Docker

Carranza Ochoa, José David
Ríos Lira, Gamaliel

16 de marzo de 2023

1. Introducción

Actualmente, los sistemas de información se tienen que implementar bajo ciertas arquitecturas un tanto complejas, con la finalidad de aumentar su escalabilidad. Para llevar a cabo lo anterior, muchas veces se tiene que hacer uso de múltiples subsistemas, los cuales se despliegan de forma independiente. Lógicamente, es complicado que cada subsistema que se implementa, se despliegue en un servidor físico distinto debido al costo de los mismos[1].

Es importante mantener cada subsistema aislado de los demás ya que cada uno de ellos podría tener algunos requerimientos de configuración específicos e incluso correr sobre diferentes sistemas operativos. Este trabajo aborda una investigación acerca de la herramienta *Docker* con la finalidad de exponer sus características fundamentales y su importancia dentro de las áreas de tecnologías de la información actualmente. Lo anterior bajo un marco de referencia orientado a la materia de Sistemas Operativos.

2. Desarrollo

2.1. Máquinas virtuales

El concepto de máquina virtual (VM por sus siglas en inglés) se trata de la implementación de arquitecturas de hardware real a través de software. Visto desde otro punto de vista, se puede decir que se trata de implementar el comportamiento de una máquina totalmente diferente (*guest*) a través de otra máquina (*host*). Por lo general, para la administración de las máquinas virtuales se requiere de un programa especializado denominado *hypervisor*.

El problema que traen consigo las máquinas virtuales es que agregan mucha carga de trabajo al

[1] Y más aún cuando se habla de soluciones basadas en la nube, en la que muchas veces no sabemos con exactitud en qué lugar físico se están desplegando las aplicaciones

host debido a la capa de abstracción que requiere el *hypervisor*. Con esto, su ejecución podría llegar a tornarse lenta.

2.2. Contenedores

Los contenedores son una aproximación para resolver las desventajas que traen consigo las máquinas virtuales. Son abstracciones en la capa de la aplicación que empaquetan el código y las dependencias juntas. La idea general es hacer paquetes de software que incluyan todos los elementos necesarios para ejecutar las aplicaciones en cualquier momento, pero de tal forma que al momento de la ejecución únicamente haya un sistema operativo en ejecución (el del *host*). Esto se logra haciendo que el núcleo, también concido como *kernel*, se comparta entre todos los conenedores. Este es el principio de funcionamiento a través del cual funciona *Docker*.

La utilización de contenedores trae consigo las siguientes ventajas:

- **Separación de responsabilidades:** Los el equipo de desarrollo se enfoca en la lógica y el equipo de soporte se encarga de desplegar y nadie se preocupa por las configuraciones de las aplicaciones.
- **Portabilidad:** Los contenedores se pueden ejecutar prácticamente en cualquier lugar. Desde servidores físicos, servidores en la nube e incluso dentro de máquinas virtuales.
- **Aislamiento:** Los contenedores virtualizan los recursos de CPU, memoria, almacenamiento y red a nivel de sistema operativo (de forma lógica).

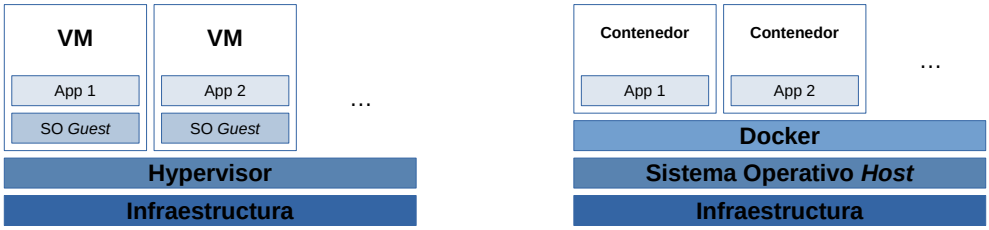
Adicionalmente, una de las ventajes de los contenedores es que ocupan mucho menos espacio en memoria que las máquinas virtuales (las imágenes son típicamente de unos cuantos MB de tamaño), pueden manejar más aplicaciones y requieren de la utilización de menos sistemas operativos.

2.3. Qué es Docker

Docker es una plataforma para desarrolladores y administradores de sistemas que sirve para desarrollar, desplegar y ejecutar aplicaciones con contenedores. De forma general, a través de *Docker* y las interfaces que proporciona, se pueden crear, modificar, borrar e interactuar internamente con los contenedores de una aplicación. El uso de contenedores de Linux para desplegar aplicaciones es denominado contenerización (del inglés *containerization*). Particularmente, la utilización de contenedores no es algo nuevo; sin embargo, su uso para desplegar aplicaciones de forma sencilla sí lo es.

