Los **contenedores** son una tecnología que permite empaquetar y ejecutar aplicaciones de manera aislada, con todas sus dependencias y configuraciones necesarias para funcionar correctamente, sin necesidad de un sistema operativo completo para cada aplicación.

Un contenedor es una herramienta que te permite compartimentar una aplicacion y las dependencias que necesita para funcionar de manera aislada.

Características clave de los contenedores:

1. **Aislamiento de aplicaciones**: Los contenedores permiten que cada aplicación se ejecute de forma aislada en su propio entorno, sin interferir con otras aplicaciones en el mismo sistema. Aunque comparten el mismo núcleo del sistema operativo, cada contenedor tiene sus propias bibliotecas y dependencias, lo que garantiza que una aplicación en un contenedor no interfiera con otra.
2. **Portabilidad**: Un contenedor empaqueta no solo el código de la aplicación, sino también todas las dependencias que necesita (librerías, configuraciones, etc.). Esto significa que puedes ejecutar un contenedor en cualquier lugar que soporte la tecnología de contenedores (como Docker), sin preocuparte por diferencias entre sistemas.
3. **Ligereza**: A diferencia de una máquina virtual (VM), los contenedores no necesitan incluir un sistema operativo completo. Esto los hace más livianos en cuanto a consumo de recursos como memoria y almacenamiento. Son rápidos para iniciar y detener, ya que no requieren cargar un OS completo.
4. **Reutilización de recursos**: Los contenedores comparten el mismo núcleo del sistema operativo del host, lo que les permite ser más eficientes que las máquinas virtuales. Por ejemplo, varias aplicaciones dentro de contenedores pueden compartir el mismo sistema operativo subyacente, pero aún ejecutarse de forma aislada.
5. **Escalabilidad**: Debido a su ligereza y la rapidez con la que se inician, los contenedores son ideales para sistemas escalables, donde se necesita lanzar rápidamente muchas instancias de una aplicación en diferentes entornos (por ejemplo, en entornos de microservicios).

Cómo funcionan los contenedores:

Los contenedores agrupan todo lo que una aplicación necesita para ejecutarse: su código, librerías, configuraciones y dependencias. Los contenedores se ejecutan sobre un **motor de contenedores**, como **Docker**, que actúa como intermediario entre los contenedores y el sistema operativo del host. Esto permite que los contenedores se ejecuten en cualquier sistema que soporte ese motor.

Comparación con máquinas virtuales (VMs):

- **Contenedores**: Ejecutan solo la aplicación y sus dependencias. Comparten el mismo núcleo del sistema operativo del host, lo que los hace ligeros y rápidos.
- **Máquinas virtuales**: Ejecutan un sistema operativo completo junto con la aplicación. Tienen mayor aislamiento pero consumen más recursos.

Contenedores Docker

1. Compatibilidad con el Sistema Operativo del Host:
    - Los contenedores Docker comparten el núcleo del sistema operativo (kernel) con el host. Esto significa que si el host está en Ubuntu (Linux), cualquier contenedor basado en Linux, como Debian, puede ejecutarse sin necesidad de virtualizar un nuevo kernel.
    - Docker no emula un sistema operativo completo; simplemente crea un entorno aislado para cada contenedor. En tu ejemplo, el contenedor de Debian no ejecuta el kernel de Debian; usa el kernel de Ubuntu (o del host).
    - Los contenedores usan solo las partes del sistema necesarias para ejecutar aplicaciones específicas, lo que los hace ligeros y eficientes.
2. Sistema de Archivos y Aislamiento:
    - Docker usa un sistema de archivos de capas, donde cada imagen contiene el sistema de archivos del contenedor (como Debian en este caso) y puede añadir cambios sobre él.
    - Cada contenedor tiene su propio sistema de archivos aislado, que se monta desde la imagen de Docker y luego se puede personalizar, pero sin modificar la base del host.

3. Recursos y Aislamiento de Procesos:
    - Docker usa namespaces y cgroups para aislar procesos y asignar recursos específicos (CPU, memoria) a los contenedores.
    - El contenedor de Debian en Docker tendrá sus propios procesos (como si estuviera en un sistema Debian), pero esos procesos son administrados y monitoreados directamente por el kernel de Ubuntu, sin necesidad de hipervisores o capas adicionales.
4. Arranque Rápido:
    - Dado que Docker comparte el kernel del host y no inicia un sistema operativo completo, los contenedores arrancan rápidamente.

