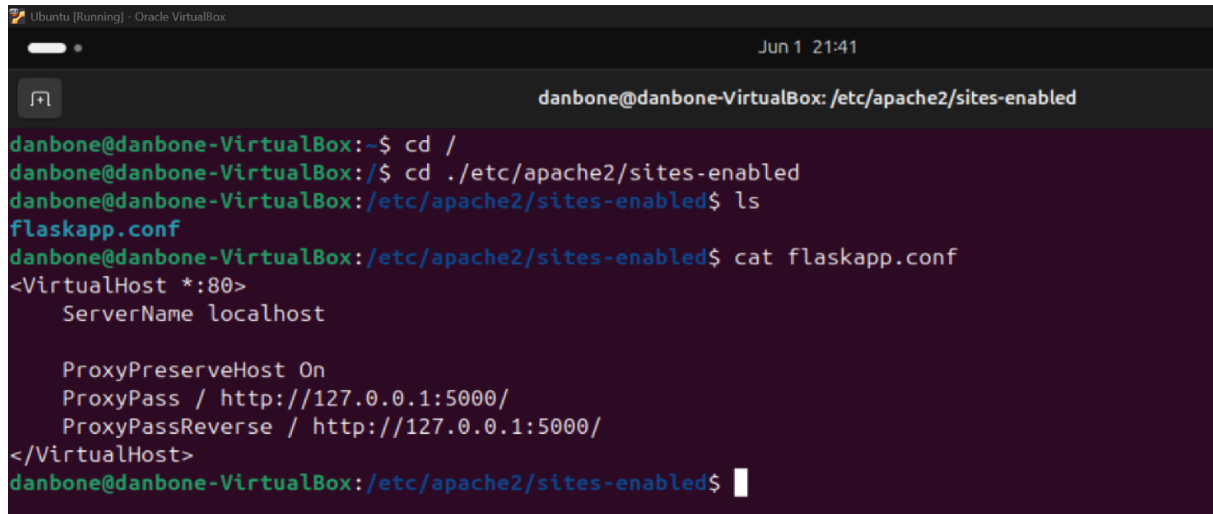


Las imágenes previas demuestran las mismas pruebas realizadas sobre un dispositivo con iOS, conectado a la misma red LAN a la cual la máquina host y máquina virtual están conectadas, en donde se corrobora el acceso, y ejecución de métodos HTTP.

Se adjunta esquema perteneciente al archivo de configuración que corresponde a Apache2 respecto al servidor Flask, en donde se identifican las siguientes configuraciones: (figura 10)

- <VirtualHost *:80> → Responde a solicitudes HTTP en el puerto 80.
- ServerName localhost → Indica a Apache a que dominio o IP aplica la configuración.
- ProxyPreserveHost On → Preserva el encabezado original de la solicitud cuando se envía al backend (Flask)
- ProxyPass / http://127.0.0.1:5000/ → Apache redirige las solicitudes al servidor Flask

- ProxyPassReverse / http://127.0.0.1:5000/ → Complementario a ProxyPass, permite recibir las respuestas del backend, permitiendo que redirecciones y URLs funcionen correctamente.



```

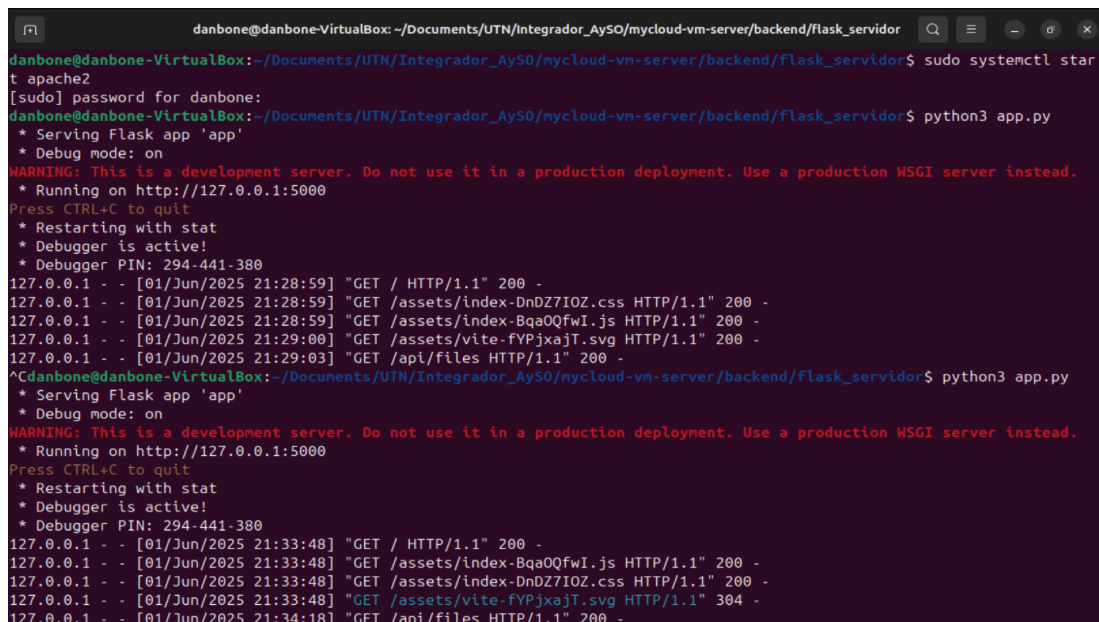
danbone@danbone-VirtualBox: /etc/apache2/sites-enabled
danbone@danbone-VirtualBox:~$ cd /
danbone@danbone-VirtualBox:/$ cd ./etc/apache2/sites-enabled
danbone@danbone-VirtualBox:/etc/apache2/sites-enabled$ ls
flaskapp.conf
danbone@danbone-VirtualBox:/etc/apache2/sites-enabled$ cat flaskapp.conf
<VirtualHost *:80>
    ServerName localhost

    ProxyPreserveHost On
    ProxyPass / http://127.0.0.1:5000/
    ProxyPassReverse / http://127.0.0.1:5000/
</VirtualHost>
danbone@danbone-VirtualBox:/etc/apache2/sites-enabled$

