Universidad Nacional Autónoma de México
Facultad de Ingeniería
División de Ingeniería Eléctrica

Sistemas Operativos

Docker Engine

Alumno:

- **Aguilar Martínez Erick Yair**

Fecha de entrega: **13/03/2024**

Grupo: **6**

Semestre: **2024-2**

Profesor: **Ing.** Gunnar Eyal Wolf Iszaevich

Introducción

¿Por qué Docker?

Imagina simplificar todos los problemas de configuración e implementación de aplicaciones. Eso es lo que logra Docker, una herramienta que como desarrolladores nos hace la vida más fácil. Pero, ¿qué hace a Docker tan atractivo? Su enfoque en contenedores lo cual permite encapsular aplicaciones y sus dependencias, garantizando que puedan ejecutarse de manera consistente en diversos entornos. Esto facilita tanto el desarrollo como la implementación, permitiendo a los equipos trabajar con la certeza de que el comportamiento local del proyecto será exactamente igual al de producción. Es decir, olvidaremos el típico problema que a más de uno le ha sucedido “Funcionaba en mi maquina”

¿Qué cambia con Docker?

Bueno, resulta que facilita mucho las cosas. Hace que el desarrollo sea más rápido, las actualizaciones más simples y una optimización en el uso de los recursos.

Así que, si alguna vez te preguntaste por qué todos hablan de Docker, hoy vas a tener una idea del porqué.

Evolución de la tecnología a través del tiempo

En este segmento, exploraremos la trayectoria temporal de la tecnología, destacando cómo ha evolucionado hasta llegar al estado actual. Desde sus primeros conceptos hasta las innovaciones más recientes, analizaremos cómo Docker ha madurado y se ha convertido en una pieza fundamental en el panorama tecnológico.

Principios y funcionamiento

En donde nos meternos de lleno en la forma en que opera. Te contaré cómo maneja las cosas, desde los recursos hasta la manera en que mantiene nuestras aplicaciones a salvo. Sin tecnicismos complicados, solo entenderás cómo Docker hace la magia para que puedas aprovechar al máximo.

Usos y Aplicaciones

Ya al final, exploraremos casos prácticos en los que he utilizado Docker para que puedas ver su potencial

Evolución de la tecnología a través del tiempo

chroot

"chroot" es una abreviatura de "change root" (cambiar raíz). Es un comando en sistemas operativos tipo Unix que cambia el directorio raíz para un proceso y sus hijos. Este comando crea un entorno aislado al establecer un nuevo directorio raíz, lo que significa que el proceso y sus descendientes ven solo ese directorio como su raíz, limitando su acceso al resto del sistema de archivos.

La utilidad principal de chroot es crear un entorno controlado para ejecutar procesos, útil para la instalación de sistemas operativos, pruebas de software o para la recuperación de sistemas. Cuando se ejecuta chroot, el sistema de archivos dentro del nuevo directorio se convierte en el punto de partida para el proceso, y este proceso no puede acceder a archivos fuera de ese directorio.

Chroot aísla principalmente el sistema de archivos de un proceso y sus descendientes, pero no aísla otros recursos del sistema.

Elementos que sí se aíslan:

Sistema de archivos: El directorio especificado en el comando chroot se convierte en el nuevo directorio raíz para el proceso y sus hijos. Esto significa que el proceso y sus descendientes solo pueden acceder a archivos y directorios dentro de este nuevo directorio.

Rutas relativas: Dentro del entorno chroot, las rutas relativas comienzan desde el nuevo directorio raíz, no desde la raíz real del sistema de archivos.

Acceso a bibliotecas y ejecutables: Los programas ejecutados dentro del entorno chroot solo pueden acceder a las bibliotecas y ejecutables ubicados dentro del nuevo directorio raíz.

Elementos que no se aislan:

Recursos del sistema: Chroot no aísla recursos del sistema como la memoria, la red o la CPU. Los procesos dentro de un entorno chroot todavía comparten estos recursos con otros procesos en el sistema.

Procesos externos: Chroot no afecta a procesos que se ejecutan fuera de su entorno. Otros procesos en el sistema operan normalmente y no se ven afectados por el entorno chroot.

Identificadores de proceso y usuarios: Chroot no afecta a los identificadores de proceso ni a los usuarios del sistema. Los procesos dentro del entorno chroot pueden tener los mismos identificadores de usuario que fuera del entorno.