Ejemplo del uso de contenedores:

Un desarrollador podría crear un contenedor que contiene una aplicación web junto con todas sus dependencias (librerías, servidor web, etc.). Este contenedor puede ejecutarse en el equipo del desarrollador, en un servidor de producción o incluso en la nube sin necesidad de hacer modificaciones, lo que facilita el despliegue y mantenimiento de aplicaciones en diferentes entornos.

Herramientas populares para manejar contenedores:

- **Docker**: La herramienta más conocida para la creación y gestión de contenedores.
- **Kubernetes**: Un sistema de orquestación que permite gestionar grandes cantidades de contenedores, facilitando su despliegue, escalado y operación.

¿Cómo funciona Docker?

Docker es una plataforma que permite crear, ejecutar y gestionar contenedores. El flujo general de funcionamiento de Docker sigue estos pasos clave:

1. Dockerfile: Creación de la imagen

- Un **Dockerfile** es un archivo de texto que contiene las instrucciones para construir una imagen de Docker. Es como una receta que especifica cómo se debe configurar el contenedor. Incluye:
    - El sistema operativo base.
    - Las dependencias que necesita la aplicación (bibliotecas, frameworks, etc.).
    - El código fuente de la aplicación.
    - Variables de entorno, comandos a ejecutar, etc.

    IMAGEN DOCKER Y DOCKERFILE

    Un DOCKERFILE un archivo de texto que contiene las instrucciones para construir la imagen.

    Una imagen es una plantilla de solo lectura que contiene el sistema de archivos y los elementos necesarios para ejecutar un contenedor. Piensa en la imagen como una "fotografía" de un entorno en un momento específico, con el sistema operativo, las dependencias, las configuraciones y las aplicaciones requeridas para funcionar.

    CMD especifica un comando por defecto que se ejecuta al iniciar el contenedor, pero solo se ejecuta si no se proporciona otro comando en la línea de ejecución. Cuando se usa junto con ENTRYPOINT, **CMD** puede servir como argumento para ENTRYPOINT.
    RUN ejecuta comandos en el contenedor durante la **construcción de la imagen** y crea una nueva capa con el resultado, siendo útil para instalar paquetes o configurar el entorno.

ENTRYPOINT define el comando principal que siempre se ejecutará al iniciar el contenedor, y en este caso, **CMD** sirve de argumento a ENTRYPOINT. ENTRYPOINT no se sobrescribe por comandos en la línea de ejecución, pero **CMD** sí puede ser sobrescrito si se pasan argumentos adicionales al contenedor.

COPY transfiere archivos desde el host al sistema de archivos del contenedor sin ejecutar ningún comando, siendo útil para incluir código, configuraciones o recursos en la imagen.

Ejemplo de sobreescritura de cmd:

- CMD ["echo", "Hola, mundo!"] ---> docker run mi-imagen --->salida: Hola, mundo!
- docker run mi-imagen echo "¡Adiós, mundo!" --->salida: Adios, mundo!

Que es una imagen?

Imagen de Docker: Es un archivo que contiene todo lo necesario para ejecutar una aplicación en un contenedor.

Plantilla: No se ejecuta directamente, se usa para crear contenedores.
Capas: Compuesta de varias capas, cada una con cambios específicos.
Inmutable: No se puede modificar; para hacer cambios, se crea una nueva imagen.
Compartible: Se puede almacenar y compartir a través de registros como Docker Hub.

SISTEMADE CAPAS

Cada instrucción en un Dockerfile (como RUN, COPY, ADD, etc.) genera una nueva capa. Estas capas son inmutables, lo que significa que no se pueden modificar una vez que se han creado.

La primera capa es la imagen base (como ubuntu o alpine)

Docker utiliza un sistema de archivos en capas para gestionar las imágenes. Este sistema permite que varias imágenes compartan capas comunes, lo que ahorra espacio en disco. Por ejemplo, si dos imágenes utilizan la misma imagen base, ambas compartirán esa capa en lugar de duplicarla.

Las capas pueden ser reutilizadas en diferentes imágenes. Esto significa que puedes crear nuevas imágenes a partir de capas existentes, lo que facilita la creación de nuevas configuraciones y entornos.

