ÍNDICE

PREFACIO

En estos tiempos tan cambiantes en el mundo de la tecnología, hay muchos productos que nacen en el mercado muy prometedores pero que desaparecen tan rápido como han aparecido.

En el mundo de la virtualización han surgido durante los últimos años diferentes soluciones que han servido para ahorrar costes a las empresas y mejorar sus prestaciones. Pero ante las nuevas necesidades, principalmente por el denominado *Big Data*, una solución ha aparecido con fuerza con el apoyo de las empresas más importantes en el ámbito tecnológico: **Docker**.

Docker se ha convertido en uno de los productos más solicitados y admirados de la actualidad. Este libro cubre todo lo necesario, desde la instalación hasta la administración más avanzada, todo acompañado con una gran variedad de ejemplos.

SOBRE ESTE LIBRO

Este libro está dirigido a todas las personas que quieran iniciarse en **Docker** o a aquella que tenga actualmente conocimientos y quiera profundizar en las tareas a realizar. No es necesario tener un dominio de un sistema operativo particular, pero se recomienda conocer la consola de **Linux** para utilizar los clientes de **Docker** de una forma más fluida.

La instalación se explica tanto para sistemas **Linux**, **Windows** y **OSX**. Aunque se recomienda el uso de una distribución reciente de **Linux**; el uso de los otros sistemas operativos son válidos para seguir el temario de este libro.

Una vez finalizada la lectura de este libro, al ser una guía práctica, el lector estará preparado para la administración de **Docker** y el despliegue de aplicaciones a través de *contenedores*.

Al ser un libro práctico, todos los comandos y las salidas que genera se insertan en un recuadro con el siguiente formato:

```
# docker --version
Docker version 1.13.0, build 49bf474
```

SOBRE EL AUTOR

Alberto González trabaja actualmente como *Cloud Consultant* en la empresa *Red Hat*. Anteriormente trabajó como *Administrador de Sistemas Senior* en la empresa *IBM.* En sus 15 años de experiencia ha centrado sus conocimientos en tecnologías principalmente de código abierto, centrándose en **Linux** y virtualización. Posee conocimientos de programación y base de datos.

Además, es profesor en una plataforma en línea ofreciendo cursos de calidad y prácticos de diferentes tecnologías (mayoritariamente de código abierto). La página web de sus cursos es https://www.oforte.net.

AGRADECIMIENTOS

El mayor agradecimiento es a mi sobrina Leire, que hace que quiera ser mejor persona cada día y ser un ejemplo para ella.

También mi agradecimiento a mi buen amigo Jonathan Veites Penedo, por acompañarme y apoyarme en la mayoría de los proyectos que he iniciado, e igualmente a todos mis amigos de la infancia, que a pesar de tantos años alejados siguen estando allí como si no hubiese distancia. Gracias a mis familiares por sentirse orgullosos de mis progresos. Gracias a Míriam Pérez por ayudarme a revisar el libro.

Por último, agradezco a todas las empresas y compañeros con los que he trabajado en estos 15 años, donde he crecido como persona y como profesional.

1 INTRODUCCIÓN

Docker se ha convertido en uno de los proyectos más populares en la actualidad. Grandes empresas tecnológicas han apoyado este proyecto en los últimos años, ayudando a su desarrollo y a su evolución. Empresas como *Red Hat*, *Google*, *IBM* o *Microsoft* no solo han colaborado económicamente, sino que también han proporcionado código y soporte para solucionar determinados errores.

Docker en sí no es una tecnología, sino la forma de acceder a ella. Dicha tecnología se conoce con el nombre de contenedores y fue adoptada por el núcleo de Linux recientemente. La finalidad de Docker es facilitar la creación y manipulación de los contenedores.

La tecnología de contenedores no es algo nuevo, es una forma histórica de intentar aislar recursos tanto a nivel de usuario como a nivel de aplicación. En *Linux*, el primer acercamiento previo a los contenedores se realizó a través de la operación "*chroot*", también conocida como "jaula"; que consistía en aislar aplicaciones y usuarios entre sí, con la limitación de no poder aislar recursos físicos (memoria, procesador o dispositivos). Las primeras implementaciones de "*chroot*" datan de principios de los años 80.

