2023
2024

Módulo: Despliegue de Aplicaciones Web

Unidad de Trabajo: 1-1

Introducción a Dockers

Miguel Ángel García Blanes
IES Doñana

GOBIERNO DE ESPAÑA MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
UNIÓN EUROPEA
Fondo Social Europeo
El FSE invierte en tu futuro
Programa financiado por el Ministerio de Educación y Formación Profesional y cofinanciado por el Fondo Social Europeo

Contenido de la unidad

1 ¿Qué es Dockers?

Docker es una plataforma de código abierto diseñada para simplificar la creación, el despliegue y la administración de aplicaciones en contenedores. Los **contenedores** son entornos aislados y ligeros que contienen todo lo necesario para ejecutar una aplicación, incluidas las bibliotecas, dependencias y configuraciones.

La idea detrás de Docker es crear contenedores ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues.

En lugar de depender de **máquinas virtuales** completas, que pueden ser más pesadas y lentas debido a la virtualización completa del sistema operativo, Docker utiliza la virtualización a nivel de sistema operativo para crear contenedores. Esto permite que las aplicaciones se ejecuten de manera consistente en cualquier entorno, ya sea en máquinas locales, servidores en la nube o incluso en centros de datos.

Para que podamos acceder como usuarios normales a una aplicación, dicha aplicación software necesita estar ejecutándose en una máquina. Pero, además, dependiendo del tipo de aplicación, dicho ordenador también necesita tener instaladas una serie de cosas para que la aplicación se ejecute correctamente: cierta versión de Java instalado, un servidor de aplicaciones (por ejemplo, Tomcat, que es el software que realmente estará ejecutando mi aplicación y haciendo que pueda interactuar con ella).

Docker, me permite meter en un contenedor (“una caja”, algo auto contenido, cerrado) todas aquellas cosas que mi aplicación necesita para ser ejecutada (java, Tomcat, …) y la propia aplicación. Así yo me puedo llevar ese contenedor a cualquier máquina que tenga instalado Docker y ejecutar la aplicación sin tener que hacer nada más, ni preocuparme de qué versiones de software tiene instalada esa máquina, de si tiene los elementos necesarios para que funcione mi aplicación, de si son compatibles…

Yo ejecutaré mi aplicación software desde el contenedor de Docker, y dentro de él estarán todas las librerías y cosas que necesita dicha aplicación para funcionar correctamente.

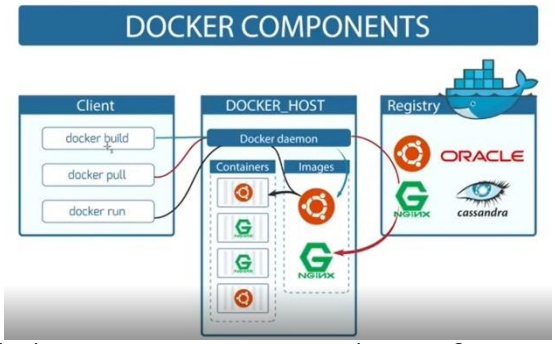
2 Arquitectura y componentes de Dockers

Docker tiene una **arquitectura cliente-servidor**, donde el cliente es un programa que se ejecuta en nuestro ordenador, y el servidor es un daemon o un servicio que se encarga de crear y administrar los contenedores.