2. Imagen de Docker: Construcción

- **Imagen**: Es el resultado del Dockerfile. Es una plantilla de solo lectura que contiene todo lo necesario para ejecutar una aplicación (sistema base, dependencias, código, configuración). Las imágenes son portables y reutilizables, lo que significa que una imagen creada en un entorno puede ejecutarse en cualquier otro con Docker instalado.
- Se crea utilizando el comando docker build.

3. Contenedor de Docker: Ejecución

- Un **contenedor** es una instancia en ejecución de una imagen. Cuando lanzas una imagen, Docker crea un contenedor, que es la unidad de ejecución donde corre tu aplicación de forma aislada del sistema host.
- Los contenedores pueden ejecutarse, detenerse, pausarse o eliminarse mediante comandos de Docker (docker run, docker stop, docker rm, etc.).

4. Docker Hub: Distribución de imágenes

- Docker tiene un registro de imágenes público llamado **Docker Hub**, donde puedes buscar, compartir y descargar imágenes de Docker creadas por otros desarrolladores o empresas. Esto facilita el uso de aplicaciones ya empaquetadas en imágenes de Docker.

Componentes clave de Docker:

- **Motor de Docker (Docker Engine)**: El servicio que corre en segundo plano y gestiona los contenedores.
- **CLI de Docker**: La interfaz de línea de comandos desde donde puedes ejecutar comandos como crear, gestionar o eliminar contenedores.

¿Cómo funciona Docker Compose?

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones que constan de varios contenedores de manera sencilla, utilizando un archivo de configuración YAML (docker-compose.yml). Con Docker Compose, puedes orquestar múltiples contenedores que necesiten interactuar entre sí, como en aplicaciones de microservicios.

Función principal:

- **Definir entornos complejos**: Si tienes una aplicación compuesta por múltiples servicios (por ejemplo, una aplicación web con un backend en Node.js, una base de datos en MySQL y un servicio de caché en Redis), **Docker Compose** permite definir todos estos servicios en un archivo YAML.
- **Orquestación**: Ejecuta todos los servicios a la vez con un solo comando (docker-compose up), gestionando la creación y arranque de los contenedores en el orden correcto.

Pasos de funcionamiento de Docker Compose:

1. **Archivo** docker-compose.yml:
    - Este archivo describe cómo deben ser configurados los servicios (contenedores) que conforman la aplicación.
    - Aquí puedes definir qué imagen utilizar para cada servicio, volúmenes (directorio compartido entre el host y el contenedor), redes, variables de entorno, dependencias entre contenedores, puertos expuestos, etc.

Redes y volúmenes:

- **Redes**: Docker Compose crea redes automáticamente para permitir que los contenedores se comuniquen entre ellos. En el archivo YAML, los servicios pueden referenciarse por nombre para conectarse entre sí.
- **Volúmenes**: Los volúmenes permiten compartir datos entre el host y el contenedor. Esto es útil para que los datos persistentes, como bases de datos o configuraciones, se mantengan aunque el contenedor sea eliminado.

Diferencia clave entre Docker y Docker Compose:

- **Docker** te permite manejar contenedores individuales, mientras que **Docker Compose** está diseñado para aplicaciones que constan de múltiples contenedores (servicios) y necesitan ser gestionadas como un conjunto, automatizando la orquestación y comunicación entre ellos.

La diferencia entre una **imagen Docker** utilizada con **Docker Compose** y una imagen Docker utilizada sin Docker Compose radica principalmente en **cómo se manejan los contenedores y servicios asociados**. Sin embargo, la **imagen** en sí es la misma en ambos casos. La diferencia clave está en la **orquestación** y la **gestión de servicios** que ofrece Docker Compose.

1. Imagen Docker sin Docker Compose

Cuando usas una **imagen Docker sin Docker Compose**, estás manejando contenedores de forma individual utilizando la línea de comandos de Docker (docker run, docker build, etc.). Este enfoque es adecuado cuando solo quieres ejecutar uno o unos pocos contenedores que no tienen dependencias complejas.

Ejemplo sin Docker Compose:

Supón que quieres ejecutar un contenedor basado en la imagen de **MySQL**:

docker run --name my_mysql -e MYSQL_ROOT_PASSWORD=my_password -d mysql:5.7

En este caso, el comando:

- **Descarga** (si no la tienes) y ejecuta la **imagen** mysql:5.7.
- Define una variable de entorno para la contraseña del root.

- Ejecuta un solo contenedor aislado.

