Módulo Herramientas de Desarrollo

Docker

Objetivo: Aprender a crear y configurar contenedores de Docker. Saber persistir datos de un contenedor aunque éste falle.

Introducción

Docker es una herramienta de software que permite empaquetar aplicaciones y servicios en contenedores, que son entornos aislados y ligeros. Los contenedores de Docker son similares a las máquinas virtuales, pero sin los costos de inicio y mantenimiento que estas tienen.

Los contenedores de Docker permiten a los desarrolladores encapsular sus aplicaciones y servicios junto con sus dependencias en un solo paquete, lo que facilita su distribución y despliegue en diferentes entornos. Docker utiliza las características de aislamiento de recursos del sistema operativo para garantizar que los contenedores sean independientes y no interfieran entre sí.

Una de las ventajas de Docker es que los contenedores comparten el mismo kernel del sistema operativo del host, lo que los hace mucho más ligeros y eficientes que las máquinas virtuales. Además, Docker ofrece herramientas para la gestión de contenedores y la creación de imágenes de contenedores personalizadas, lo que hace que sea más fácil y repetible crear y mantener entornos de desarrollo y producción.

¿Por qué usar Docker?

- **Independencia del host**: Docker permite que las aplicaciones se ejecuten de manera consistente en diferentes entornos, independientemente del sistema operativo del host

en el que se estén ejecutando. Esto se logra gracias a que Docker crea un entorno aislado y portátil para la aplicación, que incluye todas las dependencias necesarias para su correcto funcionamiento. De esta manera, la aplicación y sus dependencias se empaquetan en un contenedor que se puede ejecutar en cualquier sistema que tenga Docker instalado.

- **Aislamiento**: Docker permite mantener diferentes versiones de una aplicación con sus librerías y dependencias aisladas del sistema operativo del host, lo que facilita la coexistencia de varias versiones de la misma aplicación en un mismo equipo o en diferentes entornos. Esto es especialmente útil en entornos de desarrollo y pruebas, donde se pueden tener varias versiones de una aplicación en un mismo equipo o en diferentes equipos sin que se produzcan conflicto
- **Seguridad**: Docker está diseñado para ser seguro, con cada contenedor aislado y con acceso controlado. Cada contenedor ejecuta en su propio entorno aislado, lo que significa que no puede acceder a otros contenedores ni al sistema operativo del host a menos que se configure específicamente para ello. Además, Docker permite agregar capas de seguridad en el proceso de construcción de cada imagen, lo que reduce el riesgo de vulnerabilidades y fallos de seguridad.
- **Control de ambientes**: Docker permite crear diferentes versiones de una aplicación para distintos entornos, cada uno con sus propias configuraciones y bases de datos. Esto hace que sea más fácil gestionar y mantener diferentes ambientes de desarrollo, prueba y producción, ya que cada uno puede tener su propia imagen de Docker, con todas las dependencias y configuraciones necesarias. De esta manera, se puede asegurar que las aplicaciones se ejecuten de manera consistente en diferentes entornos, lo que reduce el riesgo de errores y fallos en la producción.
- **Integración continua y entrega rápida**: Docker está diseñado para integrarse con herramientas de integración continua y entrega continua, lo que permite automatizar tareas tediosas de despliegue y entrega, acelerando el tiempo de entrega de nuevas versiones. Al utilizar contenedores de Docker para empaquetar las aplicaciones y sus dependencias, se puede asegurar que la aplicación se ejecute de manera consistente en diferentes entornos, lo que facilita el proceso de entrega y redunda en una mayor estabilidad. Además, Docker facilita la creación y mantenimiento de diferentes versiones de la aplicación para diferentes entornos, lo que reduce el tiempo y esfuerzo necesario para configurar y desplegar la aplicación en cada entorno.
- **Escalabilidad y flexibilidad**: Docker permite ejecutar múltiples instancias de la misma imagen de forma paralela, lo que proporciona mayor escalabilidad y flexibilidad para manejar cargas de trabajo intensas y picos de demanda. Al utilizar contenedores de Docker, se pueden aumentar o disminuir fácilmente el número de instancias de una aplicación, lo que permite ajustar el rendimiento y la capacidad en función de la demanda. Esto es especialmente útil en entornos de producción, donde se necesita una alta disponibilidad y rendimiento de la aplicación. Además, Docker facilita la creación y mantenimiento de diferentes versiones de la aplicación para diferentes entornos, lo que

reduce el tiempo y esfuerzo necesario para configurar y desplegar la aplicación en cada entorno.

