UTN

TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN A DISTANCIA

TRABAJO INTEGRADOR

ARQUITECTURA Y SISTEMAS OPERATIVOS

COMISIÓN: 24

ALUMNOS:
FACUNDO ARRIETA
LUCIANO ANDRELO

DOCKER + VIRTUAL MACHINE

PROFESOR: OSVALDO FALABELLA
TUTOR: GUSTAVO STURTZ

✎ Datos Generales

- **Título del trabajo**: Docker + VM por Facundo Arrieta y Luciano Andrelo
- **Alumnos**: Facundo Arrieta/Correo: facundo.arrieta2506@gmail.com

  Luciano Andrelo/Correo: andreloluciano91@gmail.com
- **Materia**: Arquitectura y Sistemas Operativos
- **Profesor/a**: Osvaldo Falabella y Tutor Gustavo Sturtz
- **Fecha de Entrega**: 05/06/2025

📑 Índice

---

1. Introducción

En el presente trabajo, se abordará el estudio y la implementación de tecnologías de virtualización, específicamente **Docker y máquinas virtuales**. La elección de este tema radica en su creciente relevancia en la industria del desarrollo y la administración de sistemas.

La importancia de estas tecnologías en la formación como técnico en programación es

fundamental. En un panorama tecnológico donde las aplicaciones son cada vez más complejas y distribuidas, la capacidad de empaquetar, desplegar y gestionar software de manera aislada y reproducible se ha vuelto indispensable. Docker, al ofrecer un método ligero y portable para contener aplicaciones y sus dependencias, permite a los desarrolladores garantizar que el software funcione de manera idéntica en cualquier entorno, desde la máquina local hasta la nube. Por otro lado, las máquinas virtuales, si bien más pesadas, proporcionan un aislamiento completo del hardware subyacente, lo cual es crucial para escenarios que requieren entornos de sistema operativo específicos o una seguridad robusta. Dominar estas herramientas no solo mejora la productividad individual, sino que también facilita la colaboración en equipos de desarrollo y la implementación en infraestructuras modernas.

Con el desarrollo de este trabajo, se proponen los siguientes objetivos: comprender los fundamentos teóricos y las diferencias clave entre la virtualización basada en contenedores (Docker) y la virtualización de máquinas virtuales; adquirir experiencia práctica en la configuración y el despliegue de aplicaciones utilizando ambas tecnologías. Se busca consolidar los conocimientos adquiridos para aplicar estas herramientas de manera efectiva en futuros proyectos como técnicos en programación.

1.Marco Teórico

Virtualización: Conceptos y Beneficios.

Imagina tener muchos sistemas operativos a tu disposición para hacer con ellos lo que quieras, ya que si se comprometen, simplemente puedes crear más. Esto es virtualización. Gracias a esto, podrás tener, por ejemplo, cinco sistemas operativos, seis, o más...

Virtualizar es una acción que puede brindarnos muchas ventajas, tanto a nivel doméstico como empresarial. Hoy en día, consumir algún servicio que de cierta forma no esté empleando algún tipo de virtualización es bastante difícil.

¿Qué es la virtualización?

La virtualización es crear, a través de software, representaciones virtuales de servidores, almacenamiento, redes y otras máquinas físicas. Esta práctica permite a una computadora compartir sus recursos de hardware de manera eficiente con varios entornos separados de forma digital. Cada entorno virtualizado se ejecuta de manera aislada, dentro de los recursos asignados, como la memoria, el procesamiento y el almacenamiento y funcionan como si fueran máquinas independientes. Pueden crearse, copiarse y moverse fácilmente de un sistema físico a otro. Esto mejora la eficiencia y flexibilidad de los sistemas informáticos, facilita la gestión y el mantenimiento de los sistemas

---

¿Por qué es importante la virtualización?

Al aplicar esta tecnología, se logra interactuar con los recursos de hardware con mayor flexibilidad. En los entornos tradicionales, mantener múltiples servidores físicos implica altos costos, consumo eléctrico, espacio físico y mantenimiento constante.

La virtualización soluciona estas limitaciones al abstraer el hardware físico mediante software, permitiendo administrar y utilizar la infraestructura con mayor facilidad.

Por ejemplo, imaginemos una empresa que necesita tres servidores para funciones distintas: uno para correo, otro para una aplicación externa y otro para uso interno. En lugar de adquirir y mantener tres máquinas físicas, puedes crear tres máquinas virtuales en un único servidor físico, optimizando la distribución de recursos, reduciendo costos de infraestructura y facilitando la administración.

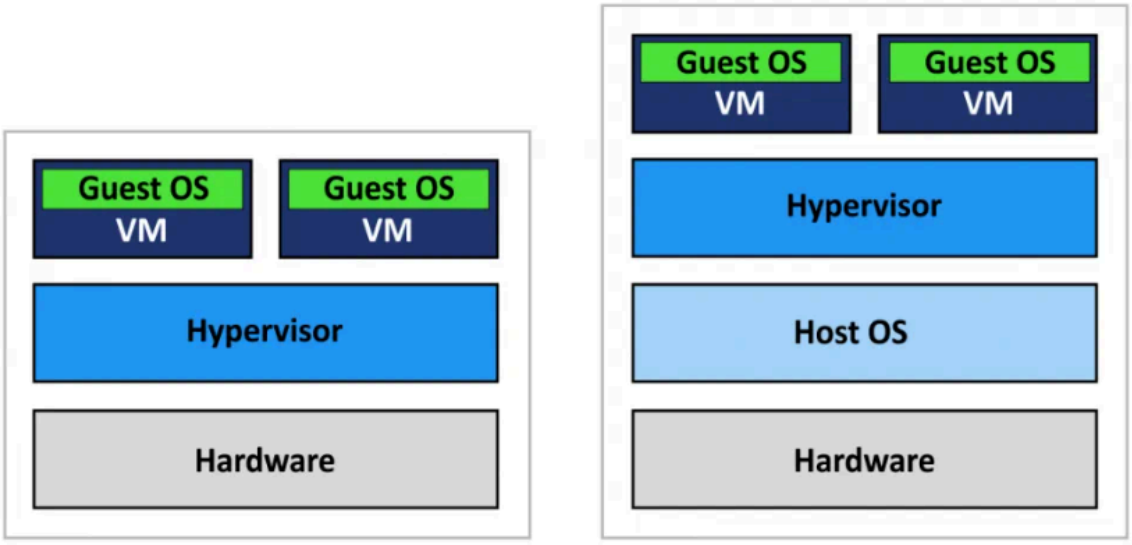
La virtualización se logra mediante el uso de software especializado conocido como **hipervisor**. Es el intermediario que existe entre el hardware físico y las máquinas virtuales, permitiendo la correcta asignación de recursos. Si queremos virtualizar un hardware físico, el encargado de convertirlo a digital y dividirlo es el hipervisor.