Limitaciones sin Docker Compose:

- Si necesitas ejecutar varias imágenes que se **comunican entre sí**, tendrás que configurarlas manualmente (por ejemplo, mediante redes, asignación de puertos, o variables de entorno).
- Gestionar servicios más complejos puede ser tedioso porque cada contenedor tiene que configurarse y lanzarse de forma individual.
- No existe un archivo de configuración estandarizado que describa toda la infraestructura en una aplicación.

2. Imagen Docker utilizada con Docker Compose

Cuando utilizas una imagen Docker con **Docker Compose**, puedes definir varios **servicios** (contenedores) en un archivo YAML (docker-compose.yml), que describe cómo deben interactuar esos contenedores entre sí, las redes que deben compartir, los volúmenes que deben usar, y más.

Ejemplo con Docker Compose:

Si quieres ejecutar la misma imagen de MySQL junto con una aplicación web en Node.js, puedes hacerlo fácilmente utilizando Docker Compose.

```
services:
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: my_password
  web:
    image: node:14
    ports:
      - "3000:3000"
    depends_on:
                – db
```

En este caso:

- **db** es el servicio basado en la imagen mysql:5.7.
- **web** es el servicio basado en la imagen node:14, que depende del servicio **db**.
- Ambos contenedores se ejecutan con un solo comando: docker-compose up.

Beneficios de Docker Compose:

- **Automatización**: Docker Compose se encarga de **automatizar** el lanzamiento de varios contenedores, gestionando las dependencias entre ellos. En el ejemplo anterior, **web** depende de **db**, por lo que Compose se asegurará de que **db** esté funcionando antes de lanzar el servicio **web**.
- **Definición centralizada**: Todos los servicios, variables de entorno, volúmenes, redes y dependencias están descritos en un único archivo (docker-compose.yml), lo que facilita su mantenimiento y portabilidad.
- **Redes y volúmenes gestionados**: Docker Compose crea y gestiona redes y volúmenes automáticamente, permitiendo que los contenedores puedan comunicarse por sus nombres de servicio sin necesidad de configuraciones adicionales.

Diferencias clave entre usar una imagen Docker con y sin Docker Compose:

Característica	Sin Docker Compose	Con Docker Compose
Número de contenedores	Principalmente uno a la vez	Varios servicios (contenedores) juntos
Gestión de servicios	Manual (necesitas ejecutar docker run para cada contenedor)	Automatizada mediante un archivo YAML

Característica	Sin Docker Compose	Con Docker Compose
Dependencias entre contenedores	Debes configurarlas manualmente (ej. redes, enlaces)	Define dependencias en el archivo docker-compose.yml
Redes y comunicación	Debes crear y gestionar las redes entre contenedores	Redes gestionadas automáticamente, los servicios se comunican por nombre
Comando de arranque	docker run por contenedor	docker-compose up para todos los servicios
Persistencia de configuración	Necesitas recordar o crear scripts	La configuración se guarda en el archivo docker-compose.yml

Resumen:

- **Sin Docker Compose**: Estás trabajando con una imagen Docker de forma individual, ejecutando y gestionando los contenedores manualmente. Esto es útil para aplicaciones simples o casos en los que no tienes múltiples contenedores que necesiten interactuar.
- **Con Docker Compose**: Utilizas un archivo YAML para **orquestar** varios contenedores que forman una aplicación completa, con redes, volúmenes y dependencias gestionadas automáticamente, simplificando la configuración y despliegue de entornos complejos.

QUE ES UNA RED DOCKER

Docker Network es el sistema que utiliza Docker para gestionar la comunicación entre contenedores, entre contenedores y el host, o entre contenedores y el mundo exterior. Docker ofrece varias opciones de redes para que los contenedores puedan comunicarse entre sí o con otros sistemas de manera flexible y controlada.

Tipos de Redes en Docker

Docker proporciona diferentes tipos de redes, cada una con sus características y casos de uso específicos:

1. **Bridge Network (Red de Puente)**:
    - Es la red por defecto cuando creas un contenedor sin especificar ninguna red.
    - Los contenedores que están conectados a la misma red "bridge" pueden comunicarse entre sí utilizando sus nombres de servicio o contenedor.
    - No están accesibles directamente desde el mundo exterior, a menos que expongas puertos.
    - Ideal para cuando necesitas que varios contenedores se comuniquen en el mismo host sin acceder desde fuera.