```

(figura 10)

Se adjunta una captura de la terminal Linux Ubuntu, donde se aprecian los logs del servidor, su inicialización y cuando un usuario hace uso del método GET y también el uso del comando `python app.py`. (figura 11)



```

danbone@danbone-VirtualBox: ~/Documents/UTN/Integrador_AySO/mycloud-vm-server/backend/flask_servidor
danbone@danbone-VirtualBox:~/Documents/UTN/Integrador_AySO/mycloud-vm-server/backend/flask_servidor$ sudo systemctl start apache2
[sudo] password for danbone:
danbone@danbone-VirtualBox:~/Documents/UTN/Integrador_AySO/mycloud-vm-server/backend/flask_servidor$ python3 app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 294-441-380
127.0.0.1 - - [01/Jun/2025 21:28:59] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Jun/2025 21:28:59] "GET /assets/index-DnDZ7IOZ.css HTTP/1.1" 200 -
127.0.0.1 - - [01/Jun/2025 21:28:59] "GET /assets/index-Bqa0QfwI.js HTTP/1.1" 200 -
127.0.0.1 - - [01/Jun/2025 21:29:00] "GET /assets/vite-fYPjxajT.svg HTTP/1.1" 200 -
127.0.0.1 - - [01/Jun/2025 21:29:03] "GET /api/files HTTP/1.1" 200 -
^Cdanbone@danbone-VirtualBox:~/Documents/UTN/Integrador_AySO/mycloud-vm-server/backend/flask_servidor$ python3 app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 294-441-380
127.0.0.1 - - [01/Jun/2025 21:33:48] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Jun/2025 21:33:48] "GET /assets/index-Bqa0QfwI.js HTTP/1.1" 200 -
127.0.0.1 - - [01/Jun/2025 21:33:48] "GET /assets/index-DnDZ7IOZ.css HTTP/1.1" 200 -
127.0.0.1 - - [01/Jun/2025 21:33:48] "GET /assets/vite-fYPjxajT.svg HTTP/1.1" 304 -
127.0.0.1 - - [01/Jun/2025 21:34:18] "GET /api/files HTTP/1.1" 200 -

```

(figura 11)

Metodología Utilizada: Marco teórico práctico

1. Investigación previa:

Recabado de información y estudio de documentación oficial respecto a [servidores HTTP Apache](#) y desarrollo de aplicaciones [Flask](#) necesarias para realizar las configuraciones de forma exitosa y ordenada. Uso de **IA generativa** para responder preguntas en base a documentación oficial y esquematizar ejemplos preliminares.

2. Diseño del proyecto:

Para lograr comunicar una interfaz de usuario con el servidor, llegamos a la conclusión de que, para nuestro caso y con la finalidad general con la Virtualización como eje central, lo mejor era usar las tecnologías antes mencionadas para usar al máximo las ventajas de la máquina virtual. Es algo totalmente inviable llevar adelante este proyecto en caso de usar como host del servidor, un sistema común y corriente que al mismo tiempo tenga la responsabilidad de servir para uso cotidiano con funcionalidades de ofimática, ¡Sería un desorden!.