Arquitectura de Docker

La arquitectura de Docker se diferencia de la arquitectura de las Máquinas Virtuales (VMs) en que Docker utiliza un enfoque de virtualización a nivel de sistema operativo, mientras que las VMs utilizan una virtualización completa de hardware.

En el caso de las VMs, cada máquina virtual requiere un sistema operativo completo, que a su vez utiliza una cantidad significativa de recursos (CPU, memoria y almacenamiento). En otras palabras, cada VM ejecuta su propio kernel de sistema operativo, lo que resulta en una sobrecarga significativa.

Por el contrario, Docker utiliza un enfoque de contenedores, donde cada contenedor comparte el mismo kernel del sistema operativo del host. Esto significa que los contenedores de Docker son más ligeros y requieren menos recursos que las VMs, ya que no necesitan un sistema operativo completo.

Además, la virtualización a nivel de sistema operativo permite a los contenedores de Docker compartir archivos y bibliotecas comunes con el host, lo que mejora la eficiencia y la escalabilidad de la solución.

¿Qué son imágenes y contenedores?

Imágenes

Son archivos sados para instanciar contenedores Docker. Cada imagen contiene instrucciones precisas para permitir la ejecución completa de una aplicación. Las imágenes están compuestas por múltiples capas, donde cada capa representa un conjunto de cambios en relación a la capa anterior. Todas las capas, excepto la última, son de solo lectura, lo que significa que no se pueden modificar después de que se hayan creado. Cada capa es una colección de cambios con respecto a la capa anterior y, en conjunto, forman una imagen completa. Las imágenes de Docker se pueden crear a partir de un archivo Dockerfile, que especifica las instrucciones necesarias para construir la imagen. Las imágenes de Docker se almacenan en un registro de imágenes, como Docker Hub, y se pueden compartir y descargar para su uso en diferentes entornos.

Contenedores

Son la unidad de software estándar para empaquetar el código de una aplicación junto con todas sus dependencias y ejecutarlo de manera rápida y confiable en cualquier ambiente. Los contenedores son instancias en ejecución de las imágenes de Docker, lo que significa que se pueden crear varios contenedores a partir de la misma imagen. Cada contenedor tiene una capa de escritura en la parte superior, donde se guardan todos los cambios realizados en el contenedor. La capa de escritura se elimina cuando el contenedor se elimina. Los cambios realizados en un contenedor no afectan a la imagen subyacente de la que se basa, lo que significa que los cambios en un contenedor no afectarán a otros contenedores o a la imagen original. Cada contenedor se ejecuta en su propio espacio aislado y funciona como una "mini" máquina virtual, lo que significa que cada contenedor tiene su propia red, almacenamiento y recursos del sistema.

Instalar y configurar en Docker

Desinstalar versiones antiguas

Antes de instalar una nueva versión de Docker, es recomendable desinstalar cualquier versión anterior que pueda estar instalada en el sistema. Para hacer esto, ejecuta el siguiente comando en una terminal:

```
sudo apt remove docker docker-engine docker.io containerd runc
```

Luego de correr este comando, podemos estar seguros de que tenemos todo lo necesario para hacer una instalación ‘limpia’. Si la consola nos reporta que ninguno de los paquetes estaba instalado, no hay problemas.