VOLUMENES PERSISTENTES

¿Qué es un Montaje en Docker?

1. **Vinculación de Directorios**:
    - Cuando montas un volumen, estás vinculando un directorio de tu máquina (host) con un directorio dentro del contenedor. Esto significa que cualquier cambio realizado en uno de ellos se reflejará en el otro.
    - Por ejemplo, si tienes un contenedor de Nginx que sirve archivos desde /var/www/html, y montas un volumen que apunta a /home/vpeinado/data/nginx en tu host, cualquier archivo que coloques en /home/vpeinado/data/nginx estará disponible en el contenedor en /var/www/html y viceversa.
2. **Persistencia de Datos**:
    - Una de las principales razones para usar volúmenes es la persistencia de datos. Si el contenedor se detiene o se elimina, los datos en el directorio montado permanecen intactos en el sistema host.
    - Esto es especialmente útil para bases de datos y aplicaciones web donde los datos deben conservarse incluso si el contenedor se reinicia o se reemplaza.

¿Cómo Funciona el Montaje?

1. **Vinculación del Volumen**:
    - En la sección de volúmenes, especificamos que el volumen wordpress_data se debe montar en /var/www/html dentro del contenedor de Nginx.
    - Esta línea: - wordpress_data:/var/www/html indica que el contenedor utilizará el volumen definido como wordpress_data.
2. **Ruta en el Host**:
    - La definición del volumen wordpress_data establece que en el sistema host, los datos se almacenarán en /home/vpeinado/data/wordpress.
    - Así que, cuando escribimos device: "/home/vpeinado/data/wordpress", estamos indicando que el directorio en el host que contendrá los archivos de WordPress es data/wordpress.
3. **Sincronización de Archivos**:
    - Cualquier archivo que coloques en el directorio del host (/home/vpeinado/data/wordpress) estará disponible en el contenedor en /var/www/html.
    - Esto significa que si agregas un archivo HTML o una imagen en el directorio del host, podrás acceder a ellos a través del servidor Nginx en el contenedor.

QUE ES NGINX

Es un servidor web, es un software que almacena y entrega páginas web a los usuarios a través de Internet. Cuando un usuario solicita una página (al ingresar una URL en su navegador), el servidor web responde enviando los archivos necesarios (como HTML, CSS, imágenes) para que el navegador los muestre. Además, puede gestionar solicitudes para aplicaciones web dinámicas y manejar tráfico HTTP/HTTPS.

Ventajas de nginx

Manejo de Conexiones Concurrentes: A diferencia de servidores como Apache, Nginx utiliza un modelo de manejo de conexiones asincrónico y basado en eventos. Esto significa que puede manejar miles de conexiones concurrentes sin consumir muchos recursos, ideal para aplicaciones de alta carga.

Bajo Consumo de Memoria: Debido a su arquitectura, Nginx utiliza significativamente menos memoria que otros servidores web bajo carga intensa, mejorando la eficiencia en servidores de recursos limitados.

También facilita la configuración de SSL/TLS para conexiones seguras, manejando certificados y proporcionando herramientas de seguridad avanzadas.

¿Por qué Nginx no debe estar en segundo plano en Docker?

En Docker, **cada contenedor ejecuta un proceso principal**. Docker monitoriza el estado de ese proceso para determinar si el contenedor debe seguir activo o no. Si el proceso principal termina, el contenedor se detiene. El problema con Nginx y otros servidores que funcionan como **daemons** es que, cuando Nginx se ejecuta en segundo plano (como lo haría normalmente fuera de Docker), el proceso principal que Docker está monitorizando termina, lo que hace que Docker piense que el contenedor ha terminado su tarea y lo detenga.

Ejecución normal de Nginx (sin Docker):

- Fuera de Docker, Nginx por defecto se ejecuta como un **daemon**, lo que significa que se inicia y luego se desacopla (se separa) de la terminal, corriendo en segundo plano como un servicio. Este comportamiento es ideal para servidores en un entorno tradicional.

Ejecución de Nginx en Docker:

- En Docker, **el proceso principal debe estar en primer plano**, para que Docker lo supervise. Si Nginx se ejecuta en modo daemon (segundo plano), el proceso en primer plano finalizará, y Docker interpretará que el contenedor ha completado su tarea, lo que provocará que el contenedor se detenga.

