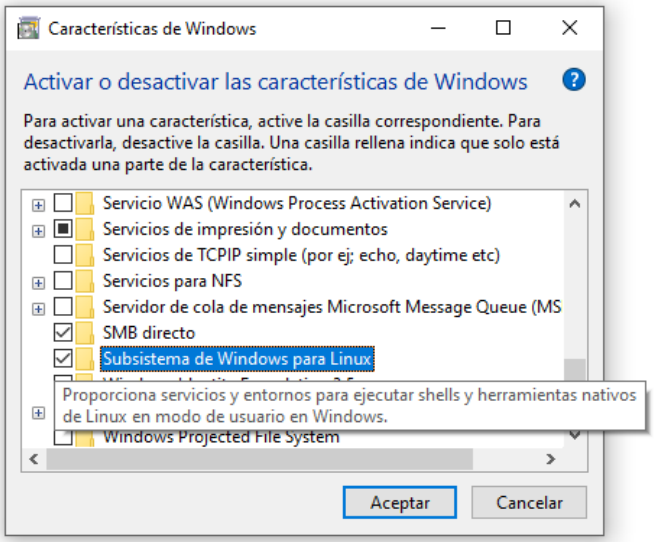
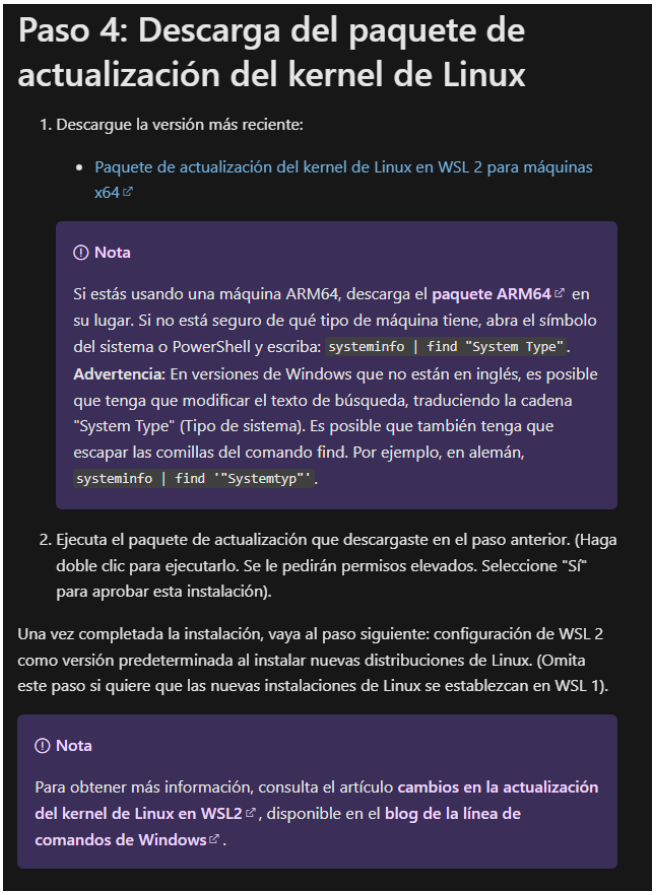
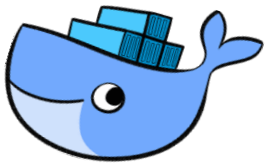
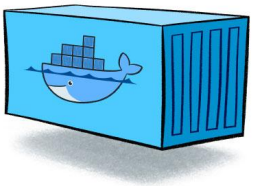
Instalación:

