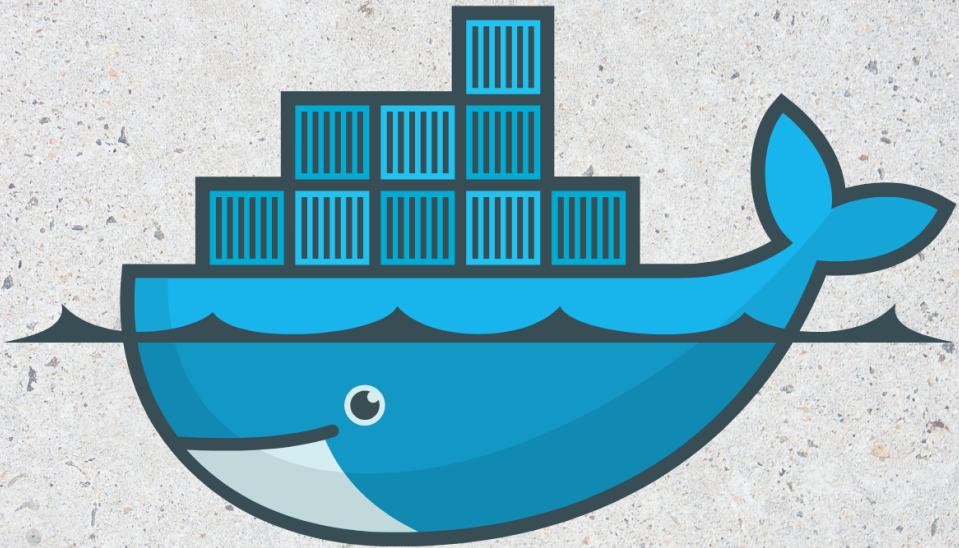


# **DOCKER**



**JOAN AMENGUAL**

# RESUMEN

En este libro se van a explicar los fundamentos esenciales de Docker, es decir, la tecnología de contenedores. Dichos conocimientos se han profundizado y se han explicado detalladamente en el curso desarrollado en Udemy.

En el siguiente enlace aparece un **descuento** del 90% para el curso Docker:

<https://blockstellart.com/30-2/docker/>

El objetivo de este curso es profundizar sobre el significado de Docker y la necesidad que existe para que dicha tecnología se esté usando tantísimo.

Te suenan empresas como: Google, Amazon, Microsoft, Red Hat? Pues estás fueron pioneras en la integración de Docker en el mundo tecnológico. Como puedes ver, Docker es usado por las más conocidas y exitosas empresas del mundo.

¡Vamos a ello!

# ÍNDICE

¿Qué es Docker?	4
Arquitectura de Docker	7
Docker vs Máquinas Virtuales (VMs)	13
Docker y contenedores	20
Docker: La Historia	30
Microservicios y monolitos	38
Docker: CI/CD	41
Docker: DEVOPS	49
Ventajas de los contenedores Docker	54
REFERENCIAS	58
SOBRE EL AUTOR	59

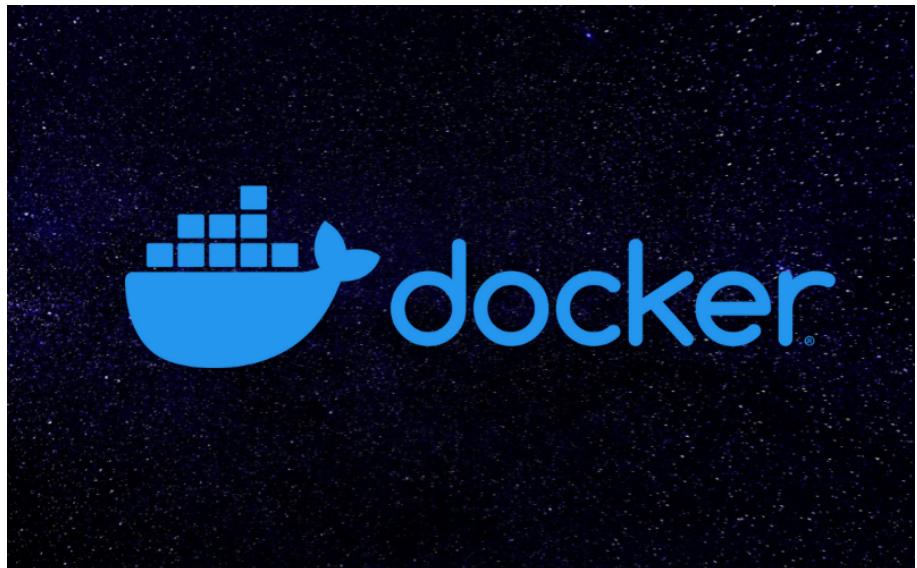
# ¿QUÉ ES DOCKER?