A partir del año 2000, aparecen las nuevas implementaciones para aislar los recursos, ya no solo para *Linux* sino para sistemas *UNIX* (Sistemas *BSD*, *Solaris*, *AIX*) y *Windows*.

Virtuozzo fue la empresa pionera en desarrollar un software para el aislamiento de recursos a nivel de sistema operativo. Su aplicación era de código cerrado, es decir, no disponible al público general. La aplicación, llamada también *Virtuozzo,* era capaz de aislar los recursos entre usuarios, limitaba el acceso a los dispositivos y separaba el acceso de las aplicaciones entre ellas. En el año 2005 la empresa lanzó, esta vez como

código abierto, el software *OpenVZ*. Este software todavía es muy popular siendo utilizado por pequeñas y grandes empresas, sobre todo aquellas que ofrecen *VPS* (servidores privados virtualizados) a bajo precio.

Entre los años 2000 y 2005, en sistemas UNIX, la tecnología avanzó en el ámbito del aislamiento de recursos. *FreeBSD* (un sistema operativo *UNIX*) implementó "*FreeBSD jail*" similar a "*chroot*" pero logrando aislar recursos. En 2001, aparece *Linux-VServer* como alternativa gratuita a *Virtuozzo* siendo considerado uno de los predecesores de los contenedores actuales. En el año 2004 aparece una de las mejores tecnologías para aislar recursos: las zonas de *Solaris*. Las zonas son consideradas una de las mejores implementaciones del aislamiento de recursos hasta la aparición de **Docker**. Las zonas de *Solaris* lograron reducir costes a empresas y reducir la infraestructura física de empresas que utilizaban servidores *Sun*.

Hasta la aparición de **Docker** en el año 2013, caben destacar las distintas implementaciones de aislamiento de recursos (conocidas como contenedores):

- *Workload partitions* (*WPARs*) para el sistema operativo de *IBM AIX*, en el año 2007.
- *HP-UX Containers* para el sistema operativo *HP-UX*, en el año 2007.
- Linux containers (*LXC*), en el año 2008.

En los primeros años de vida de **Docker** (2013 y 2014), este utilizaba *LXC* para el uso de contenedores. A partir de la versión 0.9, empezó a utilizar su propia librería (*libcontainer*) para la manipulación de los contenedores.

Contenedores vs virtualización

Una de las primeras dudas que surgen con Docker es saber diferenciar claramente las ventajas y desventajas que existen entre el uso de contenedores y la virtualización.

Antes de introducirnos en el contexto de las ventajas y desventajas, debemos revisar qué es la virtualización y los diferentes tipos que existen para evitar la confusión con los contenedores.

La virtualización consiste en añadir una capa de abstracción a los recursos físicos con el objetivo de mejorar el uso de los recursos del sistema. En el pasado, cada ele-

mento físico ejecutaba un recurso. Con la introducción de la virtualización es posible crear varios entornos simulados (máquinas virtuales) para diversos recursos. Es decir, en el caso de un servidor, gracias a la virtualización, podemos ejecutar varios sistemas dentro del mismo con diferentes entornos. Por ejemplo, un servidor puede ejecutar múltiples máquinas virtuales con diferentes sistemas operativos para diversos propósitos.

Los diferentes tipos de virtualización:

- **Virtualización completa**: la máquina virtual no tiene acceso directo a los recursos físicos y requiere de una capa superior para acceder a ellos.

Algunos ejemplos:

- VirtualBox
- QEMU
- Hyper-V
- VMware ESXi

- **Virtualización asistida por *hardware***: es el *hardware* el que facilita la creación de la máquina virtual y controla su estado. Algunos ejemplos:

- KVM
- Xen
- VMWare fusión

- **Virtualización a nivel de sistema operativo**: aquí incluimos los contenedores. Es el sistema operativo, y no el *hardware,* el encargado de aislar los recursos y proporcionar las herramientas para crear, manipular o controlar el estado de los contenedores (término utilizado en lugar de máquina virtual).

Para entender con más detalle cómo funciona **Docker**, le dedicaremos el último capítulo del libro para explicar cómo se integra con el sistema operativo y los "espacios de nombre" para aislar los recursos.

