# Trabajo Integrador

# Arquitectura y Sistemas Operativos

## Datos Generales

Título del trabajo: Virtualización

Alumnos:

- Grosso Belen (belugrosso98@gmail.com) COM-15
- Rocío Moyano (rocio.moyano434@gmail.com) COM-18

Materia: Arquitectura y Sistemas Operativos

Profesores: Martín Aristiaran,Mauricio Gabriel Pasti

Tutor: Luis Miguel Tola

Fecha de Entrega: 05/06/2025

---

## Índice



---

## Introducción

La elección del tema para el trabajo integrador de esta materia estuvo contemplada por varios motivos. El primero es que consideramos que la **virtualización** es un concepto muy importante dentro del ámbito de la tecnología por su eficiencia y capacidad para gestionar recursos; el segundo motivo estuvo guiado por la experiencia laboral y uso de esta herramienta, ya que consideramos que la virtualización tiene una importancia significativa como técnico en programación, permitiendo obtener una mejoría en el uso de hardware, flexibilidad en el desarrollo de software y adaptabilidad, lo que podemos concluir que abarca los dos componentes fundamentales de cualquier sistema informático.

Para finalizar esta introducción, los objetivos que queremos alcanzar con el desarrollo de este trabajo son:

- Una comprensión amplia de los conceptos fundamentales de la virtualización.
- El entendimiento de la importancia del tema mencionado dentro del área tecnológica.
- Su uso práctico en el ámbito profesional.
- Los resultados obtenidos gracias al análisis del caso práctico.
- Conclusión final.

---

## Marco Teórico

### Introducción histórica y concepto general:

La **virtualización** fue originada en la década de 1960 por IBM, quienes implementaron el concepto de Logical Partition, es decir la habilidad de dividir el hardware del servidor y que esas partes actúen de forma independiente dentro del mismo servidor, pero no fue hasta 1990 que VMWARE perfeccionó esta herramienta para arquitecturas x86 logrando un aumento significativo en el rendimiento de la CPU.

En la actualidad, es uno de los conceptos más importantes en la tecnología, gracias a su optimización de recursos y agilización de procesos. Es una tecnología que permite la creación de versiones virtuales de recursos físicos, como servidores, almacenamiento, redes o sistemas operativos, permitiendo ejecutar múltiples sistemas operativos y aplicaciones en un solo servidor físico, aumentando la eficiencia de los recursos tecnológicos y reduciendo los costos operativos.

## Beneficios de la virtualización

En este concepto, se destacan ciertos beneficios en su uso:

- Eficiencia
- Flexibilidad
- Ahorro de costos

La eficiencia es una parte fundamental, ya que permite un uso óptimo de recursos para lograr mayor cantidad de resultados generando procesos más ágiles y simples mejorando la infraestructura IT dentro de una empresa u organización. A su vez, la virtualización permite que varias máquinas virtuales (VMs) corran sobre un solo servidor físico, aprovechando al máximo la capacidad de procesamiento, memoria y almacenamiento. También, permite una mayor escalabilidad porque es más fácil aumentar o reducir recursos asignados a una máquina virtual sin tener que cambiar hardware.

La flexibilidad, según el sitio web oficial IBM[1], se destaca por reemplazar las computadoras físicas con máquinas virtuales definidas por software y facilita el uso y la gestión de políticas escritas en software. Esto le permite crear flujos de trabajo de gestión de servicios de IT automatizados.

El uso de la virtualización genera seguridad al ocurrir problemas técnicos, ya que se pueden crear copias o imágenes de los sistemas incluso antes de hacer las actualizaciones correspondientes para que el tiempo de recuperación sea mucho

[1] https://www.ibm.com/mx-es/topics/virtualization

más rápido. A su vez hace más fácil la administración y posibilita una gestión centralizada al utilizar servidores virtuales para instalar programas, mantener la seguridad, realizar updates, etc, lo que hace más fácil gestionar desde cualquier lugar.