Jails

FreeBSD Jails es una tecnología de virtualización a nivel del sistema operativo que permite crear entornos aislados dentro de un sistema FreeBSD. Cada "jail" es una instancia

independiente que ejecuta su propio conjunto de procesos y tiene su propio espacio de usuario, sistema de archivos y direcciones IP. El propósito principal de FreeBSD Jails es proporcionar un aislamiento para ejecutar aplicaciones o servicios de manera segura y sin interferencias entre distintos entornos.

¿Cómo funciona?

Separación de recursos: Jails utiliza mecanismos del kernel de FreeBSD para aislar recursos, como el sistema de archivos, procesos, direcciones IP y nombres de host.

Namespace: Cada jail tiene su propio espacio de nombres, lo que significa que los procesos dentro de un jail no son conscientes de la existencia de otros jails o del sistema host.

Chroot: FreeBSD Jails utiliza la llamada al sistema chroot para establecer un directorio raíz aislado para cada jail, lo que impide que los procesos dentro de un jail accedan a los archivos fuera de su espacio asignado.

IP aliasing: Cada jail puede tener su propia dirección IP y configuración de red, lo que permite la independencia en términos de conectividad de red.

¿Qué elementos aísla y cuáles no?

Aisla:

Sistema de archivos: Cada jail tiene su propio sistema de archivos raíz aislado mediante chroot, lo que significa que no puede acceder a archivos fuera de su espacio asignado.

Procesos: Los procesos dentro de un jail están aislados del sistema host y de otros jails. Esto permite la ejecución de aplicaciones o servicios de forma independiente.

Direcciones IP: Los jails pueden tener direcciones IP asignadas de manera independiente, lo que los aísla en términos de conectividad de red.

No Aísla completamente:

Kernel: Todos los jails comparten el mismo kernel del sistema host. Aunque se aíslan en términos de procesos y recursos, comparten el mismo núcleo del sistema operativo FreeBSD.

Linux vserver

La tecnología Linux-VServer es un concepto de partición basado en Contextos de Seguridad que permite la creación de muchos Servidores Virtuales Privados (VPS) independientes que se ejecutan simultáneamente en un solo servidor físico, compartiendo eficientemente los recursos del hardware.

Un VPS proporciona un entorno operativo casi idéntico al de un servidor Linux convencional. Todos los servicios, como ssh, correo, servidores web y de bases de datos, pueden iniciarse en un VPS con solo modificaciones mínimas, al igual que en cualquier servidor real.

Cada VPS tiene su propia base de datos de cuentas de usuario y contraseña de root, y está aislado de otros servidores virtuales, excepto por el hecho de que comparten los mismos recursos hardware.

Solaris Containers

Solaris Containers, también conocidos como Oracle Solaris Zones, es una tecnología de virtualización a nivel de sistema operativo desarrollada por Sun Microsystems (ahora propiedad de Oracle). Permite crear entornos de sistema operativo aislados, conocidos como "contenedores" o "zonas", dentro de un único sistema operativo Solaris. Cada contenedor actúa como un sistema operativo virtualizado independiente, con su propio espacio de usuarios, procesos y recursos.

2. ¿Cómo funciona Solaris Containers?

El funcionamiento de Solaris Containers se basa en el concepto de virtualización a nivel de sistema operativo. Utiliza recursos del kernel de Solaris para crear entornos aislados llamados zonas. Estas zonas comparten el mismo núcleo del sistema operativo, pero están completamente aisladas entre sí, lo que garantiza que los procesos en una zona no afecten a los de otras zonas. Cada zona tiene su propio sistema de archivos, espacio de direcciones de memoria y configuración de red.

3. ¿Qué hace y qué no hace Solaris Containers?

Lo que hace:

Aislamiento: Proporciona aislamiento entre las zonas, lo que permite ejecutar múltiples aplicaciones o servicios de forma segura en un único sistema operativo.

Eficiencia: Al compartir el mismo kernel, reduce la sobrecarga de recursos en comparación con la virtualización completa.

Gestión simplificada: Facilita la administración al permitir la creación, **clonación y migración** de zonas de manera eficiente.

Lo que no hace:

Hipervisor: No es un hipervisor como en la virtualización tradicional. Todas las zonas comparten el mismo kernel del sistema operativo host.