Docker al rescate

Hasta ahora hemos visto de forma resumida la historia de los contenedores y las ventajas que presenta frente a la virtualización tradicional. Pero aún no hemos respondido al porqué **Docker** es tan popular tanto para administradores de sistema como para desarrolladores.

La virtualización fue un soplo de aire fresco para la tecnología en términos de reducción de costes e infraestructura. Además, agilizó el trabajo en equipo entre los administradores de sistema y los desarrolladores. Un nuevo puesto de trabajo apareció con fuerza: la persona encargada de los entornos virtuales, ya sea *Xen* o *VMWare*, liberando la carga a los administradores de sistema que a partir de ese momento dejaban de encargarse de la parte física de los servidores.

El administrador de virtualización proporcionaba una máquina virtual con los requisitos previamente establecidos y el administrador de sistema era el encargado de instalar y configurar el sistema operativo. Una vez finalizada su tarea, por ejemplo instalar una base de datos o un servidor web, daba acceso a los desarrolladores para que pudieran desplegar sus aplicaciones.

Esta solución resultó válida durante varios años; sin embargo, en el ámbito de la tecnología que avanza a una velocidad vertiginosa, empezaron a surgir diversos problemas:

- Los desarrolladores aún estaban limitados: sus entornos estaban administrados por terceras personas.

- Los administradores de sistemas no controlaban los servidores: los desarrolladores manipulaban los sistemas y era difícil cumplir todos sus requisitos.

- Los administradores de virtualización se quejaban de los administradores de sistemas y de los desarrolladores: cada vez solicitaban cosas más inusuales.

Esto se debía a que las nuevas tecnologías requieren de entornos dinámicos, que sean fáciles de crear, destruir y transferir entre diferentes plataformas. Las aplicaciones ya no debían regirse a un tipo de *hardware,* a un sistema operativo o incluso a una versión de una distribución específica. Era necesaria la creación de una manera de programar más ágil sin tener en cuenta qué había por debajo de la aplicación.

Y de repente aparecieron los contenedores. Aunque habíamos comentado anteriormente que los contenedores llevaban tiempo en el mercado, **Docker** apareció en el momento preciso convirtiéndose en la solución del futuro.

Este nuevo proyecto solventó los siguientes problemas:

- Los desarrolladores ya no requerían de los administradores de sistemas o de terceras personas. Solo necesitaban un sistema operativo para ejecutar **Docker,** el cual era el encargado de lanzar un entorno virtual con una distribución requerida por el desarrollador, sin importar el sistema operativo o la distribución del sistema padre.

- Los administradores de sistemas ya solo requerían solicitar un sistema (físico o virtual) con unas prestaciones no demasiado grandes donde instalar y ejecutar los contenedores **Docker**. El uso de memoria y procesador de la nueva solución lograba reducir los problemas que surgían de la virtualización estándar.

- Los administradores de virtualización ya no necesitaban tanto esfuerzo o tanta dedicación a todas las peticiones que venían indirectamente de los desarrolladores. Ahora, solo era necesario mantener un sistema con unas prestaciones moderadas. Cabe destacar que en muchos casos Docker forma parte de sistemas físicos y no virtuales, requiriendo en algunos casos menos personal para administrar los entornos físicos.

Como observamos, **Docker** ha solventado, al menos temporalmente, muchos de los problemas que habían surgido con las nuevas tecnologías. En cambio no ha solucionado un asunto: el trabajo entre administradores de sistemas y desarrolladores. Con tal motivo, surge una nueva posición laboral: los *DevOps*.

Seguramente habréis notado que en los portales de búsqueda de empleo han empezado a solicitar más y más *DevOps*. ¿Pero qué es esta posición? ¿Y por qué hay otra llamada *SysOps*? Resumamos estas posiciones:

- *SysOps*: son los clásicos administradores de sistemas. Se encargan del sistema operativo y de la monitorización de los servidores. Normalmente son los encargados de instalar software, configurar el sistema y mantener las aplicaciones ejecutándose.