  Al ejecutar Nginx con daemon off;, le decimos a Nginx que **no se desacople** y se ejecute en el primer plano. Esto permite que Docker supervise su ejecución continuamente, manteniendo el contenedor activo.

Resumen:

- **Nginx en modo daemon** (segundo plano) provoca que el proceso principal termine, lo que lleva a que Docker detenga el contenedor.
- **Nginx con** daemon off; mantiene el proceso principal en primer plano, asegurando que Docker mantenga el contenedor activo.

ENTRAR A DOCKER DE NGINX

docker exec -it nginx /bin/bash
nginx -v
cat /etc/nginx/nginx.conf
cat /etc/nginx/sites-available ----> en este directorio default es un archivo modelo para hacer la conf

CONFIGURACION DE NGINX

Nginx viene con una configuración por defecto que puede ser suficiente para muchos casos. Sin embargo, para personalizar el comportamiento de Nginx o para servir aplicaciones específicas (como un sitio web o una aplicación PHP), es común agregar configuraciones adicionales en el directorio conf.d o en archivos separados en el directorio sites-available (y habilitarlos en sites-enabled).

Cómo funciona la configuración de Nginx

1. **Configuración por defecto:**
    - La configuración principal de Nginx generalmente se encuentra en /etc/nginx/nginx.conf.
    - Esta configuración puede incluir directivas que controlan el comportamiento general del servidor, como el manejo de errores, la ubicación de archivos de log, y la carga de módulos.
2. **Archivos en** conf.d**:**
    - En el directorio /etc/nginx/conf.d/, puedes colocar archivos de configuración adicionales. Cada archivo en este directorio se incluye automáticamente en la configuración de Nginx.
    - Esto es útil para organizar configuraciones específicas para diferentes sitios o aplicaciones.
3. **Uso de** sites-available **y** sites-enabled**:**
    - Muchos administradores prefieren usar el esquema de sites-available y sites-enabled para gestionar configuraciones de múltiples sitios web.
    - Las configuraciones para cada sitio se colocan en sites-available/, y luego se crea un enlace simbólico a sites-enabled/ para habilitar esas configuraciones.

En la mayoría de las instalaciones de Nginx en sistemas basados en Debian, como Ubuntu, encontrarás un archivo de configuración por defecto en /etc/nginx/sites-available/default. Este archivo se utiliza como un punto de partida para la configuración de tu servidor web y puede ser modificado para ajustarse a tus necesidades específicas.

QUE ES SSL/TLS

SSL (Secure Sockets Layer) y **TLS (Transport Layer Security)** son protocolos de seguridad que se utilizan para cifrar y proteger la comunicación en Internet, como cuando accedes a sitios web mediante **HTTPS**.

- **SSL** fue el protocolo original, pero ha sido reemplazado por **TLS**, que es más seguro y moderno.
- Ambos garantizan que los datos transmitidos entre el usuario y el servidor estén cifrados, lo que protege la información de ser interceptada o manipulada por terceros.

TLS es el protocolo actualmente en uso, aunque a menudo se sigue usando el término "SSL" para referirse a ambos.

TLSv1.3 es la última versión del protocolo **Transport Layer Security (TLS)**, diseñado para mejorar la seguridad y el rendimiento de las conexiones cifradas en Internet. Un **certificado TLSv1.3** se refiere al certificado digital utilizado en la implementación de este protocolo para cifrar la comunicación entre un servidor y un cliente (navegador web).

HTTPS (Hypertext Transfer Protocol Secure) es una extensión segura del **HTTP** (Hypertext Transfer Protocol) que se utiliza para la comunicación en la web. La principal diferencia entre HTTP y HTTPS es que HTTPS cifra la información intercambiada entre el navegador del usuario y el servidor web, lo que ayuda a proteger la privacidad y la integridad de los datos.