El ahorro de costos es un punto importante para las empresas porque al permitir ejecutar varias máquinas virtuales (VMs) en un solo servidor físico, se minimiza significativamente la necesidad de comprar múltiples equipos, como también una reducción del consumo de energía y electricidad, ahorro en mantenimiento y menos servidores físicos para centros de datos, lo que puede traducirse en menores gastos de infraestructura.

**Limitaciones y desventajas**

En contraparte, vale mencionar que la virtualización también conlleva algunas desventajas de las cuáles, si bien mencionamos el ahorro de costos como una ventaja, también podemos presentarla como un problema para ciertos casos, ya que esto se debe a que la virtualización implica costos iniciales que se requieren para la adquisición del software y el hardware en los que se basa el proceso. Las licencias y servidores que se deben adquirir suelen ser costosas al inicio de pasar a la virtualización para empresas más pequeñas. También cabe mencionar que la máquina virtual tiene una dependencia del host donde se instale, lo que puede provocar una baja de rendimiento en comparación al equipo o servidor en el que se esté usando. Por último, se requiere de un conocimiento elevado para su instalación en ciertos entornos, lo que puede conllevar más tiempo y esfuerzo.

Previamente, mencionamos varias veces la máquina virtual o VM, pero ¿a qué nos referimos con esto?

## Conceptos clave: Máquina Virtual e Hipervisores

Una máquina virtual (VM) es un entorno informático que funciona como un sistema aislado con su propia CPU, memoria, interfaz de red y almacenamiento, el cual se crea a partir de un conjunto de recursos de hardware.

¿Pero cómo las creamos? Gracias a los hipervisores, que son un software que nos permite gestionar estas máquinas virtuales actuando como intermediario entre el hardware y los sistemas operativos, permitiendo que cada VM tenga acceso a los recursos.

## Tipos de hipervisores

Existen dos tipos de hipervisores:

- **Hipervisor tipo 1: Nativo**

Se encuentran en equipos con servidores dedicados (bare metal). El hipervisor programa los recursos de la máquina virtual directamente en el sistema de hardware. La máquina virtual basada en el kernel (KVM) es un ejemplo de hipervisores de tipo 1.

¿Cómo funciona? Al encender el sistema, el hypervisor de tipo 1 se carga directamente desde el firmware (como BIOS o UEFI), sin que un sistema operativo convencional intervenga. Desde el primer momento, el hipervisor toma control completo del hardware.

El hypervisor gestiona directamente todos los componentes físicos del sistema, distribuyéndolos entre las máquinas virtuales (VMs) de forma controlada:

1. CPU: Asigna núcleos virtuales (vCPU) a cada VM, distribuyendo los ciclos de procesamiento según la demanda.
2. Memoria RAM: Otorga bloques de memoria física a cada VM, garantizando el aislamiento para que ninguna pueda acceder a la memoria de otra.
3. Almacenamiento: Interactúa con los discos físicos mediante controladores y ofrece discos virtuales a las VMs.

4. Red: Configura adaptadores de red virtuales, simulando interfaces de red para cada entorno virtualizado.

Cada VM se ejecuta dentro de un entorno aislado que simula tener acceso exclusivo al hardware. El hipervisor proporciona dispositivos virtuales (discos, interfaces de red, tarjetas gráficas, etc.) que imitan el funcionamiento del hardware real.

El hipervisor incluye controladores que se comunican directamente con dispositivos físicos como la CPU, GPU o el almacenamiento. Esta comunicación sin intermediarios elimina la necesidad de un sistema operativo anfitrión, lo que reduce la latencia y mejora significativamente el rendimiento general.

Según la página IBM sobre los hipervisores[2], estos proporcionan la base para las prácticas modernas de virtualización y son cruciales para los centros de datos, el cloud computing y los entornos de escritorio.

- **Hipervisor tipo 2: Alojado**

Los recursos de la máquina virtual se programan en un sistema operativo host, que después se ejecuta en el sistema de hardware. VMware Workstation y Oracle VirtualBox son ejemplos de hipervisores de tipo 2.