Para alcanzar este objetivo, nos dividimos las tareas de la siguiente forma:

1. Configuración del entorno virtual con el código del lado servidor.
2. Configuración de la interfaz de usuario, con las funciones asíncronas dedicadas a la ejecución de los métodos de manejo de datos.

Matías Vazquez, fue el encargado de desarrollar el entorno virtual y configurar el servidor apache, así como también gestionar el correcto desempeño de los métodos.

Patricio Sussini, por otro lado, se encargó de diseñar desde cero una aplicación web, con las funcionalidades necesarias para hacer uso de los métodos y características del proyecto.

Con una dirección clara, la estructura del proyecto se construyó de la siguiente forma:

```
MyCloud-VM-Server/
|
├── backend/
|   ├── flask_server/
|       ├── __pycache__/      ← Archivos compilados de Python
|       ├── venv/             ← Entorno virtual de Python
|       └── app.py             ← Servidor Flask con endpoints de API
|
├── frontend/
|   ├── dist/                 ← Build de producción de React (Vite)
|   ├── node_modules/         ← Dependencias de NPM
|   ├── public/               ← Archivos estáticos públicos
|   ├── src/                  ← Código fuente React (componentes, etc.)
|   ├── .gitignore
|   ├── eslint.config.js
|   ├── index.html            ← HTML base donde React inyecta su contenido
|   ├── package-lock.json
|   ├── package.json
|   ├── README.md
|   └── vite.config.js        ← Configuración de Vite (localhost)
```

3. Preparación del entorno virtual:

Se realizó la instalación de VirtualBox en una máquina anfitriona con sistema operativo Windows 11, se descargó la imagen ISO y posteriormente la creación de la máquina virtual con los recursos necesarios: 15gb de disco virtual y 4gb de memoria ram e instalación del sistema operativo Ubuntu en la misma. La configuración de red de la máquina virtual se realizó en modo **bridge**, lo cual permite que la máquina virtual reciba una IP dentro del mismo rango que la red local.

4. Instalación del Software necesario:

- a. Apache 2.4 como servidor HTTP y reverse proxy.
- b. Python 3 con las librerías necesarias para Flask.
- c. React para construir la interfaz del usuario

5. Desarrollo de la interfaz y servidor:

Se creó una API REST sencilla con Flask que permite listar, subir, descargar y eliminar archivos. La interfaz se conecta a los endpoints de la API.

6. Pruebas:

Se realizaron varias pruebas a lo largo de desarrollo de ambos componentes, así mismo una vez se conectaron los endpoints y se realizó la correcta configuración del servidor Apache, se realizaron distintas pruebas ejecutadas desde el sistema operativo host y otros dispositivos conectados a la misma red, accediendo a la IP de la máquina virtual a través del navegador.

Dichas pruebas involucraron la ejecución y acceso a la página web y validación general de la interfaz de usuario, realización de HTTP requests via comandos ***curl*** sobre los métodos GET / POST / DELETE, validación sobre el acceso y correcto direccionamiento desde la máquina host y un dispositivo smartphone, y respectivas pruebas sobre la integración total de la solución.

7. Errores y soluciones:

A lo largo del desarrollo y diseño del proyecto encontramos un error, el cual estuvo relacionado con la ruta de conexión entre la interfaz y el servidor a la hora de configurar la `STATIC_FOLDER` (encargada de manejar las rutas principales de la app hacia el `index.html`) requiriendo una ruta definitiva y no parcial. Esto fue una dificultad para realizar las primeras pruebas. (figura 12)

Otro error también relacionado con las rutas, tuvo que ver con las redirecciones manejadas con el `index.html`, que requirió una función con un condicional para su correcta configuración. (figura 13)

```

STATIC_FOLDER = os.path.join(os.path.dirname(os.path.abspath(__file__)), '/home/danbone/Documents/UTN/Integrador_AySO/mycloud-vm-server/frontend/dist')

app = Flask(__name__, static_folder=STATIC_FOLDER, static_url_path="/")
#CORS(app)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

@app.route('/')
def home_redirect():
    return send_from_directory(app.static_folder, "index.html")

```

(figura 12)

```

@app.route('/', defaults={"path":""})
@app.route('/<path:path>')

def route_front(path):
    if path != '' and os.path.exists(os.path.join(app.static_folder, path)):
        return send_from_directory(app.static_folder, path)
    else:
        return send_from_directory(app.static_folder, "index.html")

```

(figura 13)

Herramientas utilizadas:

- IDE: Visual Studio Code
- Control de versiones: GIT y Repositorio GITHUB
- Librerías y Frameworks: Flask, ReactJS
- Servidor Web: Apache2
- Entorno: VirtualBox y Ubuntu 24.0

Resultados Obtenidos

A lo largo del desarrollo del caso práctico se alcanzaron los resultados esperados, tanto en la configuración del entorno, como en la implementación y ejecución de la aplicación, a continuación se detallarán punto por punto los distintos resultados en base a las distintas fases del proyecto.