Hay dos tipos:

- **Tipo 1 (nativo - bare metal):** trabajan directamente sobre el hardware. sin depender de un sistema operativo anfitrión de intermediario. Aquí solo está el hardware, luego el hipervisor, y luego los recursos virtualizados, suelen utilizarse en entornos empresariales o servidores dedicados. Ejemplos: VMware ESXi, Citrix XenServer, Microsoft Hyper-V Server.

- **Tipo 2 (alojado):** Se ejecutan sobre un sistema operativo. Son ideales para uso doméstico o educativo, el sistema operativo anfitrión nos permite asignar memoria y capacidad de procesamiento a cada máquina virtual. Ejemplos: VirtualBox, VMware Workstation Pro, VMware Workstation Player.

En resumen, los de tipo 1 tienen mayor rendimiento y eficiencia de recursos, pero requiere configuración más compleja y hardware dedicado, y los de tipo 2 son más fáciles de usar para usuarios comunes, pero al estar el Sistema operativo de intermediario consume más recursos y depende del mismo.



fuente: nakivo.com

Otra función clave del hipervisor es **aislar** cada máquina virtual. Gracias a esto, una máquina virtual no puede acceder ni afectar directamente a otra máquina virtual en el mismo host.
Por ejemplo, puedes tener una máquina virtual con Windows y, en la misma computadora, otra máquina virtual con Linux ejecutándose sin que interfieran entre sí.

Además, los hipervisores permiten tomar **imágenes** de cada máquina virtual (es decir, copias completas). Esto significa que puedes, tomar tu máquina de Linux, hacer una imagen, descargar el archivo, ponerlo en un pendrive y transferirlo a otra computadora.

Tipos principales de virtualización

Existen distintos tipos de virtualización según el recurso que se desea utilizar, los más comunes son:

- **Virtualización de servidor:** permite ejecutar múltiples sistemas operativos en un mismo servidor físico, creando servidores virtuales independientes

- **Virtualización de escritorio**: permite a un usuario trabajar sobre una máquina sin tenerla físicamente, conectándose a un escritorio remoto desde cualquier dispositivo.

- **Virtualización de recursos hardware:** Simula componentes como memoria RAM, CPU, almacenamiento o tarjetas de red
    - CPU: Traduce instrucciones privilegiadas a seguras (traducción binaria) para que las Vms no accedan directamente al hardware físico.
    - Memoria: Utiliza mecanismos como tablas de páginas sombreadas para mapear direcciones virtuales a físicas, asegurando el aislamiento.

- **Virtualización de red**: divide una red física en varias redes virtuales, o unir varias físicas en una sola virtual, procesa y redirige paquetes de red.

- **Virtualización de aplicaciones:** posibilita ejecutar aplicaciones desde un servidor remoto, evitando instalarlas localmente en cada dispositivo.

- **Virtualización de almacenamiento:** Se logra mediante discos virtuales: archivos que actúan como discos duros para las VMs, permitiendo portabilidad y almacenamiento.

Desafíos Técnicos

Latencia: la capa del anfitrión introduce latencia adicional en la comunicación y las operaciones, comparado con la ejecución directa sobre el hardware, algunas operaciones pueden presentar un retraso.
Recursos: las VMs compiten por recursos con el sistema anfitrión.
Overhead: la traducción de instrucciones y la gestión de recursos pueden generar sobrecarga que afecta el rendimiento.

¿Qué es VirtualBox?

VirtualBox es un software de virtualización mantenido por Oracle. Es un hipervisor de tipo 2 muy popular, multiplataforma (Windows, Linux, macOS, Solaris), tiene múltiples idiomas, una interfaz gráfica simple, y ofrece muchas características.

Existen varias versiones:

- Oracle VM VirtualBox (gratuita)
- Oracle VM VirtualBox Enterprise (de pago, orientada a empresas, con soporte)
- VirtualBox Extension Pack (extensiones gratuitas para uso personal, que añaden funciones como soporte para USB 2.0/3.0 y cifrado de VMs).

Como hipervisor de tipo 2, necesita un sistema operativo para funcionar, ya que es una aplicación más del sistema.

Docker

¿Qué es Docker?

Docker es una plataforma de contenedores que permite empaquetar y ejecutar aplicaciones de forma aislada y portátil.

En este tipo de virtualización no hay un hipervisor creando máquinas virtuales. En su lugar, hay un motor de contenedores, que usa los recursos del sistema operativo para aislar los procesos de cada contenedor

¿por qué usar Docker? Permite empaquetar todo lo necesario para que la aplicación corra igual en cualquier ambiente: en tu ordenador, en un servidor o en la nube. Los

volúmenes permiten almacenar información relevante de un contenedor pero de forma externa. No más problemas de “en mi máquina funcionaba”

Características de Docker

- Portabilidad: los contenedores Docker se ejecutan en diferentes sistemas operativos sin modificaciones.
- Eficiencia: Docker utiliza recursos del host de manera eficiente, lo que reduce el consumo de recursos.

- Aislamiento: Aíslan las aplicaciones de otras, evitando conflictos.
- Escalabilidad: al permitir la creación y gestión de múltiples contenedores.

