⚠️ IMPORTANTE – Guía de Práctica Sugerida

Lo que vas a ver a continuación es una **guía paso a paso altamente sugerida** para que practiques el uso de Docker.
Te recomendamos hacerla completa, ya que te ayudará a adquirir los conocimientos necesarios.

PERO: Esta guía **NO es el trabajo práctico** que tenés que entregar

El trabajo práctico será evaluado en base a:
- Tu capacidad para **organizar tu trabajo en Docker con criterio técnico**.
- Tu capacidad para **explicar y justificar cada decisión que tomaste**.
- Una **defensa oral obligatoria** donde vas a tener que demostrar lo que sabés.

¿Dónde está el trabajo práctico?

El **TP real que debés entregar y defender** se encuentra al final de este archivo.
No alcanza con copiar esta guía. **Si no podés defenderlo, no se aprueba.**

Sobre esta guía

- Esta guía NO es exhaustiva.
- Docker es una tecnología que requiere **investigación y práctica fuera de clase**.
- En 2 horas no vas a aprender Docker completo. **Esto es solo el punto de partida.**

Guía Paso a Paso – Introducción a Docker (Práctica sugerida)

1- Objetivos de Aprendizaje
- Familiarizarse con la tecnología de contenedores
- Ejercitar comandos básicos de Docker.

2- Unidad temática que incluye este trabajo práctico
Este trabajo práctico corresponde a la unidad Nº: 2 (Libro Ingeniería de Software: Unidad 18)

3- Algunos conceptos fundamentales

A continuación, se presentarán algunos conceptos generales de la tecnología de contenedores a manera de introducción al tema desde el punto de vista práctico.

¿Qué son los contenedores?

Los contenedores son paquetes de software. Ellos contienen la aplicación a ejecutar junto con las librerías, archivos de configuración, etc. para que esta aplicación pueda ser ejecutada. Estos contenedores utilizan características del sistema operativo, por ejemplo, cgroups, namespaces y otros aislamientos de recursos (sistema de archivos, red, etc.) para proveer un entorno aislado de ejecución de dicha aplicación.

Dado que ellos utilizan el kernel del sistema operativo en el que se ejecutan, no tienen el elevado consumo de recursos que por ejemplo tienen las máquinas virtuales, las cuales corren su propio sistema operativo.

¿Qué es Docker?

Docker es una herramienta que permite el despliegue de aplicaciones en contenedores. Además, provee una solución integrada tanto para la ejecución como para la creación de contenedores entre otras muchas funcionalidades.

¿Por qué usar contenedores?

Los contenedores ofrecen un mecanismo de empaquetado lógico en el cual las aplicaciones pueden estar aisladas del entorno en el cual efectivamente se ejecutan. Este desacoplamiento permite a las aplicaciones en contenedores ser desplegadas de manera simple y consistente independientemente de si se trata de un Data Center privado, una Cloud pública, o una computadora de uso personal. Esto permite a los desarrolladores crear entornos predecibles que están aislados del resto de las aplicaciones y pueden ser ejecutados en cualquier lugar.

Por otro lado, ofrecen un control más fino de los recursos y son más eficientes al momento de la ejecución que una máquina virtual.

En los últimos años el uso de contenedores ha crecido exponencialmente y fue adoptado de forma masiva por prácticamente todas las compañías importantes de software.

Máquinas Virtuales vs Contenedores

Los contenedores no fueron pensados como un reemplazo de las máquinas virtuales. Cuando ambas tecnologías se utilizan en forma conjunta se obtienen los mejores resultados, por ejemplo, en los proveedores cloud como AWS, Google Cloud o Microsoft Azure.

[imagen]: vms-vs-containers.png

(Imagen: https://blog.docker.com/2016/04/containers-and-vms-together/)

Analogía

[imagen3]: vms-containers-analogy.png

(Imagen: https://github.com/SteveLasker/Presentations/tree/master/DockerCon2017)

Conceptos Generales

- **Container Image**: Una imagen contiene el sistema operativo base, la aplicación y todas sus dependencias necesarias para un despliegue rápido del contenedor.
- **Container**: Es una instancia en ejecución de una imagen.
- **Container Registry**: Las imágenes de Docker son almacenadas en un Registry y pueden ser descargadas cuando se necesitan. Un registry puede ser público, por ejemplo, DockerHub o instalado en un entorno privado.
- **Docker Daemon**: el servicio en segundo plano que se ejecuta en el host que gestiona la construcción, ejecución y distribución de contenedores Docker. El daemon es el proceso que se ejecuta en el sistema operativo con el que los clientes hablan.
- **Docker Client**: la herramienta de línea de comandos que permite al usuario interactuar con el daemon. En términos más generales, también puede haber otras formas de clientes, como Kitematic, que proporciona una GUI a los usuarios.
- **Dockerfile**: Son usados por los desarrolladores para automatizar la creación de imágenes de contenedores. Con un Dockerfile, el demonio de Docker puede automáticamente construir una imagen.