Uno de los puntos importantes del 'Curso de Docker de cero a experto' es el siguiente. Si estás empezando con el mundo de los contenedores seguro que una de las preguntas que te asalta es ¿Qué es Docker?. Pues Docker nace como un conjunto de herramientas que facilitan la gestión de contenedores y acaba convirtiéndose en una empresa que ofrece múltiples soluciones alrededor de los contenedores, tanto para la comunidad como soluciones enfocadas a empresas. Es por ello que podemos hablar de Docker como empresa o Docker como solución para la gestión de contenedores.

Centrándonos en Docker como gestor de contenedores, y echamos la vista atrás, teníamos que las utilidades que existían para gestionar contenedores eran complejas. Es por ello que aparece Docker. Docker es una tecnología

opensource que nos permite gestionar contenedores de una forma sencilla. Docker crea un conjunto de tools para que se pueda extender y generalizar el uso de contenedores.



Así, a través de Docker podremos crear, desplegar y ejecutar aplicaciones mediante el uso de los contenedores. Es decir, Docker proporciona una serie de herramientas que nos permiten manejar el ciclo de vida de los contenedores.

Docker permite a los desarrolladores y operadores empaquetar aplicaciones dentro de los contenedores. E integrar este proceso en los pipelines de CI/CD. Obteniendo una cadena unificada desde el desarrollo de aplicaciones hasta su puesta en producción.

Esto nos permite ir a un modelo de único despliegue de las aplicaciones. Lo cual reduce los tiempos de configuración de los entornos, los tiempos de despliegue, etc. Mejorando, en ese sentido, el tiempo de puesta en producción de las aplicaciones.

La idea principal es que las aplicaciones se empaquetan con todas las funcionalidades que necesitan para ser ejecutadas. De esta manera independientemente del entorno en el que ejecutemos el contenedor (local, desarrollo o

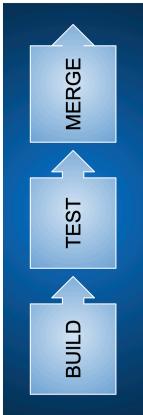
CONTINUOUS  
DEPLOYMENT (CD)

CONTINUOUS  
DELIVERY (CD)

CONTINUOUS INTEGRATION (CI)

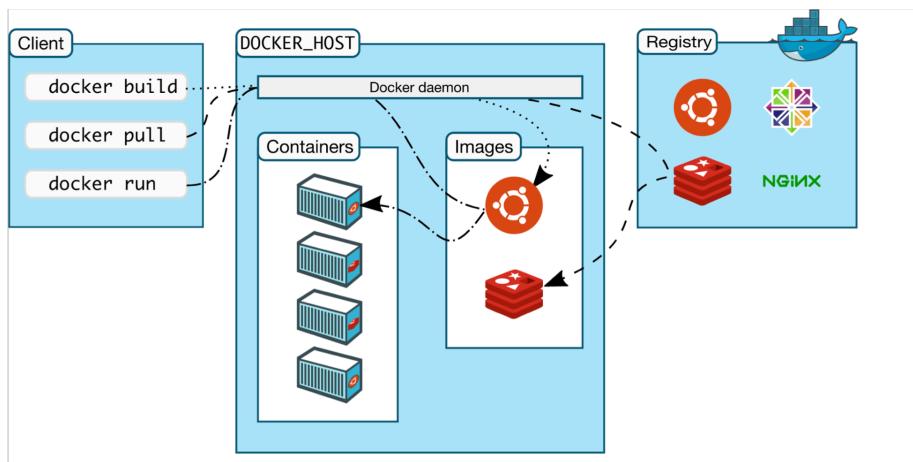
DESPLIEGUE  
AUTOMÁTICO HACIA  
PRODUCCIÓN

LIBERACIÓN  
AUTOMÁTICA AL  
REPOSITORIO



entornos productivos) siempre nos vamos a asegurar que se ejecuta de la misma manera.