Arquitectura de Docker

Docker Daemon: El servicio principal que gestiona contenedores e imágenes.
Docker CLI: Interfaz de línea de comandos, para interactuar con Docker Daemon.
Docker Registry: repositorio centralizado para almacenar y compartir imágenes de Docker.
Contenedores e imágenes: un contenedor es una instancia en ejecución de una imagen, que define el entorno y software de la aplicación. El contenedor corre la aplicación

Redes en Docker:

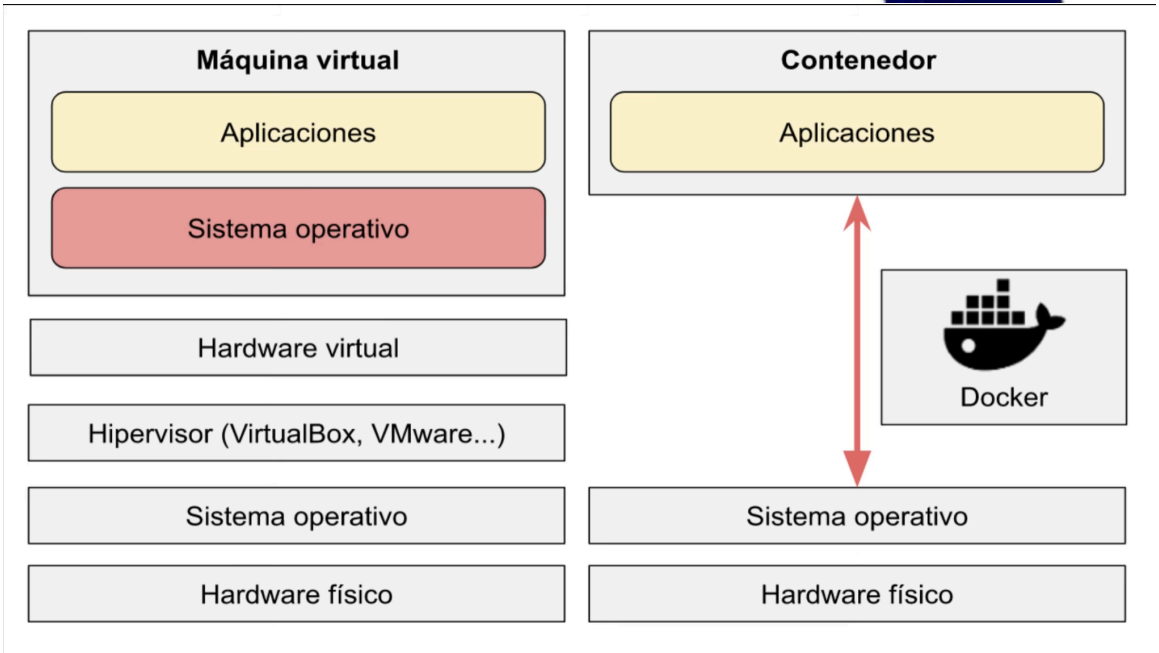
Bridge: red virtual interna que conecta contenedores dentro del mismo host.
Overlay: conexiones entre nodos en un entorno Docker multi-host.
Host: Los contenedores comparten la misma red que el host.
Maclvan: permite asignar una IP específica a un contenedor, como si fuera un dispositivo físico.

Docker vs Máquina Virtual

Docker: utiliza el kernel del sistema operativo host, lo que lo hace más ligero y rápido.
Máquina virtual: ejecuta un sistema operativo completo, dentro de una máquina virtualizada, provoca un mayor aislamiento, pero produce una mayor sobrecarga y ralentización.

Máquinas Virtuales	Contenedores (Docker)
Aislamiento completo, incluye kernel propio.	Usan el kernel del host, menos aislamiento.
Mayor consumo de recursos, más pesadas.	Más ligeros, rápidos y eficientes.
Flexibles para ejecutar cualquier SO.	Limitados al tipo de kernel del host.
Útiles para entornos completamente aislados.	Ideales para microservicios y despliegues rápidos.

fuente: Diferencias entre Maquinas Virtuales y Contenedores - Docker - YouTube



fuente: danielme.com

Casos de uso de Docker:

1. Microservicios: Docker es ideal para el desarrollo y despliegue de microservicios, ya que permite ejecutar cada servicio en un contenedor independiente.
2. Entornos de desarrollo: Docker facilita la creación de entornos de desarrollo consistentes para equipos de trabajo.
3. CI/CD: Docker se utiliza ampliamente en las prácticas de integración y de despliegue continuo, manejos de nodos y de apps, para que se desplieguen en la nube.

Usos de Docker y VM según el caso:

Docker: es una solución ágil y eficiente para el desarrollo, despliegue y escalado de

aplicaciones.
VM: proporcionan mayor aislamiento y son ideales para ejecutar entornos completos o aplicaciones que requieren un control total del sistema operativo.
Híbrida: Es posible combinar Docker y máquinas virtuales en arquitecturas híbridas para obtener los beneficios de ambos.

Imágenes, contenedores y volúmenes en Docker:

- Imágenes : plantillas/planos de código a ejecutar en un contenedor Docker.
- Contenedores : son instancias de imágenes de Docker, lo que significa que se encargan de correr el código. Se pueden crear diferentes contenedores a partir de la misma imagen, pero cada uno tendrá sus propios datos y estado.
- Volúmenes : el mecanismo para persistir datos fuera de los contenedores, completamente administrado por Docker utilizando un directorio dedicado en nuestra máquina host (entorno local, fuera de los contenedores). Un ejemplo puede ser una base de datos.

Sintaxis de Docker

La sintaxis básica de los comandos de Docker está conformada por el siguiente formato: docker <comando> [opciones] <argumentos].

Comandos comunes y su sintaxis:

➔ ***docker run:*** crea y ejecuta un contenedor. La sintaxis básica es: docker run [OPCIONES] IMAGE [CMD] [ARG].

➔ ***docker image:*** maneja imágenes de Docker.

docker image ls: Lista las imágenes.

docker image pull <imagen>: Descarga una imagen.

docker image build -t <nombre_imagen> .: Construye una imagen a partir de un Dockerfile.

docker image rm <imagen>: Elimina una imagen.

