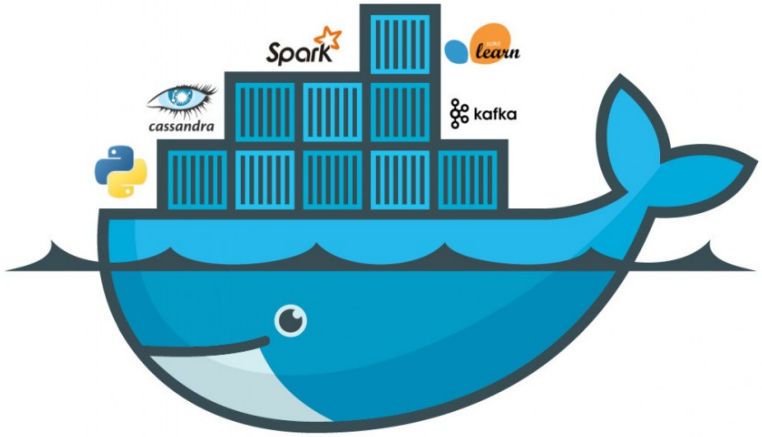
Curso Docker
Cap 1: Introducción



César Hooper Sánchez

Diciembre 2023

Un poco de historia!

- *1979 se introduce en Linux el comando chroot (change root) que permite aislar o enjaular un proceso cambiando el directorio raíz por una nueva ubicación.*
- *2000 mejora de chroot con FreeBSD Jails: sandboxing y virtualización*
- *2004: Oracle Solaris Containers: servidores virtuales.*
- *2006 Google crea Process Container (cgroups: control groups). Nueva característica para kernel de Linux.*
- *2008 aparecen los Linux Containers o LXC basados en namespaces y cgroups*
-

¿Qué es la virtualización de contenedores?

La virtualización de contenedores es una tecnología que permite empaquetar y ejecutar aplicaciones y sus dependencias, junto con parte del sistema operativo necesario, en un entorno aislado llamado contenedor. Los contenedores son unidades ligeras y portátiles que encapsulan el software y sus configuraciones, lo que facilita su implementación y ejecución de manera consistente en diferentes entornos.

A diferencia de la virtualización tradicional, donde se virtualiza todo el sistema operativo y se ejecuta en una máquina virtual, la virtualización de contenedores se centra en encapsular aplicaciones y sus dependencias específicas sin virtualizar todo el sistema operativo. Esto hace que los contenedores sean más eficientes en términos de recursos y más rápidos de implementar.

Docker es una de las plataformas de contenedores más populares que utiliza esta tecnología. Permite a los desarrolladores empaquetar una aplicación junto con todas sus dependencias y bibliotecas en un contenedor, lo que facilita su distribución y ejecución en cualquier entorno que admita Docker.

- Aislamiento: Aunque los contenedores comparten el núcleo del sistema operativo, están aislados entre sí. Cada contenedor tiene su propio espacio de usuario y recursos, lo que garantiza que las aplicaciones no interfieran entre sí y mejora la seguridad.
-

¿Qué es la virtualización de contenedores?

- **Portabilidad**: Los contenedores son portables y pueden ejecutarse de manera consistente en cualquier entorno que admita contenedores.
- **Eficiencia de recursos**: La virtualización de contenedores es más liviana que la virtualización tradicional, ya que comparte el mismo núcleo del sistema operativo subyacente y no requiere un sistema operativo completo para cada contenedor. Esto mejora la eficiencia y el rendimiento, permitiendo una implementación más rápida y un uso más eficiente de los recursos.
- **Escalabilidad**: Docker facilita la implementación y la escalabilidad de aplicaciones a través de herramientas de orquestación como Kubernetes, Docker Swarm, etc.

Comparación entre máquinas virtuales y contenedores.