- Funcionamiento correcto del sistema operativo en la máquina virtual.
- Servidor accesible en red local.
- Conexión entre la interfaz y el servidor.
- Casos de prueba realizados:
 - Acceso desde el sistema operativo host.
 - Carga de distintos tipos de archivos (PDF, imágenes, texto).
 - Descarga de archivos previamente subidos.
 - Prueba de la interfaz en distintos navegadores.
 - Verificación de rutas manejadas por Apache (comprobando, por ejemplo, que si Flask no está activo, Apache devuelve error 503).
- Errores corregidos
 - Configuración del archivo flaskapp.conf de Apache que impedían la correcta redirección al localhost. Esto se corrigió al revisar la documentación oficial.
 - Configuración del modo de red de la VM en **Bridge** (previamente configurado en modo **NAT**).

Conclusiones

Este proyecto permitió integrar el conocimiento de forma más práctica y concreta cómo interactúan los componentes dentro de una arquitectura cliente-servidor funcionando en una red local. Desde el navegador en un dispositivo cliente, pasando por el servidor web Apache, hasta llegar a la lógica del servidor en Flask y devolver una respuesta al usuario.

Como mejora futura, nos gustaría publicar el servidor en Internet, de forma que sea accesible desde redes externas más allá de la LAN local. Sin embargo, entendemos que este paso implica desafíos adicionales en términos de seguridad, como la gestión de puertos, protección contra ataques y el cifrado de las comunicaciones, por lo que requerirá preparación adicional.

Durante el desarrollo, surgieron algunas dificultades relacionadas con la configuración de red de la máquina virtual y el comportamiento del proxy inverso. Estos problemas fueron resueltos a partir de la revisión de documentación oficial, extensas pruebas locales, corrección de errores y ajustes en la configuración del servidor Apache.

Todo el proceso fue hecho teniendo fuertemente en consideración el uso ético, responsable y didáctico de la **IA generativa**, haciendo su uso para generar conocimiento, **no** para producción, lo cual se demuestra en el esfuerzo, dedicación y aprendizaje en el proyecto.

Estamos altamente conformes con el proyecto en términos de calidad, estética y experiencia ganada. Consideramos que fue un desafío que propuso una meta elevada que logramos alcanzar con éxito y reconocemos el aprendizaje en cada prueba y en cada solución.

En resumen, este proyecto no solo permitió aplicar conocimientos técnicos, sino también desarrollar habilidades clave en resolución de problemas dentro de un entorno realista. Haber logrado una integración funcional entre cliente y servidor representa un hito en nuestra formación. Nos motiva continuar perfeccionando este sistema para producción y asumir los nuevos desafíos que implica su publicación en Internet.

Bibliografía

- Documentación de VirtualBox: <https://www.virtualbox.org/manual/>
 - Configuración inicial: https://www.virtualbox.org/manual/topics/Introduction.html#configuring_virtual_box
- Ubuntu Desktop ISO: <https://ubuntu.com/desktop>
- Documentación Flask: <https://flask.palletsprojects.com/en/stable/>
 - Instalación, creación de apps y manejo de rutas: <https://flask.palletsprojects.com/en/latest/quickstart/>
 - Manejo de archivos (upload/download): <https://flask.palletsprojects.com/en/latest/patterns/fileuploads/>
- Documentación Apache (Reverse Proxy): https://httpd.apache.org/docs/2.4/howto/reverse_proxy.html
 - Documentación Apache (Directiva ProxyPass): https://httpd.apache.org/docs/current/mod/mod_proxy.html#proxypass
 - Documentación Apache (Directiva ProxyPassReverse): https://httpd.apache.org/docs/current/mod/mod_proxy.html#proxypassreverse
 - Documentación Apache (Directiva ProxyPreserveHost): https://httpd.apache.org/docs/current/mod/mod_proxy.html#proxypreservehost
- Documentación React :
 - <https://react.dev/learn>

Anexos

Link al repositorio de la aplicación web

<https://github.com/sussiniguanziroli/mycloud-vm-server.git>