En la Figura 1 se puede apreciar una comparación entre el funcionamiento de una máquina virtual contra el funcionamiento de *Docker*. Es posible apreciar la gran carga de trabajo que trae consigo la utilización de diferentes sistemas operativos para ejecutar diferentes aplicaciones, lo cual

evidentemente utiliza más recursos de hardware. Por otra parte, **Docker** únicamente utiliza un sistema operativo y a través de las capas de abstracción que proporciona, trabajando en conjunto con las funcionalidades de Linux, es posible desplegar contenedores (que únicamente traen consigo configuraciones a nivel de aplicación).



(a) Funcionamiento de una máquina virtual (b) Funcionamiento de contenedores Docker

Figura 1: VM vs. Docker

Además de todas las características mencionadas anteriormente acerca de los contenedores, los contendores de *Docker* tiene las siguientes características específicas:

- Es un **proceso** —o grupo de procesos— de Linux.
- Está **aislado** del sistema operativo *host* que lo aloja.
- Tiene una cantidad de **recursos limitada**.
- Cuenta con un **sistema de archivos independiente** del sistema operativo *host*.

Para lograr que cada uno de los contenedores tenga las características descritas anteriormente, *Docker* necesita hacer uso de algunas funciones del kernel de Linux, como los **grupos de control** y los **namespaces**.

2.3.1. Espacios de Nombres (namespaces)

Tal como se sabe, una de las mayores características que debe proporcionar un sistema operativo es el **aislamiento**. Para el ámbito en cuestión, este concepto hace referencia a que los procesos en ejecución no deben preocuparse por los demás procesos que también están siendo ejecutados. El sistema debe funcionar —idealmente— como si fuera el único proceso en ejecución.

Con la finalidad de lograr este cometido, *Docker* está diseñado de tal forma que hace uso de los *namespaces* del kernel. La idea de los *namespaces* es envolver los recursos globales del sistema en una abstracción que haga parecer que los procesos del mismo *namespace* tienen su propia instancia de dicho recurso global. De esta forma, se obtiene un aislamiento entre procesos. La

idea principal de *Docker* reside justo en este punto, ya que a través de los *namespaces* se logra crear los contenedores como "subsistemas" aislados de todos los demás procesos que ejecuta el sistema operativo. Cuando se ejecuta un contenedor, *Docker* crea un conjunto de namespaces para ese contenedor en específico.

Algunos de los *namespaces* de Linux que *Docker* utiliza internamente son los siguientes:

- **PID.** Permite que los procesos que se encuentran en diferentes *PID namespaces* puedan tener el mismo identificador de proceso (PID).
- **NET.** Permite el aislamiento de recursos relacionados con redes: números de puerto, dispositivos de red, reglas de *firewall*, etc.
- **MNT.** Permite el aislamiento de unidades montadas para procesos en cada instancia del *namespace*. Cada proceso de una instancia verá distintas jerarquías de directorios.
- **UTS.** Provee aislamiento de dos identificadores del sistema: el *hostname* y el nombre de dominio NIS.

2.3.2. Grupos de Control (cgroups)

Sumado a lo anterior —y tal como ya se mencionó anteriormente—, *Docker* también hace uso de los **grupos de control** (*cgroups*) del kernel. Estos le permiten compartir recursos de hardware disponibles con los contenedores, aplicando límites y restricciones. De forma breve, los *cgroups* son un mecanismo para controlar los recursos del sistema. Cuando un *cgroup* está activo, puede controlar la cantidad de CPU, RAM, tiempo u otros recursos que un proceso pueda requerir.

De forma interna, algunos de los grupos de control que *Docker* utiliza son:

- **Memory cgroup.** Limita el uso de memoria para procesos y genera reportes de los recursos de memoria utilizados por cada proceso.
- **CPU cgroup.** Límites de utilización de la CPU para un proceso. Evita que un proceso consuma mucho tiempo de CPU, evitando que esto afecte negativamente al desempeño del sistema.
- **Devices cgroup.** Permite o restringe el acceso a dispositivos par aun proceso.

2.3.3. Sistema de archivos

Cada contenedor, como ya se explicó anteriormente, necesita tener un sistema de archivos aislado que aparente ser "único" para el proceso que esté ejecutando. Esto no significa que sea totalmente independiente del sistema operativo *host*; más bien, hace referencia a que el proceso supondrá que el sistema de archivos al que tiene acceso es el único disponible y no sabrá que internamente no es más que una sección del sistema de archivos del sistema *host*, al cual evidentemente no tendrá acceso —al menos no directamente. Esta tarea se lleva a cabo a través de una herramienta disponible

para los sistemas operativos basados en Unix —tal como el caso de Linux— llamada *chroot*. Esta herramienta hace posible invocar un proceso, cambiando para este y sus hijos el directorio raíz del sistema. Esto no es más que otra forma de aislamiento que se agrega al proceso en ejecución por el contenedor.