Compatibilidad con otros sistemas operativos: Está diseñado específicamente para entornos Solaris y no es compatible con la ejecución de sistemas operativos diferentes en cada contenedor.

OpenVZ

OpenVZ es una virtualización basada en contenedores para Linux. OpenVZ crea múltiples contenedores Linux seguros e independientes (también conocidos como VEs o VPSs) en un solo servidor físico, lo que permite una mejor utilización del servidor y garantiza que las aplicaciones no entren en conflicto. Cada contenedor funciona y se ejecuta exactamente como un servidor independiente; un contenedor se puede reiniciar de manera independiente y tiene acceso de root, usuarios, direcciones IP, memoria, procesos, archivos, aplicaciones, bibliotecas del sistema y archivos de configuración.

OpenVZ es software gratuito de código abierto, disponible bajo la licencia GNU GPL.

Proccess Containers

Los ingenieros de Google comenzaron el trabajo en esta función en 2006 bajo el nombre de "contenedores de procesos".

Un grupo de control (abreviado como cgroup) es una colección de procesos que están vinculados por los mismos criterios y asociados con un conjunto de parámetros o límites. Estos grupos pueden ser jerárquicos, lo que significa que cada grupo hereda límites de su grupo padre. El kernel proporciona acceso a múltiples controladores (también llamados subsistemas) a través de la interfaz cgroup; por ejemplo, el controlador "memory" limita el uso de memoria, "cpuacct" contabiliza el uso de la CPU, etc.

LXC

Linux Containers (LXC) es un método de virtualización a nivel del sistema operativo para ejecutar múltiples sistemas Linux aislados (contenedores) en un host de control utilizando un único kernel de Linux.

El kernel de Linux proporciona la funcionalidad de cgroups que permite limitar y priorizar recursos (CPU, memoria, E/S de bloques, red, etc.) sin necesidad de iniciar máquinas virtuales, y también la funcionalidad de aislamiento de espacio de nombres que permite el aislamiento completo de la vista del entorno operativo de una aplicación, incluidos árboles de procesos, redes, identificadores de usuario y sistemas de archivos montados.[3]

Warden

CloudFoundry inició Warden en 2011, utilizando LXC en las etapas iniciales y luego reemplazándolo con su propia implementación. Warden puede aislar entornos en cualquier sistema operativo, ejecutándose como un daemon y proporcionando una API para la gestión de contenedores. Desarrolló un modelo cliente-servidor para administrar una colección de contenedores en múltiples hosts, y Warden incluye un servicio para gestionar cgroups, espacios de nombres y el ciclo de vida de los procesos.

Docker

Docker es una plataforma de código abierto que facilita la creación, implementación y ejecución de aplicaciones en contenedores. Los contenedores son entornos ligeros y portátiles que contienen todo lo necesario para ejecutar una aplicación, incluidas las bibliotecas, dependencias y configuraciones. Docker permite empaquetar una aplicación junto con sus dependencias en un contenedor, lo que garantiza su portabilidad y consistencia en diferentes entornos.

¿Qué puede hacer Docker?

Portabilidad: Docker facilita la portabilidad de las aplicaciones. Un contenedor Docker puede ejecutarse de manera consistente en cualquier entorno que admita Docker, ya sea en desarrollo, pruebas o producción.

Eficiencia: Al utilizar contenedores, Docker aprovecha la eficiencia del sistema operativo subyacente, ya que comparte el mismo kernel. Esto permite ejecutar múltiples contenedores en un solo sistema host de manera eficiente.

Desarrollo Ágil: Docker facilita el desarrollo ágil al proporcionar entornos de desarrollo consistentes en todos los equipos. Los desarrolladores pueden crear contenedores con todas las dependencias necesarias y compartirlos con el equipo, eliminando problemas de versiones o configuraciones.

¿Qué no puede hacer Docker?

Máquinas Virtuales completas: A diferencia de las máquinas virtuales, Docker no emula un sistema operativo completo. En lugar de eso, comparte el kernel del sistema operativo subyacente, lo que hace que los contenedores sean más livianos.

Rendimiento nativo: Aunque Docker es eficiente, en algunos casos, las aplicaciones pueden experimentar un ligero costo de rendimiento debido al uso compartido del kernel del sistema operativo.