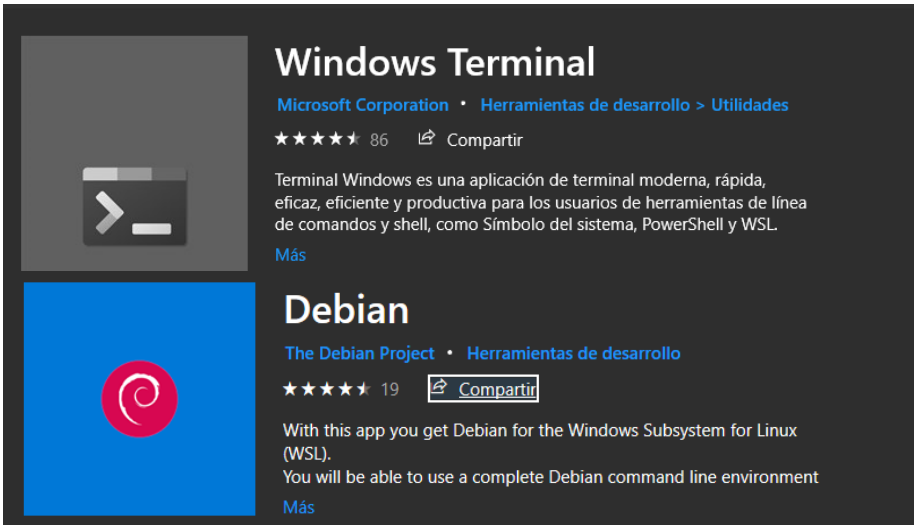




1- Activamos el subsistema linux.
2- Descargamos el paquete wsl. Link
3- Instalamos Windows Terminal. Link
4- Instalamos Debian en WT. Link
5- Descargamos Docker Desktop. Link







¡Pásate por mi web! 👉 lucas-jb

Comandos básicos Docker 🐳

version
Nos indica la versión de Docker instalada.

```
lucas@1DAM:/home/lucas$ docker --version
```

pull
Nos permite descargar imágenes de Dockerhub (repositorio de Docker).
El siguiente ejemplo se trata de la imagen del servidor HTTP Apache.

```
lucas@1DAM:/home/lucas$ docker pull httpd
```

images
Enumera todas las imágenes en el sistema con detalles como:
ETIQUETA / ID DE IMAGEN / TAMAÑO, etc.

```
lucas@1DAM:/home/lucas$ docker images
```

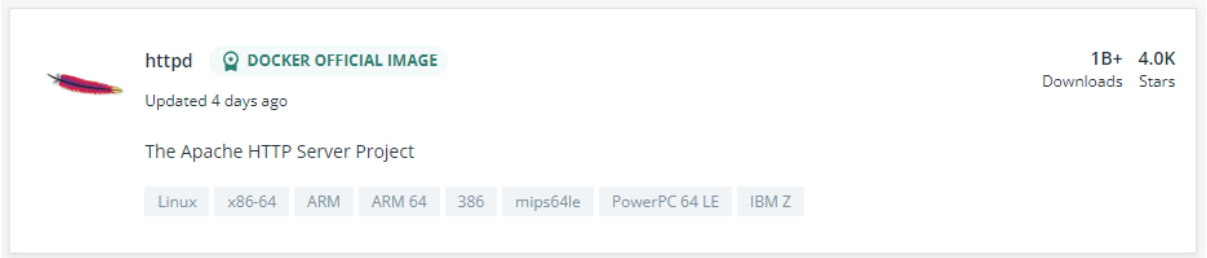
run
Ejecuta la imagen mencionada en el comando. Este comando creará un contenedor acoplable en el que se ejecutará el servidor HTTP Apache.

```
lucas@1DAM:/home/lucas$ docker run -it -d httpd
```

ps
Enumera todos los contenedores Docker que se están ejecutando con los detalles del contenedor. (añadiendo -a veremos los contenedores apagados)

```
lucas@1DAM:/home/lucas$ docker ps
```

PRÁCTICA 1: “Levanta tu web con Apache”




```
lucas@1DAM:/home/lucas$ docker pull httpd
```

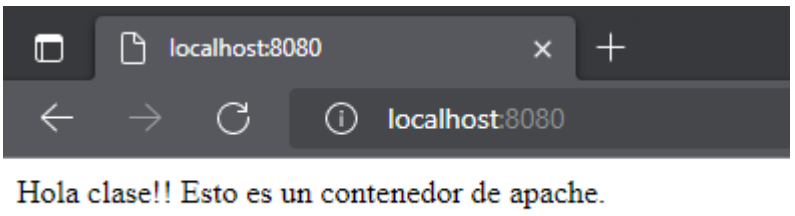
```
lucas@1DAM:/home/lucas$ docker run -dit --name my-apache-app
-p 8080:80 -v "$PWD":/usr/local/apache2/htdocs/ httpd
```

El comando **--name** indica el nombre que le daremos a nuestro contenedor.

En “**$PWD**” introduciremos la ruta donde se montará el volumen. En este caso será montado sobre el directorio htdocs del contenedor. En ese directorio vamos a introducir el html que queramos añadir a nuestro servidor web.

Mediante **-p** indicamos el puerto de salida de nuestro pc y el puerto de salida del contenedor. Esto quiere decir que nuestro contenedor apache utiliza el puerto **80 (http)** y sale por el **8080** de nuestro equipo.

Al conectarnos a nuestro **localhost** y a través del puerto preestablecido (8080), podemos ver nuestra página web sin problemas.