Layers en Docker

Las imágenes de Docker están compuestas de varias capas (layers) de sistemas de archivos y agrupadas juntas. Estas son de solo lectura. Cuando se crea el contenedor, Docker monta un sistema de archivos de lectura/escritura sobre estas capas el cual es utilizado por los procesos dentro del contenedor. Cuando el contenedor es borrado, esta capa es borrada con él, por lo tanto, son necesarias otras soluciones para persistir datos en forma permanente.

[imagen2]: docker-image.png

(Imagen:
https://washraf.gitbooks.io/the-docker-ecosystem/content/Chapter%201/Section%203/union_file_system.html)

4- Desarrollo de la Guía

1- Instalar Docker Community Edition
- Diferentes opciones para cada sistema operativo
- https://docs.docker.com/
- Ejecutar el siguiente comando para comprobar versiones de cliente y demonio.

```bash
docker version
```

2- Explorar DockerHub
- Registrarse en Docker Hub: https://hub.docker.com/
- Familiarizarse con el portal

3- Obtener la imagen BusyBox
- Ejecutar el siguiente comando, para bajar una imagen de DockerHub
```bash
docker pull busybox
```
- Verificar qué versión y tamaño tiene la imagen bajada, obtener una lista de imágenes locales:
```bash
docker images
```

4- Ejecutando contenedores
- Ejecutar un contenedor utilizando el comando **run** de docker:
```bash
docker run busybox
```

- Explicar por qué no se obtuvo ningún resultado

- Especificamos algún comando a correr dentro del contenedor, ejecutar por ejemplo:
```bash
docker run busybox echo "Hola Mundo"
```

- Ver los contenedores ejecutados utilizando el comando **ps**:
```bash
docker ps
```
- Vemos que no existe nada en ejecución, correr entonces:
```bash
docker ps -a
```
- Mostrar el resultado y explicar qué se obtuvo como salida del comando anterior.

5- Ejecutando en modo interactivo

- Ejecutar el siguiente comando
```bash
docker run -it busybox sh
```

- Para cada uno de los siguientes comandos dentro del contenedor, mostrar los resultados:
```bash
ps
uptime
free
ls -l /
```
- Salimos del contenedor con:
```bash
exit
```

6- Borrando contenedores terminados

- Obtener la lista de contenedores
```bash
docker ps -a
```
- Para borrar podemos utilizar el id o el nombre (autogenerado si no se especifica) del contenedor que se desee, por ejemplo:
```bash
docker rm elated_lalande
```
- Para borrar todos los contenedores que no estén corriendo, ejecutar cualquiera de los siguientes comandos:
```bash
docker rm $(docker ps -a -q -f status=exited)
```
```bash
docker container prune
```

7- Construir una imagen
- Conceptos de DockerFile
  - Leer https://docs.docker.com/engine/reference/builder/
  - Describir las instrucciones
    - FROM
    - RUN
    - ADD
    - COPY
    - EXPOSE
    - CMD
    - ENTRYPOINT
- A partir del código https://github.com/ingsoft3ucc/SimpleWebAPI crearemos una imagen.
- Clonar repo
- Crear imagen etiquetándola con un nombre. El punto final le indica a Docker que use el directorio actual
```bash
```

docker build -t mywebapi .
```
- Revisar Dockerfile y explicar cada línea
- Ver imágenes disponibles
- Ejecutar un contenedor con nuestra imagen
- Subir imagen a nuestra cuenta de Docker Hub
  - 7.1 Inicia sesión en Docker Hub
    - Primero, asegúrate de estar autenticado en Docker Hub desde tu terminal:
    ```bash
    docker login
    ```
  - 7.2 Etiquetar la imagen a subir con tu nombre de usuario de Docker Hub y el nombre de la imagen. Por ejemplo:
    ```bash
    docker tag <nombre_imagen_local> <tu_usuario_dockerhub>/<nombre_imagen>:<tag>
    ```
  - 7.3 Subir la Imagen
    - Para subir la imagen etiquetada a Docker Hub, utiliza el comando docker push:
     ```bash
     docker push <tu_usuario_dockerhub>/<nombre_imagen>:<tag>
     ```
  - 7.4 Verificar la Subida
     ```bash
     docker pull <tu_usuario_dockerhub>/<nombre_imagen>:<tag>
     ```

8- Publicando puertos

En el caso de aplicaciones web o base de datos donde se interactúa con estas aplicaciones a través de un puerto al cual hay que acceder, estos puertos están visibles solo dentro del contenedor. Si queremos acceder desde el exterior deberemos exponerlos.

- Ejecutar la siguiente imagen, en este caso utilizamos la bandera -d (detach) para que nos devuelva el control de la consola:

```bash
docker run --name myapi -d mywebapi
```
- Ejecutamos un comando ps:
- Vemos que el contenedor expone 3 puertos el 80, el 5254 y el 443, pero si intentamos en un navegador acceder a http://localhost/WeatherForecast no sucede nada.

- Procedemos entonces a parar y remover este contenedor:
```bash
docker kill myapi
docker rm myapi
```
- Vamos a volver a correrlo otra vez, pero publicando el puerto 80
```

```bash
docker run --name myapi -d -p 80:80 mywebapi
```
- Accedamos nuevamente a http://localhost/WeatherForecast y vemos que nos devuelve datos.

9- Modificar Dockerfile para soportar bash

- Modificamos Dockerfile para que entre en bash sin ejecutar automáticamente la app

```bash
#ENTRYPOINT ["dotnet", "SimpleWebAPI.dll"]
CMD ["/bin/bash"]
```
- Rehacemos la imagen
```bash
docker build -t mywebapi .
```
- Corremos contenedor en modo interactivo exponiendo puerto
```bash
docker run -it --rm -p 80:80 mywebapi
```
- Navegamos a http://localhost/weatherforecast
- Vemos que no se ejecuta automáticamente
- Ejecutamos app:
```bash
dotnet SimpleWebAPI.dll
```
- Volvemos a navegar a http://localhost/weatherforecast
- Salimos del contenedor

10- Montando volúmenes

Hasta este punto los contenedores ejecutados no tenían contacto con el exterior, ellos corrían en su propio entorno hasta que terminaran su ejecución. Ahora veremos cómo montar un volumen dentro del contenedor para visualizar por ejemplo archivos del sistema huésped:

- Ejecutar el siguiente comando, cambiar myusuario por el usuario que corresponda. En Mac puede utilizarse /Users/miusuario/temp):
```bash
docker run -it --rm -p 80:80 -v /Users/miuser/temp:/var/temp  mywebapi
```
- Dentro del contenedor correr
```bash
ls -l /var/temp
touch /var/temp/hola.txt
```

```
- Verificar que el archivo se ha creado en el directorio del guest y del host.

### 11- Utilizando una base de datos
- Levantar una base de datos PostgreSQL

```bash
mkdir $HOME/.postgres

docker run --name my-postgres -e POSTGRES_PASSWORD=mysecretpassword -v
$HOME/.postgres:/var/lib/postgresql/data -p 5432:5432 -d postgres:9.4
```
- Ejecutar sentencias utilizando esta instancia

```bash
docker exec -it my-postgres /bin/bash

psql -h localhost -U postgres

#Estos comandos se corren una vez conectados a la base

\l
create database test;
\connect test
create table tabla_a (mensaje varchar(50));
insert into tabla_a (mensaje) values('Hola mundo!');
select * from tabla_a;

\q

exit
```

- Conectarse a la base utilizando alguna IDE (DBeaver - https://dbeaver.io/, Azure Data Studio - https://azure.microsoft.com/es-es/products/data-studio, etc.). Interactuar con los objetos creados.

- Explicar qué se logró con el comando `docker run` y `docker exec` ejecutados en este ejercicio.

12- Hacer el punto 11 con otra base de datos
- Armar un contenedor con otra base de datos de tu elección (MySQL, MongoDB, etc.)
- Crear BD, Tablas/Colecciones y ejecutar consultas

Trabajo Práctico 02 – Introducción a Docker (2025)

🎯 Objetivo
```

Aplicar y demostrar el uso práctico de Docker mediante un caso simulado de trabajo en equipo.
Este trabajo se aprueba **solo si podés explicar qué hiciste, por qué lo hiciste y cómo lo resolviste**.

🧩 Escenario

Recibiste las siguientes tareas como parte de un equipo de desarrollo que trabaja con aplicaciones containerizadas:

1. Elegir una aplicación web en la tecnología que prefieras y containerizarla.
2. Integrar una base de datos en contenedor para desarrollo local.
3. Configurar el despliegue de la aplicación en dos entornos diferentes (QA y PROD) usando la misma imagen.
4. Asegurar persistencia de datos mediante volúmenes.
5. Configurar variables de entorno para que la misma imagen pueda comportarse de forma diferente en QA y PROD.
6. Preparar una versión estable lista para despliegue.