Principios y funcionamiento

Docker opera con un sistema cliente-servidor. El cliente de Docker interactúa con el daemon de Docker, encargado de las tareas fundamentales como construir, ejecutar y distribuir los contenedores de Docker. Pueden estar en el mismo sistema o conectar un cliente de Docker a un daemon remoto. La comunicación entre el cliente y el daemon se realiza a través de una API REST, utilizando sockets UNIX o una interfaz de red. Docker Compose es otro cliente que facilita trabajar con aplicaciones que constan de varios contenedores.

Sockets

Los sockets son un mecanismo de comunicación que permite la transferencia de datos entre procesos, ya sea en la misma máquina o a través de una red. Un socket es un punto final para enviar o recibir datos en una red. Se puede ver como una interfaz de programación de aplicaciones (API) que proporciona funciones para la creación, configuración y administración de conexiones de red.

Existen dos tipos principales de sockets en UNIX: los sockets de flujo (stream sockets) y los sockets de datagrama (datagram sockets). Los sockets de flujo utilizan el protocolo de transporte TCP (Transmission Control Protocol) y proporcionan una conexión confiable y orientada a la secuencia de bytes. Por otro lado, los sockets de datagrama utilizan el protocolo UDP (User Datagram Protocol) y ofrecen una comunicación no orientada a la conexión y sin garantía de entrega de los datos en orden.

Docker Demon

El daemon de Docker, conocido como dockerd, está atento a las solicitudes de la API de Docker y administra elementos de Docker como imágenes, contenedores, redes y volúmenes. Además, puede interactuar con otros daemons para supervisar servicios de Docker.

Que es un proceso Daemon?

Un proceso demonio (también conocido como "daemon" en inglés) es un tipo de programa de computadora en sistemas operativos tipo UNIX que se ejecuta en segundo plano, sin la interacción directa de un usuario y sin estar asociado a un terminal. Estos procesos son comúnmente utilizados para realizar tareas específicas en el sistema de forma continua o periódica. Algunos ejemplos de funciones realizadas por demonios incluyen la gestión de servicios, la actualización de índices, la programación de tareas, y la manipulación de eventos del sistema.

Docker Client

La herramienta principal para la interacción de muchos usuarios con Docker es el cliente de Docker (docker). Cuando emites comandos como docker run, el cliente transmite esas instrucciones al daemon de Docker (dockerd), que las lleva a cabo. El comando docker utiliza la interfaz de programación de aplicaciones (API) de Docker, y este cliente puede conectarse con más de un daemon.

Docker Registry

Un registro de Docker es como una biblioteca para almacenar imágenes de Docker. Docker Hub es una biblioteca pública que cualquiera puede usar, y por defecto, Docker busca imágenes allí. También puedes tener tu propia biblioteca privada.

Cuando usas los comandos docker pull o docker run, Docker trae las imágenes necesarias desde tu biblioteca personalizada. Si utilizas el comando docker push, estás enviando tu propia imagen a tu propia biblioteca.

Imágenes

Una imagen en Docker es como un modelo de solo lectura que contiene las instrucciones para crear un contenedor. Generalmente, se basa en otra imagen, pero con algunas personalizaciones adicionales. Por ejemplo, podrías construir una imagen sobre la base de ubuntu, pero añadirle el servidor web Apache, tu aplicación y los detalles de configuración necesarios para que funcione.

Puedes crear tus propias imágenes o simplemente usar las creadas por otros y que están en un registro. Para construir la tuya, creas un Dockerfile con una estructura simple que define los pasos para crear y ejecutar la imagen. Cada instrucción en el Dockerfile agrega una capa a la imagen. Si modificas el Dockerfile y vuelves a construir la imagen, solo se reconstruyen las capas que han cambiado. Esto es parte de lo que hace que las imágenes sean tan ligeras, pequeñas y rápidas en comparación con otras tecnologías de virtualización.

Contenedores

Un contenedor es como una versión en ejecución de una imagen. Puedes crear, iniciar, detener, mover o borrar un contenedor usando la API o la interfaz de línea de comandos (CLI) de Docker. También puedes conectar un contenedor a una o más redes, agregarle almacenamiento e incluso generar una nueva imagen basada en cómo está en ese momento.

Por defecto, un contenedor está bastante aislado de otros contenedores y de la máquina anfitriona. Puedes decidir cuánto quieres aislar la red, el almacenamiento y otros aspectos del contenedor con respecto a los demás contenedores o a la máquina principal.