## ARQUITECTURA DE DOCKER



Docker utiliza una arquitectura cliente-servidor. El cliente Docker habla con el demonio Docker, que hace el trabajo pesado de construir, ejecutar y distribuir sus contenedores Docker. El cliente y el demonio Docker pueden ejecutarse en el mismo

sistema, o se puede conectar un cliente Docker a un demonio Docker remoto. El cliente Docker y el demonio se comunican utilizando una API REST, a través de sockets UNIX o una interfaz de red. Otro cliente Docker es Docker Compose, que permite trabajar con aplicaciones formadas por un conjunto de contenedores.

## **El demonio Docker**

---

El demonio Docker (`dockerd`) escucha las peticiones de la API Docker y gestiona los objetos Docker como imágenes, contenedores, redes y volúmenes. Un demonio también puede comunicarse con otros demonios para gestionar los servicios de Docker.

## **El cliente Docker**

---

El cliente Docker (`docker`) es la forma principal en que muchos usuarios de Docker interactúan con Docker. Cuando se utilizan comandos como `docker`

run, el cliente envía estos comandos a dockerd, que los lleva a cabo. El comando docker utiliza la API de Docker. El cliente Docker puede comunicarse con más de un demonio.

## Registros Docker

---

Un registro Docker almacena imágenes Docker. Docker Hub es un registro público que cualquiera puede utilizar, y Docker está configurado para buscar imágenes en Docker Hub por defecto. Incluso puedes ejecutar tu propio registro privado.

Cuando usas los comandos docker pull o docker run, las imágenes requeridas se extraen de tu registro configurado. Cuando usas el comando docker push, tu imagen es empujada a tu registro configurado.

## Objetos Docker

---

Cuando usas Docker, estás creando y usando imágenes, contenedores, redes, volúmenes, plugins y otros objetos. Esta sección es un breve resumen de algunos de esos objetos.

### Imágenes

---

Una imagen es una plantilla de sólo lectura con instrucciones para crear un contenedor Docker. A menudo, una imagen se basa en otra imagen, con alguna personalización adicional. Por ejemplo, puedes construir una imagen que esté basada en la imagen de ubuntu, pero que instale el servidor web Apache y tu aplicación, así como los detalles de configuración necesarios para que tu aplicación se ejecute.

Puedes crear tus propias imágenes o puedes utilizar sólo las creadas por otros y publicadas en un

registro. Para construir tu propia imagen, creas un Dockerfile con una sintaxis simple para definir los pasos necesarios para crear la imagen y ejecutarla. Cada instrucción en un Dockerfile crea una capa en la imagen. Cuando cambias el Dockerfile y reconstruyes la imagen, sólo 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 una instancia ejecutable de una imagen. Puedes crear, iniciar, detener, mover o eliminar un contenedor utilizando la API o la CLI de Docker. Puedes conectar un contenedor a una o más redes, adjuntarle almacenamiento o incluso crear una nueva imagen basada en su estado actual.

Por defecto, un contenedor está relativamente bien aislado de otros contenedores y de su máquina anfitriona. Se puede controlar el grado de aislamiento de la red, el almacenamiento u otros

subsistemas subyacentes de un contenedor con respecto a otros contenedores o a la máquina anfitriona.

Un contenedor está definido por su imagen, así como por las opciones de configuración que le proporcionas cuando lo creas o lo inicias. Cuando se elimina un contenedor, cualquier cambio en su estado que no esté almacenado en el almacenamiento persistente desaparece.

# DOCKER VS MÁQUINAS VIRTUALES (VMS)



Uno de los puntos importantes del 'Curso de Docker de cero a experto' es el siguiente. Los contenedores están cambiando fundamentalmente la forma de desarrollar, distribuir y ejecutar el software.