➔ ***docker container:*** sirve para el manejo de contenedores.

docker container ls: Lista los contenedores.

docker container run <imagen>: Crea y ejecuta un contenedor.

docker container rm <contenedor>: Elimina un contenedor.

docker container stop <contenedor>: Detiene un contenedor.

docker container start <contenedor>: Inicia un contenedor.

docker container exec -it <contenedor> /bin/bash: Ejecuta un comando en un contenedor en modo interactivo.

➔ ***docker network:*** sirve para manejar redes de Docker.

docker network ls: Lista las redes.

docker network create <nombre_red>: Crea una red.

docker network connect <red> <contenedor>: Conecta un contenedor a una red.

docker network disconnect <red> <contenedor>: Desconecta un contenedor de una red.

➔ ***docker volume:*** sirve para manejar volúmenes de Docker.

docker volume ls: Lista los volúmenes.

docker volume create <nombre_volumen>: Crea un volumen.

docker volume rm <volumen>: Elimina un volumen.

➔ ***docker-compose***: maneja aplicaciones multi-contenedor usando un archivo docker-compose.yml.

docker-compose up: Levanta los servicios descritos en el archivo docker-compose.yml.

docker-compose down: Detiene y elimina los servicios.

docker-compose logs: Muestra los logs de los servicios.

Dockerfile:

El Dockerfile es un archivo de texto que contiene instrucciones para construir una imagen de Docker. La sintaxis del Dockerfile se basa en un conjunto de instrucciones simples como FROM, RUN, COPY, EXPOSE, ENV.

Ngrok:

ngrok (se pronuncia "en-grok") es una herramienta muy popular y útil para desarrolladores que permite exponer un servidor local a internet a través de un túnel seguro.

¿Cómo funciona ngrok?

Ngrok funciona como un proxy inverso. Cuando ejecutas ngrok y le indicas el puerto de tu aplicación local, hace lo siguiente:

1. Establece una conexión segura y persistente (un "túnel") desde tu máquina local hacia uno de los servidores de ngrok en la nube.
2. Genera una URL pública única
3. Cuando alguien en internet accede a esa URL pública, la solicitud viaja a través del servidor de ngrok y luego, por el túnel seguro, se redirige a tu aplicación local en el puerto especificado.
4. La respuesta de tu aplicación local viaja de vuelta por el túnel, a través del servidor de ngrok, y llega al navegador del usuario en internet.

Sintaxis de Ngrok:

La sintaxis básica para ejecutar ngrok en la línea de comandos es: ngrok <protocol> <port>.

Explicación de los componentes:

- ***ngrok:*** Este es el comando que inicia el agente ngrok.
- ***<protocol>***: Este es el protocolo que ngrok usará para crear el túnel (por ejemplo, http, https, tcp).
- ***<port>***: Este es el puerto local al que ngrok enviará el tráfico a través del túnel.

Opciones adicionales:

Además de la sintaxis básica, ngrok ofrece varias opciones y parámetros que puedes usar para personalizar el comportamiento del túnel. Estas opciones se añaden al comando principal:

--region <region>: Especifica la región donde se crea el túnel.
--authtoken <authtoken>: Especifica el token de autenticación para el túnel.
--address <address>: Especifica una dirección local personalizada para el túnel.
--bind-tls o --tls: Activa el tráfico HTTPS.
--basic-auth <user:password>: Establece autenticación básica para el túnel.
--help: Muestra la ayuda con todas las opciones disponibles.

Fuentes de Docker y Ngrok:

Contenido teórico y práctico:

Fuentes otorgadas por la institución academica:

https://www.youtube.com/watch?v=_OAZZZeP4tE&feature=youtu.be

https://www.youtube.com/watch?v=_OAZZZeP4tE&feature=youtu.be
https://www.youtube.com/watch?v=Ehm2eMmlt3Q&feature=youtu.be
https://youtu.be/2rlEQiUEuEA
https://youtu.be/gWhN2XVJqAg
https://youtu.be/cVNQkQAAh84

Fuentes externas:

https://dionarodrigues.dev/blog/how-to-use-docker-images-containers-volumes-and-bind-mounts
1. Introducción, docker vs VMs, OCI, casos de uso... - Curso Docker gratuito y en español - YouTube
Necesitas APRENDER a VIRTUALIZAR AHORA | Curso COMPLETO de Virtualización con VirtualBox - YouTube
Máquinas Virtuales y Contenedores - ¿Qué son? ¿A qué huelen? ¿Para qué sirven?
¿Qué es la VIRTUALIZACIÓN? ¿Para Qué sirve la Virtualización? | Tipos de VIRTUALIZACIÓN DE SISTEMAS
¿Qué es la virtualización? - Explicación de la virtualización de la computación en la nube - AWS
Virtualización, máquinas virtuales y contenedores: ventajas y desventajas ¡Bien Explicado!
Cómo Funcionan las MÁQUINAS VIRTUALES? - YouTube
Diferencias entre Maquinas Virtuales y Contenedores - Docker - YouTube

Sintaxis y usos:

https://docker-curriculum.com/
https://imaginaformacion.com/tutoriales/comandos-docker

2.Caso Práctico

Caso Práctico

Breve descripción del problema a resolver:

El caso práctico abordado consistió en la necesidad de desarrollar y desplegar una aplicación web sencilla, encapsulada en un contenedor Docker, de manera que pudiera ser accesible no solamente de manera local en un entorno de desarrollo como Google Cloud Shell, sino también de forma remota desde una máquina virtual externa. La problemática específica incluyó superar los desafíos inherentes a la exposición de servicios que se ejecutan en entornos virtualizados y garantizar que la aplicación fuera funcional y visible a través de una conexión a internet desde un entorno completamente diferente,en este caso, la máquina virtual de un compañero. El objetivo fue simular un escenario de despliegue donde una aplicación desarrollada localmente pudiera ser compartida y utilizada en un contexto remoto, validando la efectividad de Docker para la portabilidad y Ngrok para la exposición pública.