La definición de un contenedor incluye la imagen que utiliza, así como cualquier configuración que le proporcionas al crearlo o iniciarlo. Si eliminas un contenedor, desaparecen todos los cambios en su estado que no estén guardados en un almacenamiento persistente.

Almacenamiento de datos

De manera predeterminada, todos los archivos creados dentro de un contenedor se guardan en una capa de contenedor que se puede escribir por lo que no se conservan cuando el contenedor deja de existir, y recuperar los datos del contenedor puede ser complicado si otro proceso los necesita.

Docker ofrece dos opciones para que los contenedores almacenen archivos en la máquina principal, asegurando que los archivos persistan incluso después de que el contenedor se detenga: **volúmenes** y montajes de enlace (**bind mounts**).

Volume

Los volúmenes en Docker son como carpetas especiales que Docker administra. Puedes crear un volumen de forma explícita con el comando docker volume create, o Docker puede crear uno automáticamente cuando creas un contenedor o servicio.

Cuando creas un volumen, se guarda dentro de una carpeta en la máquina que hospeda Docker. Al conectar este volumen a un contenedor, es como si estuvieras conectando esa carpeta al contenedor. Esto se parece a cómo funcionan los montajes de enlace, pero con la ventaja de que los volúmenes son gestionados por Docker y están separados de la funcionalidad principal de la máquina principal.

Un mismo volumen puede ser utilizado por varios contenedores al mismo tiempo. Si ningún contenedor actualmente está utilizando un volumen, este permanece disponible para Docker y no se borra automáticamente. Puedes limpiar los volúmenes no utilizados con el comando docker volume prune.

Cuando conectas un volumen, puede tener un nombre específico o ser anónimo. Los volúmenes anónimos reciben un nombre aleatorio único en el host de Docker. Tanto los volúmenes con nombre como los anónimos persisten aunque elimines el contenedor que los utiliza, a menos que uses una opción especial al crear el contenedor, en cuyo caso se eliminará el volumen anónimo. Si creas varios contenedores uno tras otro que usan volúmenes anónimos, cada contenedor tiene su propio volumen. Estos no se comparten automáticamente entre contenedores, pero puedes hacerlo manualmente si montas el volumen anónimo utilizando su identificador único.

Además, los volúmenes en Docker admiten el uso de controladores, lo que te permite almacenar tus datos en máquinas remotas o proveedores de servicios en la nube, entre otras opciones.

Binds

Los montajes de enlace tienen funcionalidad limitada en comparación con los volúmenes. Cuando usas un montaje de enlace, un archivo o directorio en la máquina anfitriona se monta en un contenedor. El archivo o directorio se referencia por su ruta completa en la máquina anfitriona. No es necesario que el archivo o directorio exista previamente en el host de Docker; se crea según sea necesario. Los montajes de enlace son rápidos, pero dependen de que el sistema de archivos de la máquina anfitriona tenga una estructura de directorios específica disponible.

Docker Compose

Docker Compose es una herramienta para definir y ejecutar aplicaciones de múltiples contenedores. Es la clave para lograr una experiencia de desarrollo y implementación eficiente y simplificada.

Compose facilita el control de toda la pila de aplicaciones, permitiendo gestionar servicios, redes y volúmenes en un único y comprensible archivo de configuración YAML. Luego, con un solo comando, puedes crear y poner en marcha todos los servicios según tu archivo de configuración.

Fuentes

Contenedores: Un poco de historia. (s. f.). SG Buzz. https://sg.com.mx/revista/52/contenedores-un-poco-historia

Pelado Nerd. (2021, 23 marzo). *La historia de los contenedores - Desde 1979 a 2021* [Vídeo]. YouTube. https://www.youtube.com/watch?v=K0nHZlHNfQ4

Tabakman, T. (2024, 14 febrero). *A Brief History of Containers: From the 1970s Till Now*. Aqua. https://www.aquasec.com/blog/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016/

Rajhi, S. (2023b, septiembre 28). What is chroot Linux sys call and How to Control It: Sandbox with Seccomp. *Medium*. https://medium.com/@seifeddinerajhi/what-is-chroot-linux-sys-call-and-how-to-control-it-sandbox-with-seccomp-b9b9a2bedfa4

Solaris containers. (s. f.). https://www.oracle.com/solaris/technologies/solaris-containers.html