2.4. Desventajas

Durante la investigación elaborada, se encontró un dato muy importante de destacar acerca de la forma en la que trabaja *Docker*. Aún cuando se trata de una herramienta sumamente madura y ampliamente aceptada en la industria —ya que por mucho tiempo tuvo soporte de *RedHat*—, su implementación cuenta con un detalle que los expertos en el ámbito de la seguridad informática siempre han criticado. La ejecución del *Docker Deamon* —una de las herramientas que trae integrada internamente— necesita tener privilegios de *root*. Con esto, la posibilidad de que un contenedor se libere de su "contexto aislado" y ejecute código en el sistema *host* siempre está presente. Esto no es un detalle menor y aún con el paso del tiempo nunca ha podido ser corregido.

Lo anterior ha orillado a la creación de nuevas plataformas similares a *Docker*. Tal es el caso del proyecto *Podman*, en donde se está desarrollando una alternativa sumamente similar y compatible con *Docker*, en la cual se corrige de forma definitiva el detalle antes mencionado.

Finalmente, la herramienta *DockerHub*, un complemento de *Docker* a través de donde se obtienen imágenes de contenedores, ha sido vulnerada a través de la publicación de imágenes maliciosas que algunas veces ejecutan procesos por detrás con la finalidad de obtener beneficios. No pasa esto con todas las imágenes, la mayor parte de este problema se concentra en imágenes **no oficiales**, publicadas por personas aisladas que no pertenecen a algún tipo de organización. Aún cuando esto es una desventaja, se puede contrarrestar haciendo uso de imágenes **oficiales**.

3. Conclusiones

Tal como se expuso anteriormente, *Docker* es una herramienta que hace uso de una solución innovadora para para la gestión de contenedores. La parte interna de su funcionamiento está arraigada complemente a la forma en la que trabaja el kernel del sistema operativo, más específicamente el kernel de Linux —*namespaces*, *cgroups*, procesos, etc. El tema resulta ser de amplia importancia para la materia de Sistemas Operativos, más aún cuando son patrones de soluciones que se utilizan actualmente en la industria. A pesar de esto, no todo es perfecto y en el caso de *Docker*, tiene algunas limitantes que no son menores. Nos parece que *Docker* marcó un antes y un despues en los aspectos relacionados con el uso de contenedores para desplegar aplicaciones, a tal punto que está siendo la referencia de partida para tecnologías emergentes.

4. Bibliografía

[1] I. Amazon Web Services, *¿Qué es Docker?* 2023. dirección: https://aws.amazon.com/es/docker/.

[2] E. Davis, "Researchers Find over 1,600 Malicious Images on Docker Hub," dic. de 2022. dirección: https://howtofix.guide/1600-images-in-docker-hub/.

[3] *Get Started, Part 1: Orientation and setup*, ago. de 2019. dirección: https://docs.docker.com.xy2401.com/v17.12/get-started/.

[4] IBM, *Docker*. dirección: https://www.ibm.com/mx-es/topics/docker.

[5] M. Kerrisk, *mount_namespaces(7)*, dic. de 2022. dirección: https://man7.org/linux/man-pages/man7/mount_namespaces.7.html.

[6] M. Kerrisk, *namespaces(7)*, dic. de 2022. dirección: https://man7.org/linux/man-pages/man7/namespaces.7.html.

[7] M. Kerrisk, *network_namespaces(7)*, dic. de 2022. dirección: https://man7.org/linux/man-pages/man7/network_namespaces.7.html.

[8] M. Kerrisk, *pid_namespaces(7)*, dic. de 2022. dirección: https://man7.org/linux/man-pages/man7/pid_namespaces.7.html.

[9] M. Kerrisk, *uts_namespaces(7)*, dic. de 2022. dirección: https://man7.org/linux/man-pages/man7/uts_namespaces.7.html.

[10] Nishanil, *¿Qué es Docker?* Nov. de 2022. dirección: https://learn.microsoft.com/es-es/dotnet/architecture/microservices/container-docker-introduction/docker-defined.

[11] Oracle, *¿Qué es Docker?* 2023. dirección: https://www.oracle.com/mx/cloud/cloud-native/container-registry/what-is-docker/.

[12] S. Ovens, *The 7 most used Linux namespaces*, ene. de 2023. dirección: https://www.redhat.com/sysadmin/7-linux-namespaces.

[13] *¿Qué es Docker?* Dirección: https://www.oracle.com/mx/cloud/cloud-native/container-registry/what-is-docker/.

[14] *¿Qué es Docker y cómo funciona? Ventajas de los contenedores Docker*. dirección: https://www.redhat.com/es/topics/containers/what-is-docker.

[15] *¿Qué son los contenedores?* Dirección: https://cloud.google.com/learn/what-are-containers?hl=es.

[16] E. Zepeda, *¿Cómo Funciona un Container de Docker Internamente?* Mar. de 2023. dirección: https://coffeebytes.dev/container-de-docker-con-namespaces-y-cgroups/.