Los desarrolladores pueden crear software localmente, sabiendo que se ejecutará de forma idéntica independientemente del entorno del host, ya sea un rack en el departamento de IT, el portátil de un usuario o un clúster en la nube. Los ingenieros

de operaciones pueden concentrarse en las redes, los recursos y el tiempo de funcionamiento y pasar menos tiempo configurando entornos y luchando contra las dependencias del sistema. El uso y la adopción de contenedores está aumentando a un ritmo fenomenal en todo el sector, desde las empresas más pequeñas hasta las de mayor tamaño. Los desarrolladores y los ingenieros de operaciones deberían esperar utilizar regularmente contenedores de alguna manera en los próximos años.

Los contenedores son una encapsulación de una aplicación con sus dependencias. A primera vista, parecen ser sólo una forma ligera de máquinas virtuales (VM) -como una VM, un contenedor contiene una instancia aislada de un sistema operativo (OS), que podemos utilizar para ejecutar aplicaciones.

Sin embargo, los contenedores tienen varias ventajas que permiten casos de uso que son difíciles o imposibles con las VM tradicionales:

- Los contenedores comparten recursos con el sistema operativo anfitrión, lo que los hace un orden de magnitud más eficiente. Los

contenedores pueden iniciarse y detenerse en una fracción de segundo. Las aplicaciones que se ejecutan en los contenedores tienen poca o ninguna sobrecarga en comparación con las aplicaciones que se ejecutan de forma nativa en el sistema operativo anfitrión.

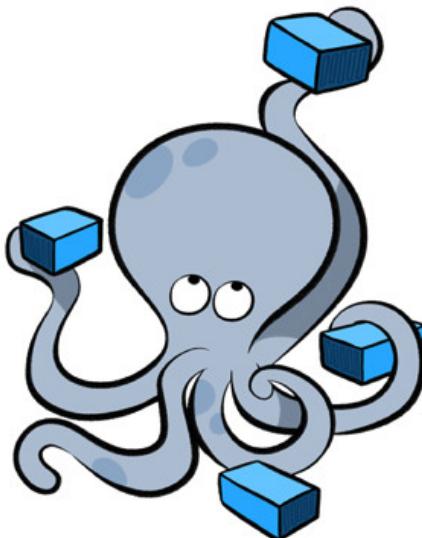
- La portabilidad de los contenedores tiene el potencial de eliminar toda una clase de errores causados por cambios sutiles en el entorno de ejecución; incluso podría poner fin al viejo estribillo de los desarrolladores de "¡pero si funciona en mi máquina!".

- La naturaleza ligera de los contenedores significa que los desarrolladores pueden ejecutar docenas de contenedores al mismo tiempo, haciendo posible emular un sistema distribuido listo para la producción. Los ingenieros de operaciones pueden ejecutar muchos más contenedores en una sola máquina anfitriona que utilizando únicamente máquinas virtuales.

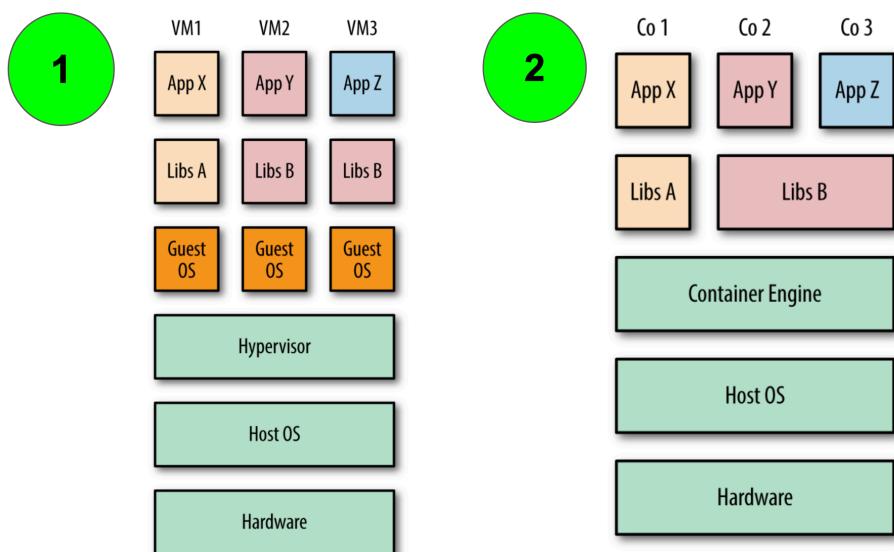
- Los contenedores también tienen ventajas para los usuarios finales y los desarrolladores fuera de la implementación en la nube. Los usuarios