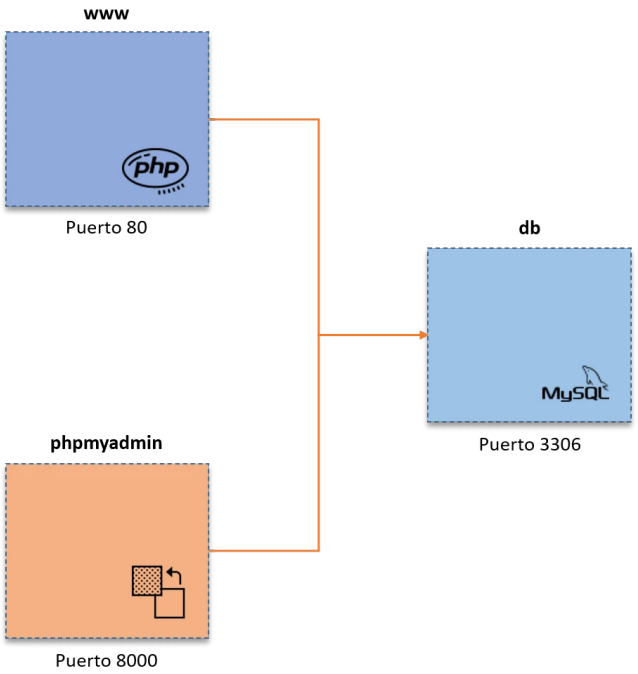
PRÁCTICA 2: “Crea tu propio Server Stack: LAMP” 🐳

Para esta práctica vamos a utilizar 3 contenedores: PHP, MySQL y Apache.

Vamos a crear una aplicación uniendo los tres contenedores con la ayuda de docker-compose.

Puedes utilizar la plantilla del proyecto o construir tu aplicación desde cero con la ayuda de esta guía.

¡A darle duro!



Testing:

Podéis bajaros la estructura del proyecto completo desde mi repositorio.

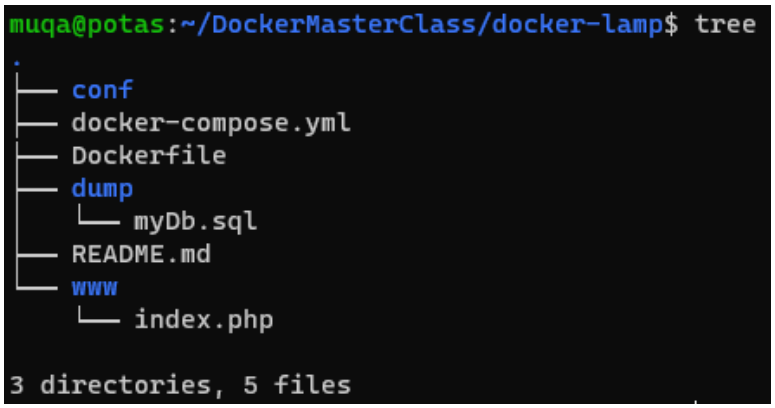
```
lucas@1DAM:/home/lucas$ apt install git
```

```
lucas@1DAM:/home/lucas$ git clone
https://github.com/lucas-jb/docker-lamp.git
```

```
lucas@1DAM:/home/lucas$ docker-compose up -d
```

… y a funcionar! :D

Ahora bien, hagamos una inspección rápida de los ficheros de configuración que tenemos en el repositorio.



Vamos a comenzar inspeccionando el dockerfile, podemos ver que usamos una imagen de **php** en la versión **8.0.0** con apache.

Se define un argumento donde se modifica una variable de entorno **DEBIAN_FRONTED** a modo no interactivo, que lo define para que los demás paquetes de instalación en consecuencia que soliciten una confirmación de algo en ejecución, lo omitan.

Las demás ejecuciones ya son para instalar extensiones de comunicación con **docker**, **php** y **mysqli**, así como las actualizaciones del repositorio de paquetes y por último la instalación de librerías y dependencias.