Configurar repositorio

Primero debemos actualizar el índice de paquetes e instalar algunos paquetes que nos permitirán usar un repositorio por HTTPS:

```
sudo apt-get update

sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent
```

```
software-properties-common
```

Una vez actualizado el índice y los paquetes necesarios instalados, pasamos a descargar y agregar la llave GPG de Docker:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Una vez descargada la llave, agregamos el repositorio de Docker a los repositorios de APT:

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
```

Instalar Docker

Actualizamos los repositorios y pasamos a instalar docker:

```
sudo apt update

sudo apt install docker-ce docker-ce-cli containerd.io
```

Una vez que todos los paquetes estén instalados, validamos la instalación corriendo la imagen hello-world

```
sudo docker run hello-world
```

Este comando descarga una imagen de prueba y la corre en un contenedor. Si todo sale bien, nos muestra un mensaje de éxito y se cierra.

Configurar Docker para que no necesite sudo

Como el demonio de Docker corre bajo el usuario root, es necesario usar sudo para poder ejecutar los comandos de docker. Para evitar que esto sea necesario, necesitamos agregar nuestro usuario personal al grupo ‘docker’.

```
sudo usermod -aG docker $USER
```

Podemos validar que todo funciona correctamente volviendo a ejecutar el hello-world, pero sin sudo:

```
docker run hello-world
```

Configurar Docker para que se inicie al bootear Ubuntu

Docker tiene un par de servicios que podemos activar para que se ejecuten automáticamente al iniciar la computadora. Para esto, hacemos uso del comando systemctl:

```
sudo systemctl enable docker
sudo systemctl enable containerd
```

Mapeo de puertos de un contenedor "NGINX"

¿Qué es NGINX?

NGINX es un servidor web de alta calidad y alto rendimiento, que se utiliza comúnmente como servidor de proxy, balanceador de carga y servidor de caché. Puedes encontrar más información sobre NGINX en el sitio web oficial: https://www.nginx.com/.

¿Qué es el mapeo de puertos?

El mapeo de puertos es una de las características más importantes de Docker. Permite a los contenedores de Docker exponer puertos en el host y permitir que las aplicaciones y servicios que se ejecutan en los contenedores sean accesibles desde el mundo exterior.

En Docker, cada contenedor se ejecuta en su propia red virtual. Por defecto, los contenedores no están conectados a la red externa del host, lo que significa que no pueden ser accesibles directamente desde el host o desde otros dispositivos en la red.

Para permitir que los contenedores sean accesibles desde el mundo exterior, debemos exponer los puertos que se utilizan dentro del contenedor al host. Esto se hace mediante el mapeo de puertos. El mapeo de puertos de Docker le permite asignar un puerto en el host a un puerto en

el contenedor. De esta manera, cuando se accede al puerto del host, se redirige al puerto correspondiente en el contenedor.

Por ejemplo, si un contenedor de NGINX está ejecutando una aplicación web en el puerto 80, pero el puerto 80 no está expuesto en el host, la aplicación web no será accesible desde el host o desde otros dispositivos en la red. Sin embargo, si se realiza un mapeo de puertos del puerto 80 del contenedor al puerto 8080 del host, la aplicación web será accesible desde http://localhost:8080.

Configurar un contenedor de NGINX mapeado al puerto 8080

Primero descargamos la imagen oficial de Docker de NGINX desde Docker Hub:

```
docker pull nginx
```

Podemos obtener más información sobre esta imagen en https://hub.docker.com/_/nginx.

Una vez descargada la imagen, pasamos a crear el contenedor mapeando el puerto 80 al puerto 8080 del host:

```
docker run -p 8080:80 -d nginx
```

En este comando, "-p 8080:80" mapea el puerto 80 del contenedor al puerto 8080 del host y "-d" indica que el contenedor debe ejecutarse en segundo plano.

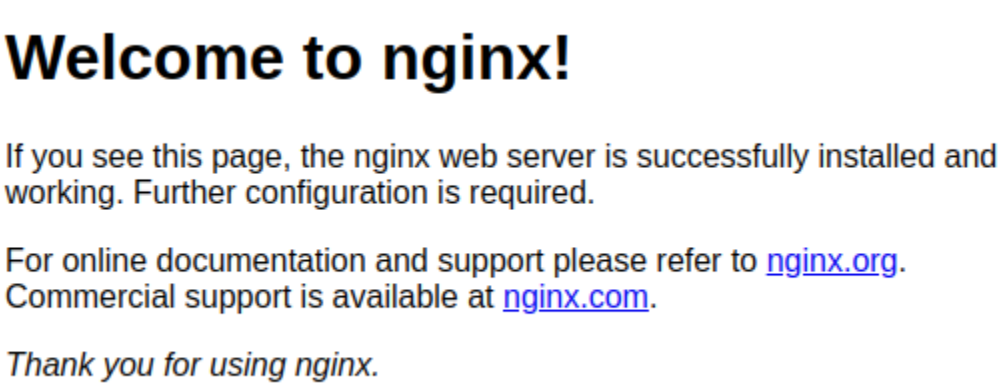
Podemos verificar que el contenedor se está ejecutando usando docker ps:

```
docker ps

CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
PORTS                                   NAMES

cadb876ab1b9   nginx     "/docker-entrypoint.…"   6 seconds ago    Up 5 seconds
0.0.0.0:8080->80/tcp, :::8080->80/tcp   serene_tesla
```

Finalmente, podemos validar que al ir a http://localhost:8080 en nuestro navegador, nos encontramos con una pantalla como esta:



Persistir datos de un contenedor:

¿Por qué es importante persistir los datos de un contenedor?

Cuando un contenedor de Docker se detiene o se elimina, todos los datos almacenados en el contenedor se pierden, incluyendo los archivos y las bases de datos. Para evitar la pérdida de datos, es importante persistir los datos del contenedor en un almacenamiento externo. De esta manera, los datos pueden sobrevivir al ciclo de vida del contenedor y pueden ser utilizados por otros contenedores y servicios.

Tipos de almacenamiento de datos en Docker

Docker ofrece varios mecanismos de almacenamiento de datos, incluyendo volúmenes, enlaces a carpetas del host y almacenamiento en la nube. Los volúmenes son la forma más común de almacenar datos de contenedores en Docker. Los volúmenes son una forma de almacenamiento persistente que se puede montar en cualquier contenedor y se puede compartir entre varios contenedores. Los volúmenes se pueden crear y administrar mediante el comando "docker volume".

Montar un volumen de Docker para persistir los datos del contenedor

Vamos a usar MySQL como ejemplo de persistencia de datos.Primero, crearemos un volumen de Docker y lo montaremos en el contenedor. De esta manera, los datos de la base de datos se almacenarán en el volumen, en lugar de en el sistema de archivos del contenedor. Podemos crear un volumen de Docker mediante el comando "docker volume create":

```
docker volume create mysql_data
```

Lo siguiente sería obtener la imagen de MySQL:

```
docker pull mysql
```

Podemos obtener más información sobre esta imagen en https://hub.docker.com/_/mysql.

A continuación, podemos ejecutar un contenedor de MySQL con las siguientes opciones:

```
docker run --name mysql -e MYSQL_ROOT_PASSWORD=root -e MYSQL_USER=usuario -e MYSQL_PASSWORD=123456 -d -p 3306:3306 -v mysql_data:/var/lib/mysql mysql
```

- **--name mysql**: Este parámetro establece el nombre del contenedor que se está creando, en este caso "mysql".
- **“-e MYSQL_ROOT_PASSWORD=root”**: Este parámetro establece la contraseña de root para el servidor de MySQL que se está iniciando en el contenedor. En este caso, la contraseña se establece en "root".
- **"-e MYSQL_USER=usuario"** crea un nuevo usuario llamado "usuario" en el servidor de MySQL dentro del contenedor.
- **"-e MYSQL_PASSWORD=123456"** establece la contraseña del usuario "usuario" con el valor "123456".
- **-d:** Este parámetro inicia el contenedor en segundo plano (en modo daemon), lo que significa que el proceso de contenedor se ejecutará en segundo plano.
- **-p 3306:3306:** Este parámetro establece el mapeo de puertos del contenedor al host. Especificamente, el puerto *3306* del contenedor se mapea al puerto 3306 del host. De esta forma, el servicio de MySQL dentro del contenedor puede ser accedido desde el host en 127.0.0.1:3306.
- **-v mysql_data:/var/lib/mysql:** Este parámetro crea un volumen llamado mysql_data y lo mapea al directorio /var/lib/mysql dentro del contenedor. Los datos de MySQL serán almacenados en este volumen, lo que asegura que los datos no se perderán si se detiene o borra el contenedor.

Para validar que todo funcione correctamente, podemos ejecutar