pueden descargar y ejecutar aplicaciones complejas sin tener que dedicar horas a cuestiones de configuración e instalación ni preocuparse por los cambios necesarios en su sistema. A su vez, los desarrolladores de dichas aplicaciones pueden evitar preocuparse por las diferencias en los entornos de los usuarios y la disponibilidad de las dependencias.

Y lo que es más importante, los objetivos fundamentales de las máquinas virtuales y los contenedores son diferentes: el propósito de una máquina virtual es emular completamente un entorno ajeno, mientras que el propósito de un contenedor es hacer que las aplicaciones sean portátiles y autónomas.



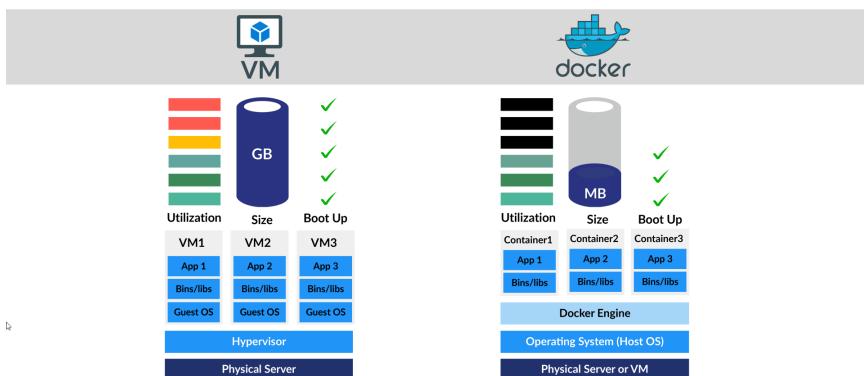
Aunque los contenedores y las máquinas virtuales parecen similares a primera vista, hay algunas diferencias importantes, que son más fáciles de explicar utilizando diagramas.



La Figura 1 muestra tres aplicaciones que se ejecutan en máquinas virtuales separadas en un host. El hipervisor es necesario para crear y ejecutar las VMs, controlando el acceso al SO y al hardware subyacente, así como interpretando las llamadas al sistema cuando es necesario. Cada VM requiere una copia completa del SO, de la aplicación que se ejecuta y de cualquier librería de apoyo.

Por el contrario, la Figura 2 de la presentación muestra cómo podrían ejecutarse las mismas tres aplicaciones en un sistema containerizado. A diferencia de las máquinas virtuales, el kernel del host se comparte con los contenedores en ejecución. Esto significa que los contenedores siempre están limitados a ejecutar el mismo kernel que el host. Las aplicaciones Y y Z utilizan las mismas bibliotecas y pueden compartir estos datos en lugar de tener copias redundantes. El motor del contenedor es responsable de iniciar y detener los contenedores de manera similar al hipervisor en una VM. Sin embargo, los procesos que se ejecutan dentro de los contenedores son equivalentes a los procesos nativos en el host y no incurren en los gastos generales asociados a la ejecución del hipervisor.

Tanto las VM como los contenedores pueden utilizarse para aislar las aplicaciones de otras aplicaciones que se ejecutan en el mismo host. Las máquinas virtuales tienen un grado añadido de aislamiento del hipervisor y son una tecnología de confianza y reforzada. Los contenedores son relativamente nuevos, y muchas organizaciones dudan en confiar completamente en las características de aislamiento de los contenedores antes de que tengan un historial probado. Por este motivo, es habitual encontrar sistemas híbridos con contenedores que se ejecutan dentro de máquinas virtuales para aprovechar las ventajas de ambas tecnologías.