- Docker (D) y las máquinas virtuales (VM) son tecnologías de virtualización, pero difieren en sus enfoques y la forma en que logran el aislamiento y la portabilidad de las aplicaciones. Aquí algunas diferencias clave entre Docker y máquinas virtuales:
- **Nivel de abstracción:**
  - D: Los contenedores comparten el núcleo del sistema operativo subyacente y son más livianos en comparación con las máquinas virtuales.
  - VM: Cada VM tiene su propio sistema operativo y utiliza recursos dedicados.
- **Consumo de recursos:**
  - D: Los contenedores comparten el mismo núcleo del sistema operativo, lo que significa que son más eficientes en el uso de recursos. Multiples contenedores
  - VM: Cada una tiene su propio sistema operativo y consume más recursos, ya que incluye duplicación de sistemas operativos, múltiples kernels y una mayor carga de administración.

Comparación entre máquinas virtuales y contenedores.

- Tiempo de inicio:
  - D : Los contenedores se inician rápidamente, en cuestión de segundos, ya que no requieren arrancar un sistema operativo completo.
  - VM : Las VMs suelen tardar más en iniciarse, ya que deben arrancar un sistema operativo completo
- Portabilidad:
  - D: Las aplicaciones en contenedores son altamente portátiles y pueden ejecutarse de manera consistente en cualquier entorno que admita Docker, desde el entorno de desarrollo local hasta la nube.
  - VM: Las VMs también son portátiles, pero debido a su mayor tamaño y complejidad, la migración entre diferentes entornos puede ser más desafiante.
- Escalabilidad:
  - D: facilita la implementación y la escalabilidad de aplicaciones a través de herramientas de orquestación como Kubernetes, Docker Swarm, etc.
  - VM: Aunque las VMs se pueden escalar, la gestión y orquestación de VMs generalmente implica más complejidad.

Comparación entre máquinas virtuales y contenedores.

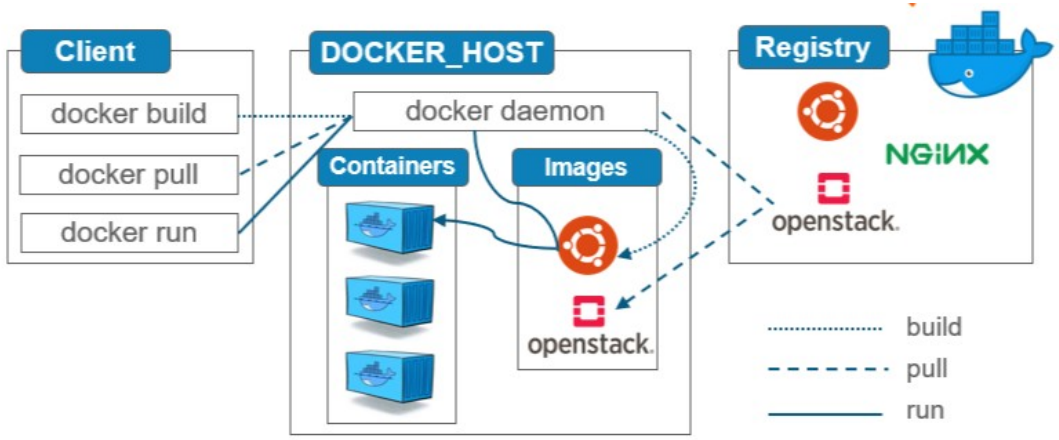
- **Aislamiento:**
  - D: No es tan fuerte como el aislamiento de las máquinas virtuales
  - VM: Ofrecen un mayor nivel de aislamiento, cada una tiene su propio SO y kernel. Importante para aplicaciones que requieren alto nivel de separación.
- **Ventajas:**
  - D: Eficiencia en uso de Recursos, Tiempo de arranque, Postabilidad, Gestión de Dependencias (creación y distribución de aplicaciones junto con todas sus dependencias), Escalabilidad.
  - VM: Aislamiento Completo, Compatibilidad con diferentes SO, Flexibilidad en la Configuración de Recursos (permiten asignar recursos específicos a cada instancia), Migración y Copias de Seguridad. Seguridad debido al mayor aislamiento.
- Docker es preferido por su eficiencia, portabilidad y facilidad de implementación, especialmente en entornos de desarrollo ágiles. Las máquinas virtuales ofrecen un mayor aislamiento y son ideales para cargas de trabajo que requieren una separación más completa y la ejecución de sistemas operativos diferentes.
- En muchos casos, estas tecnologías se utilizan de manera complementaria para aprovechar sus fortalezas respectivas.