Al iniciar el hardware, primero se carga un sistema operativo tradicional (como Windows, Linux o macOS). El hipervisor tipo 2 se ejecuta posteriormente como una aplicación dentro de este sistema operativo, lo que implica una dependencia directa de su entorno.

A diferencia de un hypervisor tipo 1, este no se comunica directamente con el hardware, en su lugar, realiza solicitudes a través del sistema operativo anfitrión.

Toda la administración de recursos (CPU, RAM, almacenamiento, red) está mediada por el sistema operativo anfitrión. Esto limita la capacidad del hipervisor para optimizar el uso de hardware, ya que debe compartir los recursos con el propio sistema operativo.

[2] https://www.ibm.com/es-es/think/topics/hypervisors

Como resultado, parte del rendimiento disponible se consume en mantener el sistema anfitrión, lo que reduce los recursos asignables a las máquinas virtuales.

**Virtualización en CPU, memoria, almacenamiento, red y dispositivos**

Ahora bien, si utilizamos la virtualización en contexto de hipervisor tipo 2 dentro de la CPU encontraremos que se utiliza la traducción binaria (binary translation), una técnica de virtualización que intercepta y traduce dinámicamente instrucciones privilegiadas del sistema operativo invitado en instrucciones seguras para el hardware del host.

En la memoria, permite que cada máquina virtual tenga la ilusión de disponer de su propia memoria física, aislada y continua, aunque en realidad esté compartiendo la memoria física del sistema con otras VMs y el host.

Con respecto al almacenamiento en entornos virtualizados, se abstrae y gestiona el almacenamiento físico como si fuera un recurso lógico, único y flexible. Permite que las máquinas virtuales accedan a recursos de almacenamiento sin saber su ubicación física real ni cómo están organizados.

En las redes, la virtualización de la red es la técnica que permite crear una red lógica y abstracta sobre la infraestructura física de red existente, que funcionan como si fueran redes físicas separadas, pero compartiendo el mismo hardware de red subyacente.

La virtualidad en los dispositivos, permite que una máquina virtual acceda y utilice dispositivos hardware físicos del sistema host (como tarjetas de red, discos, GPU, USB, etc.) a través de una capa de abstracción que simula o comparte estos dispositivos con las VMs.

**Virtualización vs Contenedores: Caso Docker**

Si bien existen muchas tecnologías que se relacionan con la virtualización, como por ejemplo Podman, containerd, Citrix Virtual Apps and Desktops, VMware Horizon, vamos a hablar de una en particular: **Docker**.

Ambas tecnologías permiten crear entornos aislados para el uso de aplicaciones, pero Docker utiliza el término “contenedores”, refiriéndose a una unidad ligera, portable y autosuficiente que se utiliza para ejecutar una aplicación con todas sus dependencias, configuraciones y archivos necesarios, aislada del resto del sistema.

Su arquitectura está compuesta por:

Imágenes: una imagen Docker es una plantilla inmutable que contiene todo lo necesario para ejecutar una aplicación, incluyendo sistema base, librerías y dependencias, archivos de configuración y código de la aplicación. Al ejecutar una imagen en Docker, este la convierte en contenedor añadiendo una capa superior de lectura-escritura donde se almacenan los cambios temporales.

Docker Daemon (dockerd): servicio en segundo plano que se ejecuta en host, administrando contenedores, imágenes y redes manejando las peticiones del cliente.

Docker CLI (Command Line Interface): es una herramienta de línea de comandos que permite interactuar, ejecutar contenedores, construir imágenes y administrar recursos en Docker.

Docker Registry: se trata de un almacén de imágenes que pueden ser distribuidas en contenedores.

Dockerfile: Específica instrucciones para la construcción de una imagen.