Capturas de pantalla (Referencias):

Para ilustrar el proceso y la validación del funcionamiento, se adjuntan las siguientes capturas de pantalla:

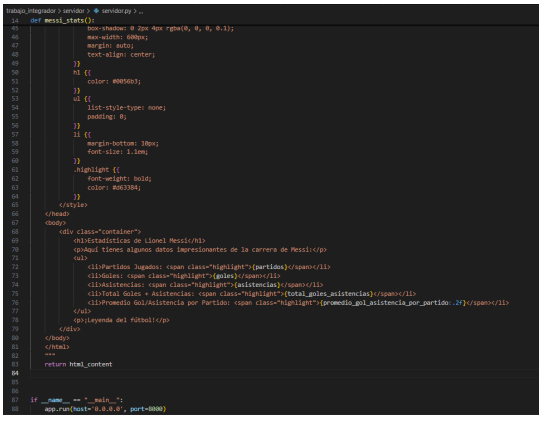
1. **docker-compose.png:** (Muestra la configuración del archivo docker-compose.yml con la definición de los servicios servidor y cliente, la red mi_red, y el mapeo del puerto 8000 del contenedor servidor al puerto 8000 del

trabajo_integrador > docker-compose.yml

```yaml
services:
  servidor:
    build: ./servidor
    container_name: servidor
    networks:
      - mi_red
    ports:
      - "8000:8000" # Mapea el puerto 5000 del contenedor al puerto 8000 del host

  cliente:
    build: ./cliente
    container_name: cliente
    networks:
      - mi_red
    depends_on:
      - servidor

networks:
  mi_red:
    driver: bridge
```

host).

2. **servidorpy-1.png y servidorpy-2.png:** (Muestran el código fuente de la aplicación Flask servidor.py, donde se define una ruta / para mostrar estadísticas y la aplicación se ejecuta en el puerto 8000).

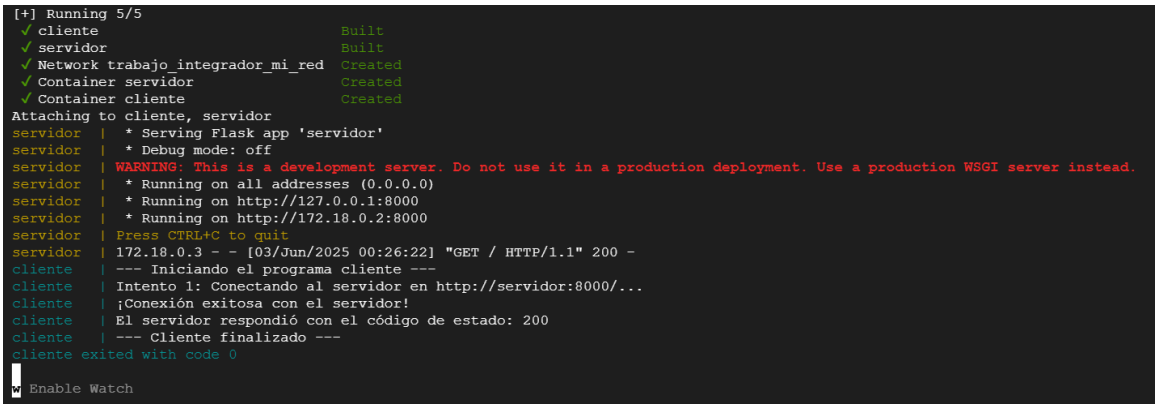


trabajo_integrador > servidor > servidor.py > messi_stats

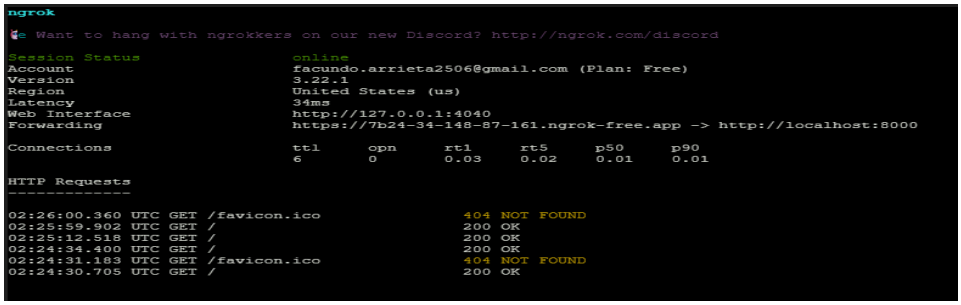
```python
from flask import Flask, jsonify

app = Flask(__name__)

# Datos de ejemplo de Messi
# Puedes ajustar estos números
MESSI_STATS = {
    "partidos_jugados": 1001,
    "goles": 804,
    "asistencias": 350
}

@app.route('/')
def messi_stats():
    partidos = MESSI_STATS["partidos_jugados"]
    goles = MESSI_STATS["goles"]
    asistencias = MESSI_STATS["asistencias"]

    total_goles_asistencias = goles + asistencias

    # Calcular promedio solo si hay partidos jugados para evitar división por cero
    if partidos > 0:
        promedio_gol_asistencia_por_partido = total_goles_asistencias / partidos
    else:
        promedio_gol_asistencia_por_partido = 0.0

    html_content = f"""
    <!DOCTYPE html>
    <html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Estadísticas de Lionel Messi</title>
        <style>
            body {{
                font-family: Arial, sans-serif;
                margin: 20px;
                background-color: #f4f4f4;
                color: #333;
            }}
            .container {{
                background-color: #fff;
                padding: 20px;
                border-radius: 8px;
                box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
```

3. (Muestra la salida del comando docker compose up --build, evidenciando la construcción de las imágenes, la creación de los contenedores y la ejecución de la aplicación Flask en el puerto 8000 dentro del contenedor, así como la

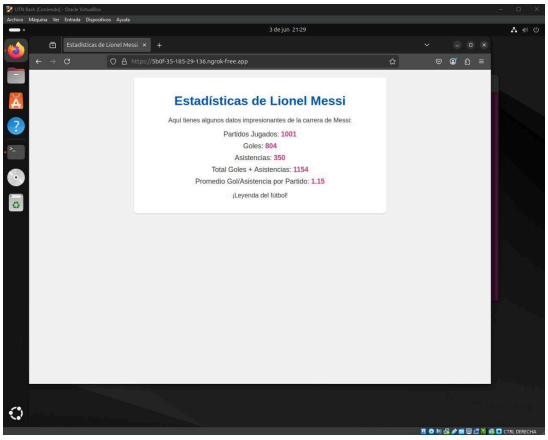
conexión exitosa del cliente al servidor internamente).