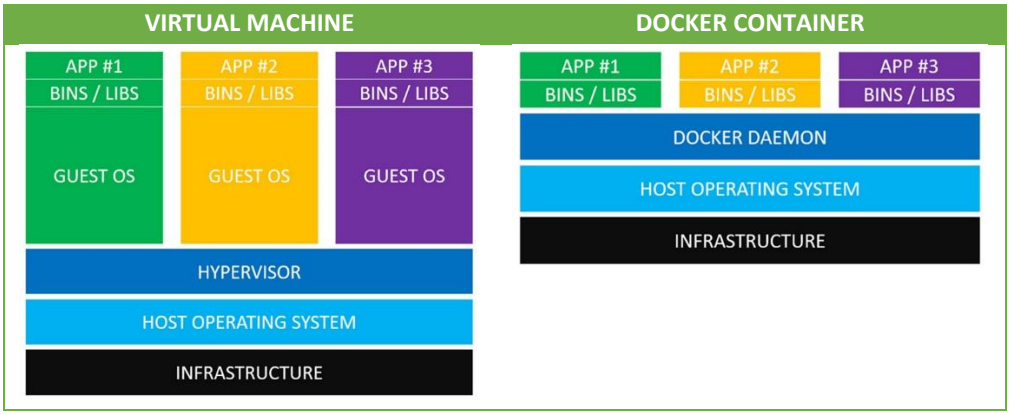
En Docker podemos distinguir varios componentes que trabajan juntos para permitir la creación, el despliegue y la administración de aplicaciones en contenedores. Entre los distintos componentes que conforman Docker, destacamos los siguientes **componentes claves**:

- **Docker Daemon (Dockerd)**: Es el componente principal de Docker que se ejecuta en el host. Es responsable de administrar los contenedores, imágenes, redes y volúmenes. El Docker Daemon escucha las solicitudes de la API de Docker y se comunica con el resto de los componentes.
- **API de Docker:** La API proporciona una interfaz para que los usuarios y las herramientas interactúen con el Docker Daemon. Permite a los usuarios ejecutar comandos y realizar operaciones en contenedores, imágenes, redes y volúmenes.
- **Docker Client:** Es la interfaz de línea de comandos (CLI) que permite a los usuarios interactuar con el Docker Daemon a través de la API de Docker. Los usuarios pueden ejecutar comandos desde el cliente para crear, administrar y supervisar contenedores e imágenes.
- **Imágenes de Docker**: Las imágenes son plantillas de sólo lectura que contienen todos los componentes necesarios para ejecutar una aplicación. Esto incluye el código, las bibliotecas, las dependencias y la configuración. Las imágenes se utilizan para crear contenedores.
- **Contenedores de Docker**: Los contenedores son instancias en ejecución de imágenes de Docker. Cada contenedor es un entorno aislado que comparte el mismo kernel del sistema operativo del host. Esto permite la portabilidad y la eficiencia, ya que los contenedores son ligeros en comparación con las máquinas virtuales.
- **Docker Registry:** Es un repositorio en el que se almacenan y se pueden compartir imágenes de Docker. Docker Hub es un ejemplo de un registro público, pero también se pueden configurar registros privados para almacenar imágenes específicas de la organización.

- **Docker Compose:** Aunque técnicamente es una herramienta separada, Docker Compose es esencial para la arquitectura de Docker. Permite definir aplicaciones multi-contenedor en un archivo YAML, especificando servicios, redes y volúmenes, y luego lanzar y administrar esos servicios como una unidad.



Los contenedores son procesos que corren sobre la máquina, esto significa que consumen muchos menos recursos que levantar una máquina virtual completa para correr una aplicación.
Por ejemplo: Si yo solo quiero un servidor apache, ¿por qué debo tener todas las herramientas que vienen con los SSOO?



3 Conceptos principales

En este apartado previo a la instalación de Docker, vamos a ver algunos concetos que tenemos que entender para poder empezar a trabajar.

3.1 Imágenes

Ya hemos visto anteriormente que una imagen Docker era una plantilla autónoma y ligera que contenía todo lo necesario para ejecutar una aplicación en un contenedor.
Las imágenes Docker están construidas utilizando una técnica llamada "**sistema de archivos en capas**". Cada capa representa un conjunto de cambios en el sistema de archivos. Esta arquitectura en capas tiene varias ventajas, como la reutilización eficiente de capas comunes entre diferentes imágenes, lo que reduce el uso de almacenamiento y acelera la construcción de imágenes.
Para le creación de una imagen en Docker, se usa un fichero especial de texto denominado **Dockerfile**. Dicho fichero define las instrucciones y configuraciones necesarias para crear una imagen Docker. Es una parte esencial del proceso de construcción de imágenes.
A continuación, se muestra una descripción más detallada de las capas de una imagen Docker:

- **Capa base (Base Layer):** es la capa inicial y fundamental sobre la que se construye la imagen. Puede ser una imagen oficial proporcionada por Docker o una imagen personalizada. Esta capa proporciona un sistema de archivos básico y un sistema operativo mínimo. Es una capa de sólo lectura.
  Ejemplos de imágenes base incluyen distribuciones de Linux como Ubuntu, CentOS o Alpine.
- **Capas intermedias (Intermediate Layers)**: Cada instrucción en el Dockerfile (por ejemplo, la copia de archivos, la instalación de paquetes, la configuración de variables de entorno) crea una nueva capa. Cada capa captura solo los cambios realizados por esa instrucción en particular. Estas capas de cambios se apilan unas sobre otras.
  Las capas intermedias son también de solo lectura y se apilan sobre la capa base y las capas anteriores
- **Capa superior (Top Layer):** Cuando se crea un contenedor a partir de una imagen, Docker agrega una capa adicional que actúa como capa de escritura. Esta capa es donde se almacenan los cambios realizados en el contenedor durante su ejecución. Esta capa de escritura es específica del contenedor y **es temporal**; desaparece una vez que el contenedor se detiene.

En un Dockerfile, se describen una serie de pasos que Docker debe seguir para construir una imagen. Cada instrucción en el Dockerfile agrega una capa adicional a la imagen resultante. Algunas de las instrucciones más comunes que se pueden incluir en un Dockerfile son:

- **FROM:** Esta instrucción define la imagen base a partir de la cual se construirá la nueva imagen. La imagen base actúa como la capa base para la construcción.
- **RUN:** La instrucción RUN se utiliza para ejecutar comandos en la capa actual. Puede ser utilizado para instalar paquetes, configurar el entorno, compilar aplicaciones y más.
- **COPY y ADD**: Estas instrucciones se utilizan para copiar archivos desde el sistema de archivos del host al sistema de archivos de la imagen. COPY se utiliza para copiar archivos locales, mientras que ADD puede hacer lo mismo además de extraer archivos comprimidos y descargar archivos de URLs.
- **WORKDIR**: Esta instrucción establece el directorio de trabajo dentro de la imagen, donde se ejecutarán los comandos RUN, CMD y ENTRYPOINT.
- **ENV:** La instrucción ENV establece variables de entorno dentro de la imagen, lo que puede ser útil para configurar variables que afectan el comportamiento de la aplicación.
- **EXPOSE:** Define los puertos en los que la imagen expondrá aplicaciones cuando se ejecute un contenedor basado en ella.
- **CMD:** Establece el comando predeterminado que se ejecutará cuando se inicie un contenedor a partir de la imagen. Solo puede haber una instrucción CMD en un Dockerfile, y si se proporciona más de una, solo se considerará la última.

Ejemplo de Dockerfile:

```
# Usar la imagen oficial de PostgreSQL como base
FROM postgres:latest

# Variables de entorno para configurar la base de datos
ENV POSTGRES_DB mydatabase
ENV POSTGRES_USER myuser
ENV POSTGRES_PASSWORD mypassword

# Expone el puerto 5432 para que se pueda acceder a la base de datos
EXPOSE 5432

# Comando de inicio para el contenedor (opcional)
CMD ["postgres"]
```

En este ejemplo:

- Estamos utilizando la imagen oficial de PostgreSQL (postgres:latest) como base.
- Establecemos variables de entorno para configurar la base de datos (nombre de la base de datos, usuario y contraseña).
- Exponemos el puerto 5432, que es el puerto por defecto de PostgreSQL.
- La última línea especifica el comando que se ejecutará cuando se inicie un contenedor basado en esta imagen. En este caso, simplemente ejecuta el proceso de PostgreSQL.

3.2 Contenedores

Los **contenedores** no son más que instancias de una imagen Docker y que tienen todo lo necesario para ejecutar nuestra aplicación, es decir, son unidades de ejecución livianas y aisladas que encapsulan aplicaciones y todos sus componentes tales como código, bibliotecas, dependencias y configuraciones.