📋 Tareas que debés cumplir

1. Elegir y preparar tu aplicación
- Elegí una aplicación web para containerizar (puede ser propia, de un tutorial, o un proyecto simple)
- Creá un repositorio en GitHub para tu proyecto
- Configurá tu entorno Docker y dejá constancia en el archivo `decisiones.md` de cómo lo hiciste y por qué elegiste esa aplicación.

2. Construir una imagen personalizada
- Creá un `Dockerfile` para la aplicación.
- Elegí la imagen base que consideres más adecuada (justificá tu elección en `decisiones.md`).
- Etiquetá la imagen con tu usuario de Docker Hub y un tag significativo.
- Explicá en `decisiones.md` las instrucciones utilizadas y el porqué de la estructura del Dockerfile.

3. Publicar la imagen en Docker Hub
- Subí la imagen a tu cuenta de Docker Hub.
- Explicá en `decisiones.md` la estrategia de versionado de imágenes.

4. Integrar una base de datos en contenedor
- Elegí la base de datos que prefieras (PostgreSQL, MySQL, MongoDB, SQL Server, etc.)
- Montá un volumen persistente para datos.
- Mostrá cómo conectaste la aplicación al contenedor de base de datos.

- Justificá en `decisiones.md` por qué elegiste esa base de datos.

5. Configurar QA y PROD con la misma imagen
- Usá variables de entorno para que la misma imagen corra con diferentes configuraciones según el entorno (ej: cadenas de conexión, modos de log, etc.).
- Correr dos contenedores simultáneamente (uno QA y uno PROD) usando la misma imagen, cada uno con su configuración.
- Justificar en `decisiones.md` cómo definiste las variables de entorno y cómo se aplican.

6. Preparar un entorno reproducible con docker-compose
- Creá un archivo `docker-compose.yml` que levante:
  - La app en QA
  - La app en PROD
  - La base de datos correspondiente
- Configurá volúmenes y variables de entorno en `docker-compose.yml`.
- Documentá cómo asegurarías que este entorno se ejecute igual en cualquier máquina.

7. Crear una versión etiquetada
- Etiquetá la imagen de la aplicación con un tag `v1.0` y actualizá el `docker-compose.yml` para usar esa versión.
- Explicá la convención de versionado elegida.

📄 Entregables

1. **Repositorio en GitHub** con:
   - Dockerfile funcional.
   - docker-compose.yml para QA y PROD.
   - Imágenes publicadas en Docker Hub con al menos dos tags (uno de desarrollo y uno estable).
   - Configuración de volúmenes y variables de entorno.
   - **README.md** con instrucciones detalladas de cómo:
     - Construir las imágenes
     - Ejecutar los contenedores
     - Acceder a la aplicación (URLs, puertos)
     - Conectarse a la base de datos
     - Verificar que todo funciona correctamente
2. Archivo `decisiones.md` explicando:
   - Elección de la aplicación y tecnología utilizada.
   - Elección de imagen base y justificación.
   - Elección de base de datos y justificación.
   - Estructura y justificación del Dockerfile.
   - Configuración de QA y PROD (variables de entorno).
   - Estrategia de persistencia de datos (volúmenes).
   - Estrategia de versionado y publicación.
   - **Evidencia de funcionamiento**: capturas de pantalla o logs mostrando:
     - La aplicación corriendo en ambos entornos

    - Conexión exitosa a la base de datos
    - Datos persistiendo entre reinicios de contenedor
  - Problemas y soluciones.

🗣️ Defensa Oral Obligatoria

Vas a tener que mostrar tu trabajo y responder preguntas como:
- ¿Cuál es la diferencia entre `CMD` y `ENTRYPOINT`?
- ¿Qué es un volumen en Docker y cómo lo implementaste?
- ¿Cómo publicaste los puertos y por qué de esa manera?
- ¿Cómo controlaste que QA y PROD tengan configuraciones distintas sin cambiar la imagen?

✅ Evaluación

Criterio	Peso
Organización técnica de los contenedores	20%
Claridad y justificación en `decisiones.md`	30%
Defensa oral: comprensión y argumentación	50%

⚠️ Uso de IA

Podés usar IA (ChatGPT, Copilot), pero **deberás declarar qué parte fue generada con IA** y justificar cómo la verificaste.
Si no podés defenderlo, **no se aprueba**.

curl -sSL https://raw.githubusercontent.com/ArnonNahmias/INGSOFT3/main/TP2/install.sh | bash