Redes: Se gestionan redes de distintos tipo según el sitio oficial Docker:

- Bridge[3]: es una capa de enlace que permite el avance de tráfico de datos entre segmentos de red.
- Host[4]: el contenedor no está aislado del host del Docker y no generará su propia dirección IP. Permite una mayor optimización y

[3] https://docs.docker.com/engine/network/drivers/bridge/

[4] https://docs.docker.com/engine/network/drivers/host/

es recomendable en casos donde el host debe manejar una cantidad considerable de puertos.

- Overlay[5]: conecta varios entornos de Docker para crear una red virtual entre hosts.
- Macvlan[6]: controlador de red para asignar una dirección MAC a la interfaz de red virtual de cada contenedor, haciéndolo parecer una interfaz de red física.

Para concluir, en este marco teórico se analizó la importancia y necesidad de esta tecnología fundamental en la actualidad. Sus usos variados y herramientas internas conllevan a un amplio abanico de posibilidades dentro de un entorno profesional. Es por eso que consideramos importante la implementación de la virtualización porque transforma la manera en que se utilizan y gestionan los recursos tecnológicos, aportando beneficios clave tanto a nivel técnico como económico.

A continuación, se desarrollará el caso práctico que canalizará el concepto de virtualización implementado dentro de la plataforma Docker, donde pondremos en contexto su utilización.

---

[5] https://docs.docker.com/engine/storage/drivers/overlayfs-driver/

[6] https://docs.docker.com/engine/network/drivers/macvlan/

**Caso Práctico**

Este se basa en una sencilla **aplicación to-do list**, donde podemos incorporar tareas nuevas, recordatorios, eliminarlos y marcarlos como completados.

Además analizaremos algunos conceptos que ayudarán a entender cómo es que funciona la aplicación y que se necesita.

¿Para qué sirven los archivos Dockerfile y docker-compose.yml?

Para dockerizar una aplicación (es decir, prepararla para que se ejecute en contenedores), utilizamos dos archivos clave:

- Dockerfile
- docker-compose.yml
- Ambos cumplen funciones diferentes pero se complementan entre sí.

¿Qué es el Dockerfile?

El Dockerfile es una especie de receta o guía que le indica a Docker cómo construir una imagen. Una imagen de Docker es una especie de "fotografía" que contiene todo lo necesario para ejecutar una aplicación: sistema base, dependencias, código, configuración, etc.

El Dockerfile suele contener instrucciones como:

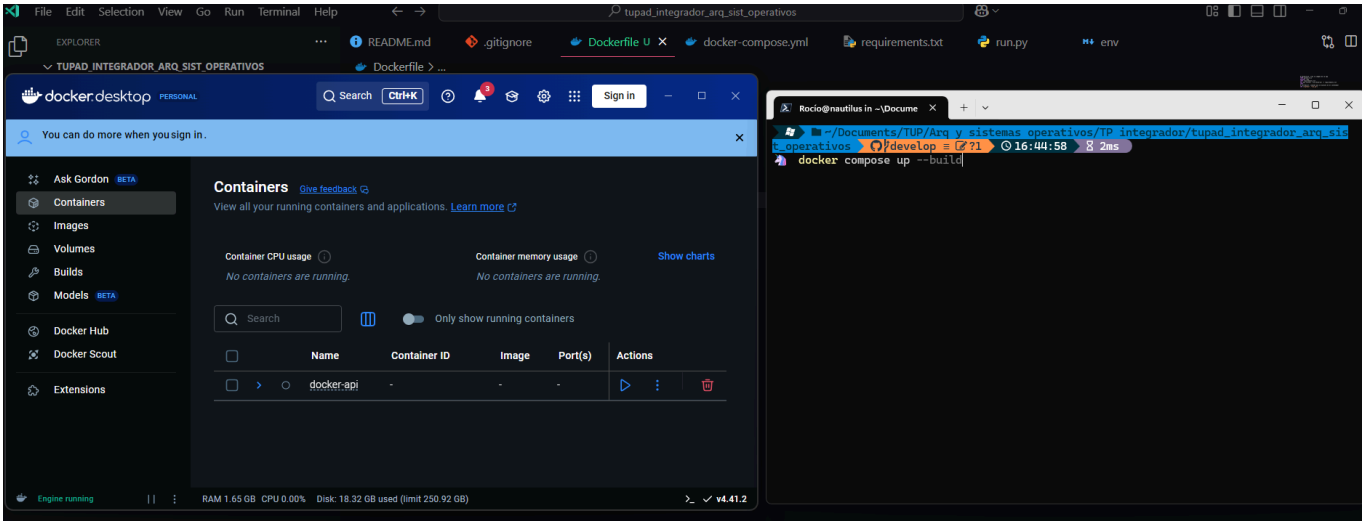
- Qué imagen base usar (por ejemplo, una imagen de Node.js, Python, etc.)
- Cómo instalar las dependencias del proyecto
- Copiar el código fuente al contenedor
- Qué comando ejecutar al iniciar el contenedor (por ejemplo, npm start, python app.py, etc.)