MARIADB

MariaDB es un sistema de gestión de bases de datos relacional (RDBMS) que es una bifurcación de MySQL.
La relación entre **MariaDB** y **WordPress** en un entorno de **Docker** se establece principalmente a través de la arquitectura de cliente-servidor. Aquí te explico cómo funciona esta relación y cómo se implementa dentro de Docker:

Relación entre WordPress y MariaDB

1. **Base de Datos**:
    - WordPress necesita una base de datos para almacenar su contenido, configuraciones y datos de usuarios. MariaDB actúa como el sistema de gestión de bases de datos que almacena toda esta información.
    - Cuando WordPress se instala, se conecta a MariaDB para crear las tablas necesarias y almacenar datos.
2. **Conexión**:
    - WordPress se comunica con MariaDB utilizando la API de MySQL (a través de funciones PHP como mysqli o PDO).
    - En el archivo de configuración de WordPress (wp-config.php), se especifican los detalles de la conexión, como el nombre de la base de datos, el usuario, la contraseña y el host.
3. **Operaciones CRUD**:
    - WordPress realiza operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos MariaDB a medida que los usuarios interactúan con el sitio (por ejemplo, al publicar un artículo o registrar un nuevo usuario).

Cliente-Servidor en Docker

1. **Contenedores**:
    - En Docker, tanto WordPress como MariaDB se ejecutan en contenedores separados. Cada contenedor actúa como un entorno aislado que tiene sus propias dependencias y configuraciones.
    - Un archivo docker-compose.yml puede usarse para definir y ejecutar ambos contenedores.

Red de Contenedores:

- Docker crea una red por defecto para los contenedores que permite que se comuniquen entre sí. WordPress, al ejecutarse en su contenedor, puede acceder a la base de datos en el contenedor de MariaDB utilizando el nombre del servicio definido en docker-compose.yml (db en este caso) como el host.
- La comunicación ocurre a través de la red de Docker, lo que significa que los contenedores pueden interactuar sin exponer sus puertos al mundo exterior (excepto aquellos que se definen explícitamente, como el puerto 8000 en el ejemplo anterior).

2. **Aislamiento y Persistencia**:
    - Cada contenedor es independiente, lo que permite que se escalen y gestionen de manera eficiente. Por ejemplo, puedes tener varios contenedores de WordPress que se conectan a un solo contenedor de MariaDB.
    - Los volúmenes (wordpress_data y mariadb_data en el ejemplo) aseguran que los datos se mantengan persistentes incluso si los contenedores se detienen o se recrean.

Resumen

En resumen, **MariaDB** sirve como la base de datos para **WordPress** en un entorno **Docker**, actuando como el "servidor" que almacena datos y el contenedor de **WordPress** actúa como el "cliente" que realiza operaciones en la base de datos. La conexión se realiza a través de la red de Docker, lo que permite que ambos contenedores interactúen de manera eficiente y segura.

El archivo `wp-config.php` es un archivo de configuración fundamental en una instalación de WordPress. Contiene la información necesaria para que WordPress se conecte a la base de datos y también define varias configuraciones

globales del sistema.

Configuracion

Este archivo de configuración corresponde a la configuración de PHP-FPM (FastCGI Process Manager) en un contenedor de Docker para ejecutar WordPress. Específicamente, define cómo PHP-FPM gestiona los procesos que sirven las solicitudes PHP. Aquí tienes una explicación de cada una de las configuraciones:

Configuraciones Generales de PHP-FPM

- **user = www-data** y **group = www-data**:
    - Define al usuario y grupo bajo los cuales se ejecutarán los procesos de PHP-FPM. `www-data` es el usuario por defecto en muchos entornos web (incluyendo Nginx y Apache) y está diseñado para ser un usuario sin privilegios.
- **listen = 0.0.0.0:9000**:
    - PHP-FPM escuchará conexiones en la dirección `0.0.0.0` (todas las interfaces de red) y en el puerto `9000`. Esto significa que otros contenedores en la misma red de Docker pueden acceder a PHP-FPM en este puerto.
- **listen.owner = www-data** y **listen.group = www-data**:
    - Establece el usuario y grupo propietarios del socket que PHP-FPM usa para escuchar conexiones. Esto asegura que los procesos web (como Nginx o Apache, que corren bajo el usuario `www-data`) tengan permiso para comunicarse con PHP-FPM.
- **listen.mode = 0660**:
    - Define los permisos del socket para que solo el usuario y el grupo (`www-data`) tengan acceso de lectura y escritura. Esto aumenta la seguridad al restringir el acceso a otros usuarios.