4. Muestra la salida del comando ./ngrok http 8000, indicando que se ha creado un túnel seguro desde una URL pública HTTPS hacia el puerto 8000 del localhost de Cloud Shell).



5. **Captura de pantalla, de la máquina virtual entrando a la página.**



Validación del funcionamiento:

La validación del funcionamiento se llevó a cabo mediante las siguientes acciones y resultados:

1. **Despliegue local con Docker Compose:** La ejecución de docker compose up --build en Google Cloud Shell permitió construir y ejecutar la aplicación Flask y el cliente dentro de contenedores Docker. Los logs confirmaron que la aplicación Flask se inició correctamente en el puerto 8000 dentro de su contenedor y que el cliente pudo comunicarse con ella a través de la red interna de Docker.

2. **Exposición del puerto con Docker Compose:** La configuración ports: - "8000:8000" en el docker-compose.yml aseguró que el puerto 8000 interno del contenedor servidor fuera accesible a través del puerto 8000 de la máquina virtual de Google Cloud Shell (el host). Esto se verificó implícitamente al poder establecer el túnel con Ngrok en este puerto.

3. **Creación del túnel público con Ngrok:** La ejecución de ./ngrok http 8000 creó un túnel seguro, asignando una URL pública HTTPS (https://<tu_subdominio>.ngrok-free.app) que retransmitía el tráfico al puerto 8000 de nuestro Cloud Shell. La salida de Ngrok en la terminal confirmó que el túnel estaba activo y reenviando las solicitudes.

4. **Acceso remoto desde la Máquina Virtual del compañero:** Al compartir la URL de Ngrok, el compañero pudo acceder a la aplicación web desde el navegador de su máquina virtual. La visualización de la página con las estadísticas de Messi, demostró que el túnel de Ngrok estaba funcionando correctamente y que el tráfico desde la máquina virtual externa estaba llegando a nuestra aplicación Flask en Cloud Shell. Los logs de Ngrok también registraron las peticiones HTTP con códigos de respuesta 200 OK, confirmando que la aplicación respondió exitosamente a las solicitudes remotas.

Este caso práctico validó la efectividad de la combinación de Docker para la portabilidad y aislamiento de la aplicación, Docker Compose para la orquestación de los

contenedores y la exposición de puertos, y Ngrok para la creación de un punto de acceso público seguro, permitiendo que una aplicación desarrollada en un entorno virtualizado fuera accesible desde una máquina virtual externa a través de Internet.

Tanto como Docker y Máquina Virtual, pudimos ir aplicando los conocimientos teóricos prácticos que se nos fueron dando. Eligiendo como problemática a resolver, crear una appi en Docker, para luego subirla a la red a través de “docker compose” y que esta pueda ser visualizada desde la máquina virtual.

3.Metodología Utilizada

Nuestra Metodología de Trabajo

Para desarrollar este proyecto, seguimos pasos claros y trabajamos en equipo:

1. Investigación y Aprendizaje

Primero, nos dedicamos a aprender sobre el tema. Para eso, usamos:

- El material de estudio (teoría y vídeos) que nos dio la institución.
- Vídeos adicionales de internet que nos ayudaron a entender mejor.

2. Instalación de Programas y Creación de la Máquina Virtual

En esta fase, preparamos el entorno de trabajo. Esto incluyó la instalación de software base (como el sistema operativo en la VM) y la **creación de la máquina virtual (VM)**, y

la asignación de sus recursos posteriormente.

3. Diseño y Pruebas

Una vez el entorno estuvo listo, pasamos al diseño y las pruebas de los componentes:

- Comprobamos que la máquina virtual funcionaba bien a través de un ejercicio en Python.
- Aplicamos el contenido práctico otorgado por la institución en **Docker**: esto abarcó la creación de imágenes, contenedores, redes, y otros aspectos esenciales.
- Una vez probado que los entornos que estábamos utilizando eran comprendidos por nosotros y funcionaban como esperábamos, procedimos con la **configuración del servidor y del cliente en Docker**, lo que incluyó la **creación del archivo docker-compose.yml**.
- Luego de configurar lo necesario en Docker, pasamos a la creación de un pequeño código que combinaba bucles y condicionales sencillos, conocimientos que habíamos adquirido en la materia de programación.
- Como extra, agregamos **HTML**, un conocimiento previo que ya traíamos y pensábamos que podía sumar algo de estética e impronta a la web.
- Una vez creado el código, a través de **Docker Compose** —el cual nos ayudó a evitar tener que escribir una gran cantidad de comandos para la creación de imágenes, contenedores, redes y exposición del puerto, entre otros— corrimos el servidor de manera local.
- A través de la aplicación de **Ngrok**, logramos que la página web, que solo se visualizaba de manera local, pudiera estar expuesta públicamente con un cifrado que le agregaba una capa de seguridad, un tema visto anteriormente en la unidad de redes.
- Por último, decidimos comprobar si este servidor era visible en la máquina virtual.

3. Herramientas que Usamos

Para trabajar, nos apoyamos en varias herramientas:

- **Google Cloud Shell:** plataforma en donde se utilizó Docker.
- **Git Hub:** creación de repositorio, en donde ir subiendo la información solicitada.
- **Docker:** Nos sirvió para que nuestra aplicación funcionará igual en todas las computadoras, como si fuera una "caja" lista para usar.
- **Ngrok:** solución para poder poner de manera pública el servidor creado, además de agregarle una capa de seguridad extra.

4. Trabajo en Equipo

Reparto de Tareas y Colaboración

Para asegurar un desarrollo eficiente y organizado, repartimos las tareas de manera que cada miembro del equipo tuviera roles definidos, sin dejar de lado la colaboración constante:

- **Luciano** fue el encargado de la **instalación y asignación de recursos a la máquina virtual (VM)**, asegurando que tuviéramos un entorno base robusto para trabajar. También lideró la **ideación de la aplicación** a ejecutar y la **investigación clave** sobre cómo lograr una red pública y segura, utilizando herramientas como Ngrok. Además, fue fundamental en la producción de un **contenido audiovisual de calidad** para nuestra presentación.