# DOCKER Y CONTENEDORES



## Docker y contenedores

Uno de los puntos importantes del 'Curso de Docker de cero a experto' es el siguiente. Los contenedores son un concepto antiguo. Durante décadas, los sistemas UNIX han tenido el comando chroot que proporciona una forma simple de aislamiento del sistema de archivos. Desde 1998, FreeBSD ha tenido la utilidad jail, que extendía el sandboxing de chroot a los procesos. Solaris Zones ofrecía una tecnología de contenedor comparativamente completa alrededor de 2001, pero estaba limitada al sistema operativo Solaris.

También en 2001, Parrallels Inc, (entonces SWsoft) lanzó la tecnología comercial de contenedores Virtuozzo para Linux y más tarde abrió el núcleo de la tecnología como OpenVZ en 2005.3 Luego Google comenzó el desarrollo de CGroups

para el kernel de Linux y empezó a mover su infraestructura hacia los contenedores. El proyecto Linux Containers (LXC) se inició en 2008 y reunió a CGroups, los espacios de nombres del kernel y la tecnología chroot (entre otros) para ofrecer una solución completa de contenedores. Finalmente, en 2013, Docker aportó las últimas piezas al rompecabezas de la contenedorización, y la tecnología comenzó a entrar en la corriente principal.

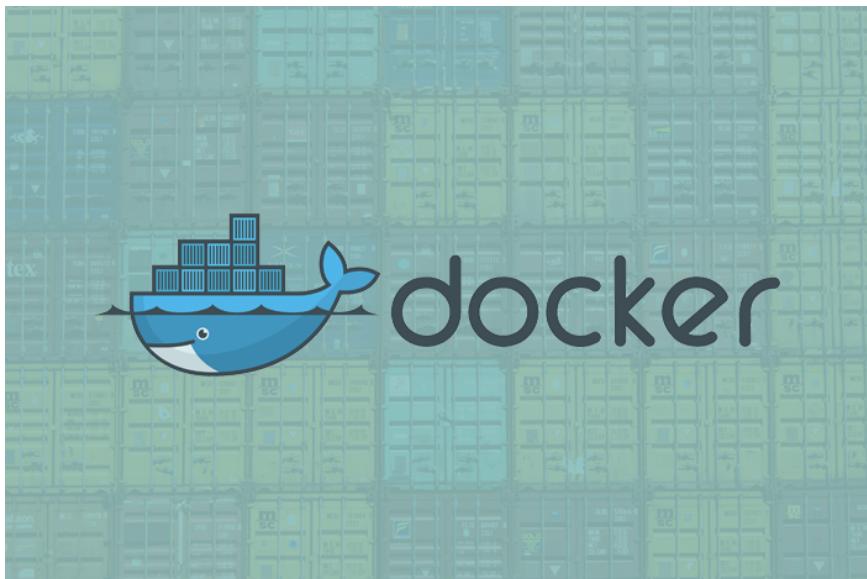


Docker tomó la tecnología de contenedores de Linux existente y la envolvió y amplió de varias maneras -principalmente a través de imágenes portátiles y una interfaz fácil de usar- para crear una solución completa para la creación y distribución de contenedores. La plataforma Docker tiene dos componentes distintos: el motor Docker, que se encarga de crear y ejecutar contenedores, y el Docker Hub, un servicio en la nube para distribuir contenedores.

El motor Docker proporciona una interfaz rápida y cómoda para ejecutar contenedores. Antes, la ejecución de un contenedor mediante una tecnología como LXC requería importantes conocimientos especializados y trabajo manual. El Docker Hub ofrece un enorme número de imágenes públicas de contenedores para su descarga, lo que permite a los usuarios empezar rápidamente y evitar la duplicación del trabajo ya realizado por otros. Otras herramientas desarrolladas por Docker son Swarm, un gestor de clústeres; Kitematic, una interfaz gráfica de usuario para trabajar con contenedores; y

Machine, una utilidad de línea de comandos para aprovisionar hosts Docker.