- DevOps: este nuevo rol consiste en un puente entre los desarrolladores y los administradores de sistema. Su fuerte es la administración de sistemas pero tienen un amplio conocimiento en los entornos de desarrollo: saben cómo funcionan los controles de versión, la integración entre diversos entornos, el desarrollo continuo y

sobre todo, solventan las peticiones de los desarrolladores sin necesidad de un procedimiento largo.

Como observamos, **Docker** se ha posicionado en el mercado de una forma muy clara. Los desarrolladores pueden lanzar los entornos necesarios, desarrollar sus aplicaciones y desplegarlas dentro de un contenedor, el cual se convertirá en una imagen para su uso en diversos entornos.

Serán los *DevOps* o los *SysOps* los encargados de ejecutar esa imagen (con todo lo necesario para ejecutar la aplicación) en los diversos entornos: desde el entorno de pruebas hasta el entorno de producción. Y todo eso sin preocuparse si el sistema contiene las versiones correctas de librerías o la distribución apropiada.

El futuro

En el mundo de la tecnología, es prácticamente imposible predecir el futuro. Ya no solo es difícil saber qué pasará, sino también conocer si ciertas tecnologías o *software* van a sobrevivir en un entorno tan cambiante y con tanta demanda de nuevos requisitos. Solo hay que ver empresas como *Heroku*, que surgió con gran popularidad para solventar los problemas de los desarrolladores y que ha perdido fuerza con la aparición de **Docker**.

El presente y el futuro próximo están ligados a los *microservicios*: aplicaciones que se ejecutan sin necesidad de alojar datos en disco, facilitando el despliegue de nuevos servidores para soportar la demanda de carga y ahorrar costes. Con los *microservicios* es posible tener una infraestructura dinámica: la carga de usuarios o de operaciones marcará el número de servidores necesarios.

Como en toda innovación, siempre hay críticos y personas que ponen en duda diversas soluciones. **Docker** no ha sido una excepción y ha recibido las siguientes acusaciones:

- **No está preparado para producción**: esta es la crítica que surge desde diferentes sectores, sobre todo de los administradores de sistemas y administradores de virtualización. Cabe decir que **Docker** ha evolucionado mucho, como es lógico debido al apoyo de grandes empresas, en la estabilidad y en la madurez del producto. En las primeras versiones era habitual un bajo rendimiento en aplicaciones con gran uso de entrada y salida, tanto en disco como en procesador.

- **No hay alta disponibilidad**: otra de las críticas que surgen, quizá relacionada con la anterior, es que no fue inicialmente diseñado para la alta disponibilidad. Nuevas soluciones aparecieron para solventarlo, las más populares: *CoreOS* y *Kubernetes.* En la actualidad, **Docker** ofrece un componente llamado *Swarn.*

- **Falta de seguridad**: una de la demandas más comunes es la posibilidad de aislar los contenedores entre sí a nivel de red. Las nuevas versiones de Docker permiten crear redes virtuales para aislar contenedores entre ellos. Además, la seguridad de las imágenes estuvo en entredicho hasta la introducción de un control de veracidad llamada *digest*.

- **Inestabilidad en los "*drivers*"**: la diversidad de *drivers* utilizados por Docker para sus sistemas de ficheros ha sido la gran odisea a solventar, ya que cada uno de ellos contiene ventajas y desventajas. Lo veremos con más calma en el capítulo 8 de Almacenamiento: *Drivers*.

Contenido de un contenedor

Un contenedor está compuesto de todo lo necesario para ejecutar una o varias aplicaciones. Su contenido es el siguiente:

- Librerías del sistema operativo: como comentábamos anteriormente, dentro de un contenedor tenemos todo lo necesario de una distribución para aislarlo del sistema que ejecuta el servicio de **Docker**. Un ejemplo son las librerías para SSL cuya finalidad es que el servidor web pueda aceptar conexiones seguras.

- Herramientas del sistema: existe una gran diversidad de herramientas dentro de cada contenedor. Algunos ejemplos:

  - Editores de texto (VI, Emacs)

  - Monitorización (Nagios, Check-MK)

  - Herramientas de registros (Elasticsearch, Splunk)

- *Runtime*: es el software que se necesita para ejecutar la aplicación dentro del contenedor. Puede ser:

  - Lenguajes interpretados: el intérprete para el código de fuente, como puede ser PHP, Perl o Python.