¿Qué es una imagen de Docker?

Una imagen es como una instantánea de una app y su entorno.

Contiene:

- El sistema base (Linux, Node, Python, etc.)
- Código fuente de la app
- Librerías y configuraciones necesarias




¿Qué es docker-compose.yml?

- El archivo docker-compose.yml se utiliza para orquestar varios contenedores como si fueran una sola aplicación. Es ideal cuando tu aplicación necesita más de un servicio para funcionar, como una base de datos, una API, un backend, etc.
- Por ejemplo, puedes definir:
    - Un contenedor para la aplicación
    - Otro contenedor para la base de datos
    - Configuraciones para que los datos persistan incluso si los contenedores se apagan

¿Qué se configura con docker-compose.yml?

Este archivo te permite:

- Definir múltiples servicios (por ejemplo, app, db, redis, etc.)
- Crear redes internas para que los contenedores se comuniquen entre sí
- Establecer variables de entorno, puertos expuestos y volúmenes para persistencia de datos
- Controlar el orden de arranque de los servicios (por ejemplo, que la base de datos arranque antes que la aplicación)

Flujo de trabajo típico con Docker

1. Crear un Dockerfile para definir cómo construir la imagen de tu aplicación

Dockerfile > ...

```
# Dockerfile, crea la imagen de la app
FROM python:3.11
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
# Entrypoint: comando que se ejecuta en el contenedor
CMD ["python", "run.py"]
```

2. Escribir un archivo docker-compose.yml para definir y configurar los servicios necesarios

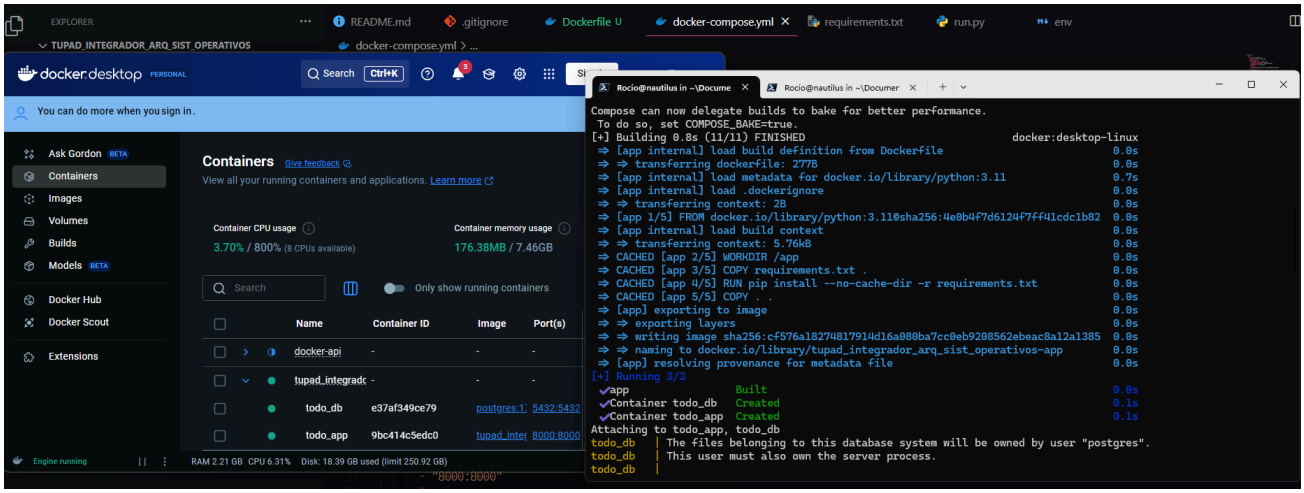
docker-compose.yml > ...

```
services:
  db:
    image: postgres:17
    container_name: todo_db
    environment:
      POSTGRES_USER: todo_user
      POSTGRES_PASSWORD: yourpassword
      POSTGRES_DB: todo_db
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U todo_user -d todo_db"]
      interval: 5s
      timeout: 5s
      retries: 5
  app:
    build: .
    container_name: todo_app
    depends_on:
      db:
        condition: service_healthy
    environment:
      DATABASE_URL: postgresql://todo_user:yourpassword@db:5432/todo_db
    ports:
      - "8000:8000"
    volumes:
      - .:/app
    command: bash -c "sleep 5 && python run.py"
volumes:
  postgres_data:
```