Gracias a la apertura del motor Docker, Docker pudo crear una gran comunidad en torno a Docker y aprovechar la ayuda del público para corregir errores y realizar mejoras. El rápido auge de Docker hizo que se convirtiera en un estándar de facto, lo que llevó a la industria a presionar para desarrollar estándares formales independientes para el tiempo de ejecución y el formato de los contenedores. En 2015, esto culminó con la creación de la Open Container Initiative, una "estructura de gobierno" patrocinada por Docker, Microsoft, CoreOS y muchas otras organizaciones importantes, cuya misión es desarrollar dicho estándar. El formato de contenedor



y el tiempo de ejecución de Docker constituyen la base del esfuerzo.

La adopción de los contenedores ha sido impulsada en gran medida por los desarrolladores, que por primera vez han recibido las herramientas para utilizar los contenedores de forma eficaz. El rápido tiempo de puesta en marcha de los contenedores Docker es esencial para los desarrolladores que anhelan ciclos de desarrollo rápidos e iterativos en los que puedan ver rápidamente los resultados de los cambios en el código. Las garantías de portabilidad y aislamiento de los contenedores facilitan la colaboración con otros desarrolladores y operaciones; los desarrolladores pueden estar seguros de que su código funcionará en todos los entornos, y las operaciones pueden centrarse en alojar y orquestar los contenedores en lugar de preocuparse por el código que se ejecuta en ellos.

Los cambios introducidos por Docker están cambiando significativamente la forma en que desarrollamos el software. Sin Docker, los contenedores habrían permanecido en las sombras de las TI durante mucho tiempo.

## La metáfora del transporte

Uno de los puntos importantes del 'Curso de Docker de cero a experto' es el siguiente. La filosofía de Docker se explica a menudo en términos de la metáfora del contenedor de transporte, que presumiblemente explica el nombre de Docker. La historia suele ser algo así:

Cuando se transportan mercancías, éstas tienen que pasar por una serie de medios diferentes, que pueden incluir camiones, carretillas elevadoras, grúas, trenes y barcos. Estos medios tienen que ser capaces de manejar una gran variedad de mercancías de diferentes tamaños y con diferentes requisitos (por ejemplo, sacos de café, bidones de productos químicos peligrosos, cajas de productos electrónicos, flotas de coches de lujo y estantes de cordero refrigerado). Históricamente, este era un proceso complicado y costoso, que requería mano de obra, como los trabajadores de los muelles, para cargar y descargar los artículos a mano en cada punto de tránsito (Figura).



La industria del transporte se revolucionó con la introducción del contenedor intermodal. Estos contenedores vienen en tamaños estándar y están diseñados para ser trasladados entre modos de transporte con un mínimo de trabajo manual. Toda la maquinaria de transporte está diseñada para manejar estos contenedores, desde las carretillas elevadoras y las grúas hasta los camiones, trenes y barcos. Existen contenedores refrigerados y aislados

para el transporte de mercancías sensibles a la temperatura, como alimentos y productos farmacéuticos.



Las ventajas de la estandarización se extienden también a otros sistemas de apoyo, como el etiquetado y el sellado de los contenedores. Esto significa que la industria del transporte puede dejar que los productores de mercancías se preocupen por el contenido de los contenedores para poder centrarse en el movimiento y el almacenamiento de los propios contenedores.

El objetivo de Docker es llevar las ventajas de la estandarización de los contenedores a la informática. En los últimos años, los sistemas de software han explotado en términos de diversidad. Atrás quedaron los días de una pila LAMP<sup>4</sup> que se ejecutaba en una sola máquina. Un sistema moderno típico puede incluir frameworks de Javascript, bases de datos NoSQL, colas de mensajes, APIs REST y backends, todo ello escrito en una variedad de lenguajes de programación. Esta pila tiene que ejecutarse parcial o totalmente sobre una variedad de hardware, desde el portátil del desarrollador y el clúster de pruebas interno hasta el proveedor de la nube de producción. Cada uno de estos entornos es diferente, y ejecuta diferentes sistemas operativos con diferentes versiones de bibliotecas en diferentes hardware.

En resumen, nos encontramos con un problema similar al de la industria del transporte: tenemos que invertir continuamente un esfuerzo manual considerable para mover el código entre entornos. Al igual que los contenedores intermodales

simplificaron el transporte de mercancías, los contenedores Docker simplifican el transporte de aplicaciones de software.