- Máquina virtual Java
- El programa compilado y autoejecutable

- En el caso de lenguajes interpretados, el código fuente (por ejemplo, ficheros .php, .py o .pl) o en el caso de máquina virtual los ficheros que contienen la aplicación (por ejemplo .jar o .war)

Dentro del contenedor tendremos las utilidades, la distribución que hemos elegido y utilizando los gestores de paquete (por ejemplo, APT o YUM) podremos instalar software de los repositorios de la misma.

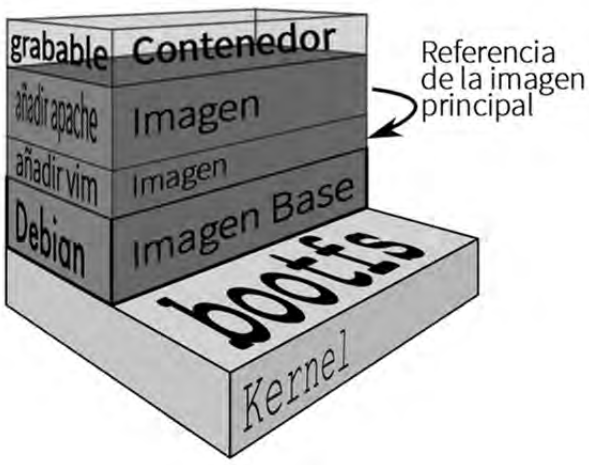
Un contenedor puede ejecutar varias aplicaciones. No obstante, se recomienda separarlas en varios contenedores debido a que estos son muy ligeros. De esta manera, las tareas de mantenimiento serán más fáciles. Algunos ejemplos:

- Monitorizar las aplicaciones individualmente monitorizando el uso del contenedor.
- Limitar los recursos de las aplicaciones.
- Reinicio de aplicaciones (ya que no afectamos a las demás).
- Copia de seguridad y restauración.

Imágenes

Antes de profundizar en **Docker**, es necesario definir un nuevo concepto: las imágenes. Hasta ahora hemos visto qué es un contenedor y sus ventajas. Pero seguramente haya surgido ya la pregunta: ¿De dónde surge el contenido de los contenedores? Y la respuesta es: de una imagen.

Una imagen contiene distintas capas de datos (la distribución, diferente software, librerías y la personalización).



Como vemos en esta representación, tenemos una imagen base, que contiene la distribución Debian y a la que se le ha añadido diferentes capas:

- Instalación del editor Emacs
- Instalación del servidor web Apache

Una vez ejecutado un contenedor basándonos en esta imagen, tendremos instalados el editor y el servidor web y no tendremos que hacerlo manualmente.

Todo contenedor puede ser convertido en una imagen empleando las utilidades de Docker. Estas imágenes pueden ser transferidas a otro servidor para ejecutar un contenedor basado en ellas y pueden hacerse copias de seguridad. Más adelante, veremos cómo utilizar imágenes dentro del repositorio público de Docker y cómo alojar nuestras propias imágenes en el mismo.

En Docker, las imágenes contienen un historial y un control de versiones, lo que facilita listar los cambios y poder restaurar la anterior.

Componentes

Gracias a su popularidad, Docker ha progresado y se ha desarrollado en numerosas áreas, por lo que actualmente, ya no solo consiste en un software para manejar contenedores. Se ha diversificado en los siguientes términos:

- **Docker Engine**: es el componente principal que sirve para trabajar con contenedores e imágenes. Como parte de este componente, podemos indicar tres componentes relacionados:

  - **Docker para Linux**: instalar Docker en un ordenador en el cual Linux ya está instalado.

  - **Docker para Windows**:: una aplicación nativa que incluye todas las herramientas necesarias para ejecutar Docker.

  - **Docker para Mac::** una aplicación nativa que utiliza el "sandbox" de macOS para proporcionar todas las herramientas necesarias de Docker.

- **Docker Hub**:: un servicio alojado en la nube para administrar y almacenar imágenes de Docker.

- **Docker Cloud**:: otro servicio alojado en la nube para crear, probar y desplegar imágenes en servidores de su propiedad.

- **Docker compose**:: define aplicaciones utilizando varios contenedores.