- **Facundo** se centró en el **manejo de Docker**, incluyendo la **configuración del servidor y cliente**, la creación del **Docker Compose** y la **puesta a prueba del código**. Aportó sus conocimientos de **HTML** para mejorar la estética del proyecto y gestionó la **exposición del puerto 8000** junto con los **scripts necesarios** para la interacción entre Docker Compose y Ngrok. Adicionalmente, se encargó del diseño y creación de la **portada del proyecto** utilizando Illustrator.

A pesar de que cada uno tuvo responsabilidades específicas en distintos momentos,

mantuvimos una comunicación fluida y constante durante todas las etapas de planeación y desarrollo. Esto nos permitió aportar ideas de forma colaborativa y fomentar un **excelente ambiente de trabajo**, lo cual fue clave para el éxito del proyecto.

4.Resultados Obtenidos

Resultados Obtenidos del Caso Práctico

Este caso práctico nos permitió consolidar los conocimientos adquiridos y validar la implementación de un entorno que combina virtualización y contenerización. Logramos un despliegue exitoso de una aplicación web, superando diversas dificultades técnicas en el proceso.

Aspectos que Funcionaron Correctamente

1. **Creación y Configuración de la Máquina Virtual (VM):** Logramos instalar y configurar correctamente una VM con Ubuntu Server utilizando VirtualBox. Esta VM sirvió como la base estable para nuestro entorno de contenerización.
2. **Manejo de Docker Básico:** Implementamos con éxito la creación de imágenes personalizadas, el despliegue de contenedores individuales y la configuración de redes internas en Docker.
3. **Configuración de Nginx en Contenedores:** Pudimos configurar Nginx como servidor web y proxy inverso dentro de un contenedor Docker, sirviendo nuestro contenido HTML y gestionando las solicitudes adecuadamente.
4. **Uso Eficiente de Docker Compose:** La herramienta Docker Compose fue crucial para orquestar los diferentes servicios (servidor, cliente) de nuestra aplicación. Nos permitió levantar y gestionar el entorno completo con un solo comando, validando su utilidad para aplicaciones multicontenedor.
5. **Despliegue Local de la Aplicación Web:** La aplicación web, que combinaba Python y HTML, se ejecutó sin problemas en un contenedor Docker y fue accesible localmente a través del puerto configurado (8000).

6. **Exposición Pública Segura con Ngrok:** El uso de Ngrok fue un éxito rotundo. Pudimos exponer nuestra aplicación web localmente hosteada a internet de forma pública y segura, demostrando la capacidad de hacer accesible un servicio desde la VM a un entorno global con cifrado.
7. **Integración de Conocimientos Transversales:** Logramos integrar conceptos de programación (Python con bucles y condicionales), diseño web (HTML) y redes (exposición de puertos, seguridad), lo que enriqueció el resultado final del proyecto.

Casos de Prueba Realizados

Durante el desarrollo, realizamos diversas pruebas para asegurar la funcionalidad:

- **Pruebas de Instalación de VM:** Verificación de que Ubuntu se instalara y arrancara correctamente con los recursos asignados.
- **Pruebas de Conectividad Básica:** Pings y acceso SSH a la VM.
- **Pruebas de Contenedores Individuales:** Verificación del arranque y correcto funcionamiento de cada servicio (Nginx, aplicación Python) en contenedores separados.
- **Pruebas de Docker Compose:** Ejecución de docker-compose up y docker-compose down para confirmar la orquestación automática de los servicios.
- **Pruebas de Acceso Local:** Verificación del acceso a la aplicación web a través del navegador web desde la VM usando localhost:8000.
- **Pruebas de Acceso Público con Ngrok:** Acceso a la aplicación web desde un dispositivo externo a través del enlace generado por Ngrok, confirmando la visibilidad y el cifrado.
- **Pruebas de Persistencia Básica:** (Si aplica) Verificación de que los cambios en archivos dentro de volúmenes persistentes se mantuvieran.

Dificultades Presentadas y Errores Corregidos

Las siguientes dificultades surgieron durante el proyecto, y su resolución fue clave para nuestro aprendizaje:

- **Problemas durante la Instalación del ISO de Ubuntu:** La carga del instalador de Ubuntu en VirtualBox se quedaba congelada. **Se corrigió** aumentando la RAM y el número de vCPUs asignados a la VM, lo que permitió que la instalación progresara.
- **Errores de Permisos:** Nos encontramos con situaciones donde los comandos requerían permisos más allá de sudo. **Se resolvió** ejecutando sudo su para elevar los privilegios al usuario root, permitiendo la ejecución de operaciones críticas.
- **Gestión Incorrecta de Ramas en GitHub:** Al subir nuestros cambios, los archivos se enviaron inicialmente a la rama master en lugar de main. **Se corrigió** cambiando a la rama main y realizando el push nuevamente, lo que reforzó nuestro entendimiento del control de versiones.
- **Acceso a Carpeta Compartida:** La VM no podía acceder a la carpeta compartida desde el host. **La dificultad se resolvió** al identificar que no se habían instalado las Guest Additions de VirtualBox; se cargó la imagen de las Guest Additions y se procedió a su instalación.
- **Errores de Sintaxis en Docker Compose:** Enfrentamos numerosos problemas con la indentación y errores ortográficos sutiles en el archivo docker-compose.yml, que impedían su correcta ejecución. **Se corrigió** leyendo atentamente la salida de la terminal y los mensajes de error de Docker Compose, lo que nos proporcionó un gran aprendizaje sobre la importancia de la sintaxis precisa.
- **Conflicto en la Exposición del Puerto 5000:** Intentamos exponer la aplicación en el puerto 5000, como se sugería en el material institucional, pero el acceso resultaba en un error 404, a pesar de que la terminal no arrojaba errores explícitos. **La dificultad se resolvió** tras investigar y darnos cuenta de que el puerto 5000 podría estar reservado o en uso por otra aplicación en la máquina host. Se optó por utilizar el **puerto 8000**, el cual funcionó correctamente.
- **Limitación de Acceso Local del Hosting:** Inicialmente, comprendimos que el sitio web solo sería accesible desde la máquina local (o la VM directamente). **Se solucionó** implementando **Ngrok**, lo que nos permitió hacer la aplicación

accesible públicamente, superando la restricción de un hosting meramente local y posibilitando la prueba desde otros dispositivos