Configuración del Administrador de Procesos (pm) de PHP-FPM

- **pm = dynamic**:
    - Configura PHP-FPM para que gestione dinámicamente la cantidad de procesos. Esto permite ajustar automáticamente el número de procesos en función de la carga.
- **pm.max_children = 25**:
    - Define el número máximo de procesos simultáneos que pueden estar activos. En este caso, se permite que hasta 25 procesos sirvan solicitudes en paralelo.
- **pm.start_servers = 5**:
    - Especifica el número de procesos que se inician al arrancar PHP-FPM. En este caso, 5 procesos estarán activos desde el inicio.
- **pm.min_spare_servers = 1**:
    - Define el número mínimo de procesos en espera para manejar solicitudes adicionales. Si hay menos de 1 proceso en espera, PHP-FPM creará más para satisfacer la demanda.
- **pm.max_spare_servers = 10**:
    - Establece el número máximo de procesos en espera. Si el número de procesos en espera supera este límite, PHP-FPM detendrá los procesos sobrantes.

Esta configuración asegura que PHP-FPM esté preparado para servir múltiples solicitudes simultáneas de WordPress, con un límite en la cantidad de recursos que puede utilizar.

COMANDOS NECESARIOS
Comprobar acceso:
curl -I -k https://vpeinado.42.fr:443
curl -I -k http://vpeinado.42.fr:80
curl -I -L http://vpeinado.42.fr para comprobar si hay redirecciones

SSH:
ip a
ssh vpeinado@ipmaquina
cp -r 42inception-pyscript/ vpeinado@ipmaquina:/home/vpeinado/Escritorio

# Actualizar el sistema
sudo apt update
sudo apt upgrade

# Instalar Make
sudo apt install make

# Instalar Docker
sudo apt install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release -cs) stable"
sudo apt update
sudo apt install docker-ce
sudo docker --version

# Instalar Docker Compose
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version

# Agregar tu usuario al grupo de Docker (opcional)
sudo usermod -aG docker $USER
desintalar firefox
sudo apt purge firefox-esr
sudo apt autoremove
which firefox

INSTALAR CHROME
wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb
sudo apt install fonts-liberation
sudo dpkg -i google-chrome-stable_current_amd64.deb
sudo apt --fix-broken install

COMANDOS DE DOCKER
docker start [container_name]

Inicia un contenedor que está detenido, pero **no crea** uno nuevo.

docker stop [container_name]

Detiene un contenedor en ejecución de manera **ordenada** (intenta que el contenedor termine sus procesos).

docker down

Es parte de docker-compose y **detiene y elimina todos los contenedores, redes, volúmenes** y demás recursos definidos en el archivo docker-compose.yml.
Uso: docker-compose down

docker system prune

Limpia Docker de **recursos no utilizados**: contenedores detenidos, imágenes sin usar, redes no referenciadas y cachés de construcción. **No afecta a volúmenes por defecto**.
Para eliminar volúmenes no utilizados, añade --volumes: docker system prune --volumes.

docker volume ls

una lista de todos los volúmenes persistentes, con sus respectivos nombres.

docker network ls
as redes creadas, como bridge, host, y cualquier red personalizada.
docker images
una lista de las imágenes que has descargado o construido, junto con sus etiquetas y tamaños.
docker ps -a
todos los contenedores, mostrando el estado de cada uno (ejecutándose o detenido).
docker info

un resumen que muestra la versión de Docker, número de contenedores, imágenes, volúmenes y más.

VERIFICAR USUARIOS DE WORDPRESS

docker exec -it mariadb mysql -u root -p

docker exec: Este es el comando de Docker utilizado para ejecutar un comando en un contenedor en ejecución.

- -it: Son dos opciones combinadas:
- -i: Opción "interactive". Esta opción permite que el contenedor mantenga la entrada estándar abierta, lo que te permite interactuar con el proceso que estás ejecutando (en este caso, el cliente MySQL).
- -t: Opción "tty". Esta opción asigna un pseudo-terminal (TTY) al contenedor, lo que proporciona una interfaz de terminal más rica. Esto es útil para comandos que requieren un entorno de terminal.
- mariadb: Este es el nombre del contenedor de Docker donde se está ejecutando el servidor de MariaDB. Debes asegurarte de que este nombre coincida con el nombre del contenedor que has configurado.
- mysql: Este es el comando que se va a ejecutar dentro del contenedor. En este caso, es el cliente de MySQL que permite interactuar con la base de datos de MariaDB.
- -u root: Este es un argumento para el comando `mysql`:
- -u: Especifica el nombre de usuario que utilizarás para conectarte a la base de datos. En este caso, el usuario es `root`, que es el usuario administrador de la base de datos de MariaDB.

use inception;

select * from wp_users;