Arquitectura de Docker: imágenes y contenedores.

- Los principales compenentes de Docker son:
- **Docker Cliente:** se encarga de gestionar las operaciones realizadas sobre contenedores, se comunica con *Docker Deamon* (demonio Docker). Es la principal via de comunicación entre Usuario y Docker.
- **Docker Deamon:** Se instala en la máquina host. Se encarga de gestionar los contenedores y los servicios requeridos
- **Índice de imágenes:** o registro de imágenes (Docker Registry) es donde se almacenan las imágenes Docker. Puede ser un repositorio público o privado.
- **Contenedor Docker:** Conjunto de carpetas y ficheros donde se define la aplicación que queremos ejecutar. Incluye sistema operativo necesario para poder ejecutarla: es el entorno de ejecución Docker
- **Dockerfiles:** Conjunto de scripts que contienen comandos comandos necesarios para automatizar la creación de imágenes.

Arquitectura de Docker: imágenes y contenedores.

- Desde cliente Docker (via Docker client o una API) se ejecutan comandos correspondientes a la creación de la imagen y los comandos que serán ejecutados en el contenedor. El deamon realiza el procesamiento e inicia creación de imagen.
- El deamon crea una imagen base (que contiene el SO) e añadirá diferentes capas que incluyen las instrucciones definidas en Dockerfile.
- Una vez creada la imagen (que puede ser local o pública) se genera el contenedor con la aplicación que queremos ejecutar.
-



Diferencias entre imagen y contenedor Docker

- Un contenedor no puede existir sin una imagen: en términos de POO el contenedor es la instancia de una imagen.
- Cada vez que realizamos una acción sobre una imagen se crea un contenedor que actuará como una nueva capa sobre la imagen. En la capa contenedor (de la imagen) se realizan operaciones y cambios, éstos no afectarán directamente a la imagen, sólo al contenedor.
- Un contenedor es como un snapshot (captura) de una imagen que contiene todos los cambios que hemos realizado al fichero principal de la imagen. Podemos tener varios contenedores en ejecución basados en la misma imagen.
- Cuando se elimina un contenedor, los cambios que se hayan realizado, también son borrados. [Si queremos que los cambios persistan sobre la imagen es necesario usar comando Docker commit que veremos más adelante. Ésto crea una nueva imagen]

Docker Deamon

- Lo dejamos para próximo capitulo

Preguntas:

- ¿Qué es Docker y cuál es su principal propósito en el desarrollo de software?
- ¿Cuál es la diferencia fundamental entre un contenedor Docker y una máquina virtual?
- ¿En qué se asemejan las máquinas virtuales y los contenedores en términos de virtualización?
- ¿Cómo describe la arquitectura básica de Docker y su papel en la ejecución de contenedores?
- ¿Cuál es el propósito principal de las imágenes en el contexto de Docker?
- ¿Cuáles son las ventajas de utilizar máquinas virtuales en comparación con Docker en ciertos escenarios?
- ¿En qué situación sería más eficiente utilizar Docker en lugar de máquinas virtuales?
- ¿Cómo facilita Docker la portabilidad de las aplicaciones en comparación con las máquinas virtuales?
- ¿Cuáles son algunas aplicaciones prácticas comunes para el uso de Docker en el desarrollo y despliegue de software?
-

- **Fin Capítulo 1**