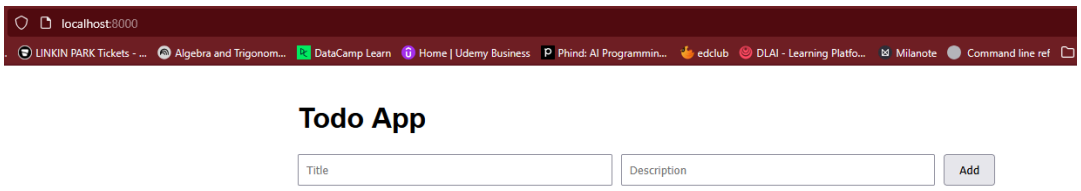
3. Ejecutar el comando:

```
docker-compose up --build
```

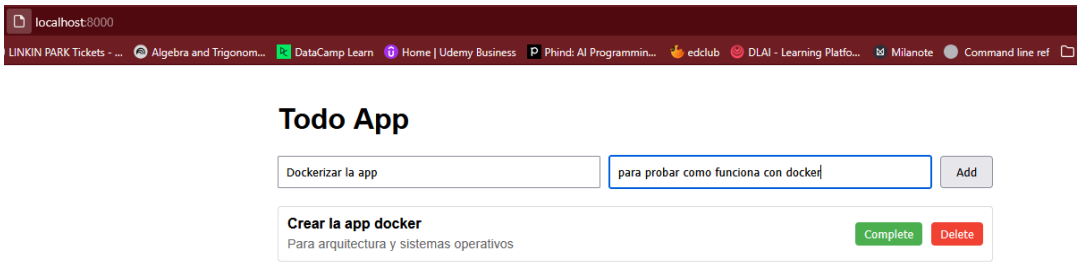
4. Esto construye la imagen y levanta todos los contenedores definidos en docker-compose.yml



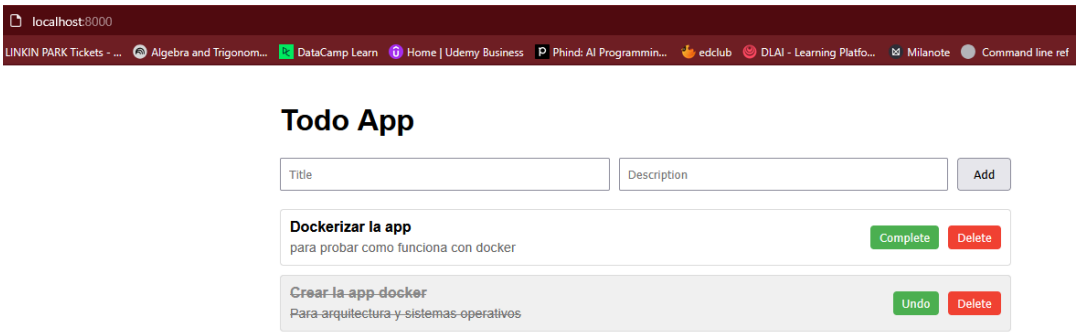

Al finalizar podremos ver nuestra página web localizada en localhost800.