5. **Conclusiones del Grupo de Trabajo**

Este proyecto práctico representó una valiosa oportunidad para **profundizar en el mundo de la virtualización y la contenerización**. Logramos no solo comprender los conceptos teóricos, sino también aplicarlos de manera práctica, lo que nos permitió observar las diferencias, ventajas y escenarios de uso ideales para **máquinas virtuales (VMs)** y **contenedores Docker**. La experiencia de trabajar con herramientas como **VirtualBox** y **Docker** fue fundamental para solidificar nuestro aprendizaje.

Aprendizajes Clave

Durante el desarrollo, adquirimos conocimientos esenciales sobre:

- **Virtualización:** Entendimos cómo crear y configurar entornos virtuales utilizando VirtualBox.
- **Contenerización con Docker:** Nos familiarizamos con la gestión de contenedores, la creación de imágenes, el manejo de redes y la orquestación con Docker Compose.
- **Comparativa de Tecnologías:** Comprendimos las ventajas y desventajas de las VMs frente a los contenedores, y cuándo es más apropiado utilizar cada uno.
- **Lectura de la Terminal:** La depuración de errores de Docker Compose, particularmente los relacionados con la indentación y la sintaxis, nos enseñó la importancia crucial de leer e interpretar los mensajes de error de la terminal.

Dificultades Encontradas y Soluciones

A lo largo del proceso, surgieron diversos desafíos que pusimos a prueba nuestra capacidad de resolución de problemas:

- **Instalación de Ubuntu en VirtualBox:** Tuvimos problemas iniciales con la congelación durante la carga del archivo .iso de Ubuntu. Esto se resolvió

aumentando los recursos (RAM y vCPU) asignados a la VM y siendo pacientes con el tiempo de carga.

- **Problemas de Permisos:** Nos encontramos con errores de permisos persistentes que no se solucionaban con sudo. La solución fue **elevar los privilegios al usuario root** mediante el comando sudo su, lo que nos permitió realizar las operaciones necesarias.
- **Gestión de Ramas en GitHub:** Al subir archivos, inicialmente los enviamos a la rama master en lugar de main. Este error se corrigió **cambiando a la rama correcta** y realizando el push nuevamente.
- **Acceso a Carpetas Compartidas:** La imposibilidad de acceder a carpetas compartidas en la VM se debió a la falta de **Guest Additions**. Resolvimos esto **cargando la imagen e instalando** esta herramienta en la máquina virtual.
- **Errores en Docker Compose:** Enfrentamos múltiples errores en el archivo docker-compose.yml, principalmente relacionados con la **indentación y la sintaxis**. Superamos estas dificultades **leyendo cuidadosamente los mensajes de error** de la terminal, lo que se convirtió en una valiosa lección sobre la depuración.
- **Exposición de Puertos:** Al intentar exponer el puerto 5000 (sugerido en el material institucional), obteníamos un error 404 sin mensajes claros en la terminal. Tras investigar, descubrimos que este puerto probablemente **ya estaba en uso** por otra función del sistema operativo. La solución fue **utilizar el puerto 8000** en su lugar.
- **Acceso Remoto al Hosting Local:** Inicialmente, el sitio web solo era accesible localmente, impidiendo que la VM y otros dispositivos de prueba pudieran visualizarlo. Esto se resolvió eficazmente mediante el uso de la aplicación **Ngrok**, que nos permitió **exponer públicamente el sitio web local** con una capa de seguridad.

Posibles Mejoras y Extensiones Futuras

Pensando en el futuro, identificamos varias áreas para mejorar o extender este trabajo:

- **Persistencia de Datos:** Implementar bases de datos en contenedores Docker con volúmenes persistentes para la gestión de datos.

- **Orquestación Avanzada:** Explorar herramientas de orquestación más robustas como Kubernetes para la gestión de aplicaciones a gran escala.
- **Contenedores Más Complejos:** Desarrollar aplicaciones más complejas que involucren múltiples servicios interconectados dentro de Docker.

5.Bibliografía

Contenido teórico y práctico:

Fuentes otorgadas por la institución academica:

https://www.youtube.com/watch?v=_OAZZZeP4tE&feature=youtu.be
https://www.youtube.com/watch?v=_OAZZZeP4tE&feature=youtu.be
https://www.youtube.com/watch?v=Ehm2eMmlt3Q&feature=youtu.be
https://youtu.be/2rlEQiUEuEA
https://youtu.be/gWhN2XVJqAg
https://youtu.be/cVNQkQAAh84

Fuentes externas:

https://dionarodrigues.dev/blog/how-to-use-docker-images-containers-volumes-and-bind-mounts
1. Introducción, docker vs VMs, OCI, casos de uso... - Curso Docker gratuito y en español - YouTube
Necesitas APRENDER a VIRTUALIZAR AHORA | Curso COMPLETO de Virtualización con VirtualBox - YouTube
Máquinas Virtuales y Contenedores - ¿Qué son? ¿A qué huelen? ¿Para qué sirven?
¿Qué es la VIRTUALIZACIÓN? ¿Para Qué sirve la Virtualización? | Tipos de VIRTUALIZACIÓN DE SISTEMAS
¿Qué es la virtualización? - Explicación de la virtualización de la computación en la nube - AWS
Virtualización, máquinas virtuales y contenedores: ventajas y desventajas ¡Bien Explicado!
Cómo Funcionan las MÁQUINAS VIRTUALES? - YouTube
Diferencias entre Maquinas Virtuales y Contenedores - Docker - YouTube

Sintaxis y usos:

https://docker-curriculum.com/

https://imaginaformacion.com/tutoriales/comandos-docker