Dockerfile:

```
FROM php:8.0.0-apache
ARG DEBIAN_FRONTEND=noninteractive
RUN docker-php-ext-install mysqli
RUN apt-get update \
    && apt-get install -y libzip-dev \
    && apt-get install -y zlib1g-dev \
    && rm -rf /var/lib/apt/lists/* \
    && docker-php-ext-install zip

RUN a2enmod rewrite
```

Ahora vamos a inspeccionar el fichero "**docker-compose.yml**". Encontramos 3 bloques importantes donde se definen los servicios: **www**, **db** y **phpmyadmin**.

```
version: "3.1"
services:
    www:
        build: .
        ports:
            - "80:80"
        volumes:
            - ./www:/var/www/html
        links:
            - db
        networks:
            - default
    db:
        image: mysql:8.0
        ports:
            - "3306:3306"
        command:
--default-authentication-plugin=mysql_native_password
        environment:
            MYSQL_DATABASE: dbname
            MYSQL_USER: root
            MYSQL_PASSWORD: test
            MYSQL_ROOT_PASSWORD: test
        volumes:
            - ./dump:/docker-entrypoint-initdb.d
            - ./conf:/etc/mysql/conf.d
            - persistent:/var/lib/mysql
        networks:
            - default
    phpmyadmin:
        image: phpmyadmin/phpmyadmin
        links:
            - db:db
        ports:
            - 8000:80
        environment:
            MYSQL_USER: root
            MYSQL_PASSWORD: test
            MYSQL_ROOT_PASSWORD: test
volumes:
    persistent:
```

Ahora vamos a analizar cada bloque por separado.

www

Este será el contenedor principal, cuya imagen hemos construido desde el Dockerfile. Tiene los siguientes atributos:

- **build** tiene como valor ., indicando que se construirá el .dockerfile sobre este contenedor.
- **ports** mantendrá el mismo puerto con que el servicio internamente está corriendo, que es el 80 y será expuesto hacia afuera.
- **volumes** aplica una definición de donde sincronización entre 2 directorios o volumen de datos, para ello, se sincroniza ./www con /var/www/html que se encuentra en el contenedor.
- **links** selecciona con quien podrá verse y compartir recursos en red, para ello es necesario que el servidor web esté conectado al sistema gestor de base de datos que se encuentra en el contenedor db.
- **networks** define el nombre de la red por la cual estarán conectados los contenedores, para este caso default.

```
www:
    build: .
    ports:
        - "80:80"
    volumes:
        - ./www:/var/www/html
    links:
        - db
    networks:
        - default
```

db

Este bloque creará la bbdd tiene un poco más de atributos que revisar:

- En primera instancia comentar que no tiene un atributo build, dando a entender que no es necesario instalar paquetes desde un .dockerfile como si lo necesitaba el contenedor www.
- **image** selecciona la de mysql en la versión 8.0.
- **ports** mantendrá el mismo puerto con que el servicio internamente está corriendo, que es el **3306** y será expuesto hacia afuera.
- **command** es funcional para que se inserte un comando, donde dicha instrucción será para habilitar la autenticación con la contraseña nativa de MySQL.
- **environment** es la aplicación de variables de entorno, donde este contenedor recibe 4 con el prefijo de MYSQL_ donde escribe el nombre de la base de datos, nombre de usuario y contraseña.
- **volumes** aplica una definición de donde sincronización entre ficheros, así como la configuración y donde corre el demonio de MySQL.
- **networks** define el nombre de la red por la cual estarán conectados los contenedores, para este caso default, donde ya se puede conocer con el contenedor **www**.

```yaml
db:
    image: mysql
    ports:
        - "3306:3306"
    command: --default-authentication-plugin=mysql_native_password
    environment:
        MYSQL_DATABASE: dbname
        MYSQL_PASSWORD: test
        MYSQL_ROOT_PASSWORD: test
    volumes:
        - ./dump:/docker-entrypoint-initdb.d
        - ./conf:/etc/mysql/conf.d
        - persistent:/var/lib/mysql
    networks:
        - default
```

phpmyadmin

Este bloque construirá phpmyadmin, tiene los siguientes atributos:

- **image** selecciona la de phpmyadmin/phpmyadmin.

- **ports** es diferente a los demás, porque internamente el servicio está funcionando con el puerto **80**, pero al exponerlo se necesita hacer con otro, puesto que colisionaría con el puerto que tiene expuesto el contenedor del servidor web www. De este modo, se expone el puerto **8000**.

- **links** se conecta con el contenedor db para tener acceso al sistema gestor de base de datos.

- **environment** es la aplicación de variables de entorno, donde este contenedor recibe 3 con el prefijo de **MYSQL_** donde escribe el nombre de usuario, contraseña y contraseña de root.

```
phpmyadmin:
    image: phpmyadmin/phpmyadmin
    links:
        - db:db
    ports:
        - 8000:80
    environment:
        MYSQL_USER: root
        MYSQL_PASSWORD: test
        MYSQL_ROOT_PASSWORD: test
```

Para levantar el docker-compose introducimos el siguiente comando desde la carpeta principal del proyecto.

```
lucas@1DAM:/home/lucas$ docker-compose up -d
```