Se pueden ver las tareas pendientes y podemos agregar nuevas o eliminarlas.



Y finalmente, podemos visualizar la tarea completada y otra que fue incluida.




Conclusión del ejercicio

La finalidad de Docker es facilitar la creación, despliegue y ejecución de aplicaciones utilizando contenedores. En términos simples, Docker permite empacar una aplicación junto con todas sus dependencias (bibliotecas, configuraciones, herramientas, etc.) en un solo paquete estandarizado llamado contenedor.

- Dockerfile: define cómo construir un contenedor
- docker-compose.yml: define cómo ejecutar varios contenedores juntos
- Juntos permiten crear apps portables, consistentes y fáciles de desplegar

---

## Metodología Utilizada

Para llevar a cabo el presente proyecto integrador sobre virtualización y práctica con Docker, se adoptó una metodología aplicada y experimental, con enfoque práctico-descriptivo. El objetivo fue adquirir experiencia directa en el uso de tecnologías de virtualización y contenedores, mediante el desarrollo y despliegue de servicios utilizando Docker.

Se utilizaron las siguientes herramientas:

- Conceptos clave sobre virtualización.

- Diferencias entre virtualización y contenedores.
- Arquitectura de Docker.
- Imágenes, contenedores, redes y volúmenes en Docker.
- Instalación de Docker Desktop.
- Creación de imágenes personalizadas con Dockerfile.
- Uso de docker-compose para levantar servicios interconectados.
- Gestión de volúmenes y persistencia de datos.
- El sistema fue documentado, probado y desplegado en el entorno local.

---

## Conclusión final

Con el paso del tiempo, la virtualización se ha consolidado como una de las herramientas fundamentales de la tecnología en los últimos años, proporcionando varios beneficios en su uso. No obstante, también presenta desafíos como los costos iniciales de implementación, la dependencia del hardware host y la necesidad de contar con personal especializado, pero esto no deja de lado la importancia de esta herramienta en el área IT. Es por eso que este presente trabajo desarrolló y analizó su aporte, considerando todos los conceptos claves disponibles concluyendo en un aprendizaje extenso del tema.

---

## Bibliografía


- https://www.hn.cl/blog/hablemos-de-virtualizacion/
- https://walternavarrete.com/historia-sobre-virtualizacion-de-sistemas/
- https://aws.amazon.com/es/what-is/virtualization/#:~:text=La%20virtualizaci%C3%B3n%20es%20un%20proceso,de%20procesamiento%20y%20el%20almacenamiento.
- https://www.ibm.com/mx-es/topics/virtualization
- https://www.axentio.com/ventajas-y-desventajas-de-las-maquinas-virtuales/#Desventajas_de_usar_maquinas_virtuales

- https://www.redhat.com/es/topics/virtualization/what-is-a-virtual-machine
- Apunte TUPaD AySO “Hipervisores tipo 1 y 2”
- Apunte TUPaD Ayso “Docker vs Virtualización”
- https://www.ibm.com/es-es/think/topics/hypervisors

---

**Anexos**

- Presentación PowerPoint: https://gamma.app/docs/Arquitectura-y-Sistemas-Operativos--9ofhno3itb700xs
- Video explicativo: https://www.youtube.com/watch?v=M2lx6l5Kdf0
- Carpeta virtual con archivos del programa Docker: https://github.com/rochimo2/tupad_integrador_arq_sist_operativos/tree/develop