```
docker exec -it mysql-container mysql -uroot -proot
```

Y una vez conectado, podemos probar crear una base usando “CREATE DATABASE” de datos vacía y salimos usando “exit”:

```
mysql> CREATE DATABASE dbprueba;
mysql> exit;
```

Luego, detenemos el contenedor, lo eliminamos y lo recreamos con el mismo comando que usamos antes:

```
docker stop mysql

docker rm mysql

docker run --name mysql -e MYSQL_ROOT_PASSWORD=root -e MYSQL_USER=usuario -e MYSQL_PASSWORD=123456 -d -p 3306:3306 -v mysql_data:/var/lib/mysql mysql
```

Volvemos a conectarnos a MySQL:

```
docker exec -it mysql-container mysql -uroot -proot
```

Y finalmente, validamos que la base de datos que creamos esté presente usando “SHOW DATABASES”:

```
mysql> SHOW DATABASES;

+--------------------+
| Database           |
```

```
+--------------------+
| dbprueba           |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

Dockerfile

Un Dockerfile es un archivo de texto plano que contiene una serie de instrucciones que Docker usa para crear una imagen de contenedor. Un Dockerfile se puede usar para automatizar el proceso de construcción de una imagen de Docker, lo que hace que sea más fácil y repetible crear imágenes de contenedor personalizadas y de alta calidad.

Ejemplo

Comenzaremos creando una carpeta y creando dos archivos, un “index.html” y un “Dockerfile”:

```
mkdir ejemplodockerfile
cd ejemplodockerfile
touch index.html
touch Dockerfile
ls
Dockerfile  index.html
```

Vamos a editar el archivo “Dockerfile” con algunas instrucciones:

```
# Se inicia con una imagen base de Ubuntu
FROM ubuntu
# Se actualizan los paquetes del sistema
RUN apt-get update && apt-get upgrade -y
# Se instala NGINX
RUN apt-get install nginx -y
# Se copia el archivo index.html personalizado
COPY index.html /var/www/html/index.html
```

```
# Se exponen el puerto 80
EXPOSE 80
# Se inicia el servicio NGINX
CMD ["nginx", "-g", "daemon off;"]
```

En este ejemplo, se utiliza una imagen base de Ubuntu y se actualizan los paquetes del sistema. Luego se instala NGINX y se copia el archivo index.html personalizado en la ubicación adecuada en el sistema de archivos del contenedor. Después, se expone el puerto 80 y se inicia el servicio NGINX.

Antes de crear la imagen y el contenedor, vamos a editar el archivo index.html, para que no esté totalmente en blanco:

```
<h1>Hola mundo</h1>
```

Finalmente, creamos la imagen con el comando docker build y luego la corremos con docker run:

```
docker build -t mi-imagen .

docker run -d -p 80:80 --name ejemplo-dockerfile mi-imagen
```

Con este comando primero construimos la imagen a partir del Dockerfile en el directorio actual y la etiquetamos con el nombre "mi-imagen". Luego, con el segundo comando ejecutamos un contenedor de la imagen recién construida, exponiendo el puerto 80 del contenedor al puerto 80 del host con la opción “-p 80:80”, y lo ejecutamos en segundo plano con la opción “-p”.

En el navegador, podremos ver:

Hola mundo