Los contenedores contienen varios elementos:

- **Sistema de archivos encapsulado**: Cada contenedor tiene su propio sistema de archivos aislado. Este sistema de archivos se compone de capas.
- **Imágenes Docker**: Los contenedores se crean a partir de imágenes Docker.
- **Proceso en ejecución**: Cada contenedor ejecuta un proceso aislado en su espacio de ejecución. Este proceso puede ser una aplicación, un servicio o cualquier proceso definido en la imagen.
- **Entorno aislado**: Los contenedores proporcionan aislamiento en términos de recursos y redes. Esto significa que cada contenedor tiene su propio espacio de recursos y red, lo que evita conflictos entre contenedores y permite la coexistencia de múltiples aplicaciones en un mismo sistema.

- **Variables de entorno y configuraciones**: Los contenedores pueden definir variables de entorno que influyen en su comportamiento. Estas variables pueden configurarse durante la creación del contenedor o en el momento de la ejecución.
- **Puertos y redes**: Los contenedores pueden exponer puertos para que las aplicaciones sean accesibles desde fuera del contenedor. También pueden conectarse a redes virtuales para permitir la comunicación entre contenedores y con otros sistemas.
- **Volúmenes**: Los contenedores pueden utilizar volúmenes para almacenar datos persistentes que deben sobrevivir más allá del ciclo de vida del contenedor.

3.3 Volúmenes

Un **volumen en Docker** es un mecanismo que permite persistir y compartir datos entre contenedores y el sistema anfitrión. Los volúmenes son utilizados para almacenar información que debe sobrevivir más allá del ciclo de vida de un contenedor, ya que los sistemas de archivos dentro de los contenedores son efímeros y se destruyen cuando el contenedor se detiene o se elimina. Los volúmenes ofrecen una solución para manejar datos persistentes y compartirlos entre contenedores de manera eficiente.

En Docker, existen varios tipos de volúmenes que puedes utilizar para administrar y persistir datos entre contenedores y el sistema anfitrión. La elección del tipo de volumen depende de tus necesidades específicas. Cada tipo tiene sus propias ventajas y desventajas, por lo que es importante considerar los requisitos de tus aplicaciones antes de decidir qué tipo de volumen utilizar.

Más adelante nos centraremos en este concepto, por ahora lo importante es comprender qué es y para que se usa.

3.4 Redes

En Docker, hay varios **tipos de redes** que puedes utilizar para conectar y comunicar contenedores, así como para conectar contenedores con el host y con redes externas. Cada tipo de red tiene su propio propósito y casos de uso.

Algunos de los tipos de redes que podemos usar en Docker son (aunque nos centraremos en la bridge):

- **Bridge Network (Red Bridge)**: Esta **es la red predeterminada** cuando instalamos Docker. Cada contenedor en una red de puente puede comunicarse directamente con otros contenedores en la misma red. Los contenedores en una red de puente también pueden comunicarse con el host, pero no con contenedores en otras redes puente. Docker asigna direcciones IP a los contenedores en esta red de manera automática.
- **Host Network (Red Host)**: Cuando usas la red de host, el contenedor comparte la pila de red del host, lo que significa que no tiene su propia dirección IP ni su propio espacio de puertos. Esto puede ser útil para aplicaciones que requieren un alto rendimiento de red, pero es menos aislado que otros tipos de redes.
- Macvlan Network (Red Macvlan): Esta red permite que los contenedores obtengan direcciones MAC y direcciones IP en la misma red que la interfaz física del host. Esto puede ser útil si necesitas que los contenedores tengan direcciones IP y MAC reales en la red local.
- **None Network (Red Ninguna)**: En esta red, el contenedor no tiene conectividad de red externa. Puede ser útil para contenedores que solo necesitan acceder a la red interna del host.
- **Custom Network (Red Personalizada)**: Puedes crear tus propias redes personalizadas para agrupar contenedores y controlar cómo se comunican. Esto permite segmentar y aislar tus aplicaciones en diferentes redes virtuales.

Al igual que ocurría con los volúmenes, la elección del tipo de red dependerá de nuestras necesidades específicas, como la comunicación entre contenedores, la seguridad, el rendimiento y la conectividad con redes externas.