Los desarrolladores pueden concentrarse en crear la aplicación y enviarla a través de las pruebas y la producción sin preocuparse por las diferencias en el entorno y las dependencias. Las operaciones pueden centrarse en las cuestiones fundamentales de la ejecución de los contenedores, como la asignación de recursos, el inicio y la detención de los contenedores y su migración entre servidores.



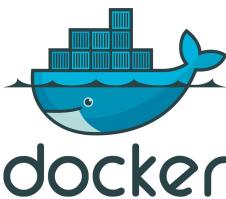
# DOCKER: LA HISTORIA



En 2008, Solomon Hykes fundó dotCloud para crear una oferta de plataforma como servicio (PaaS) independiente del idioma. El aspecto agnóstico del lenguaje era el único punto de venta de dotCloud, ya que los PaaS existentes estaban ligados a conjuntos particulares de lenguajes (por ejemplo, Heroku soportaba Ruby, y Google App Engine soportaba Java y Python). En 2010, dotCloud participó en el programa de aceleración Y Combinator, donde se puso en contacto con nuevos socios y empezó a atraer inversiones importantes.

El punto de inflexión más importante se produjo en marzo de 2013, cuando dotCloud abrió Docker, el bloque de construcción central de dotCloud. Mientras que algunas empresas podrían haberse asustado de estar regalando sus judías mágicas, dotCloud reconoció que Docker se beneficiaría

enormemente de convertirse en un proyecto impulsado por la comunidad.



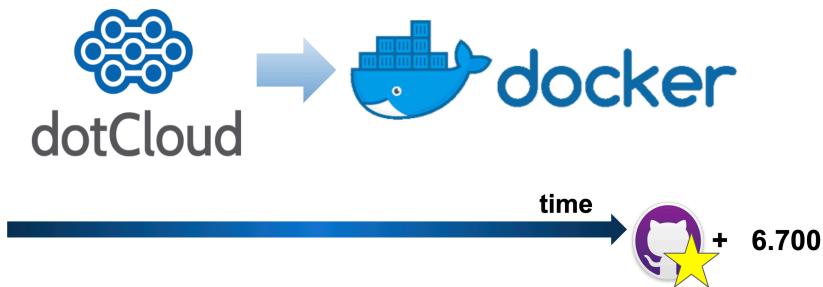
2013-2015

2015-2017



2017-PRESENTE

Las primeras versiones de Docker eran poco más que una envoltura de LXC junto con un sistema de archivos de unión, pero la aceptación y la velocidad de desarrollo fueron sorprendentemente rápidas. En seis meses, contaba con más de 6.700 estrellas en GitHub y 175 colaboradores no empleados.



Esto llevó a dotCloud a cambiar su nombre por el de Docker, Inc. y a reorientar su modelo de negocio. Docker 1.0 se anunció en junio de 2014, apenas 15 meses después de la versión 0.1. Docker 1.0 representó un salto importante en la estabilidad y la fiabilidad: ahora se declaraba "listo para la producción", aunque ya había visto el uso de

producción en varias empresas, como Spotify y Baidu.

Al mismo tiempo, Docker empezó a convertirse en una plataforma completa y no sólo en un motor de contenedores, con el lanzamiento de Docker Hub, un repositorio público para contenedores.

Otras empresas no tardaron en ver el potencial de Docker. Red Hat se convirtió en un socio importante en septiembre de 2013 y comenzó a utilizar Docker para impulsar su oferta de nube OpenShift. Google, Amazon y DigitalOcean se apresuraron a ofrecer soporte para Docker en sus nubes, y varias startups comenzaron a especializarse en el alojamiento de Docker, como Stack- Dock. En octubre de 2014, Microsoft anunció que las futuras versiones de Windows Server serían compatibles con Docker, lo que representó un enorme cambio de posicionamiento para una empresa tradicionalmente asociada con el software empresarial hinchado.

En la DockerConEU de diciembre de 2014 se anunció Docker Swarm, un gestor de clusters para Docker, y Docker Machine, una herramienta CLI

para el aprovisionamiento de hosts Docker. Esto fue una clara señal de la intención de Docker de proporcionar una solución completa e integrada para ejecutar contenedores y no permitir que se limiten a proporcionar únicamente el motor Docker.

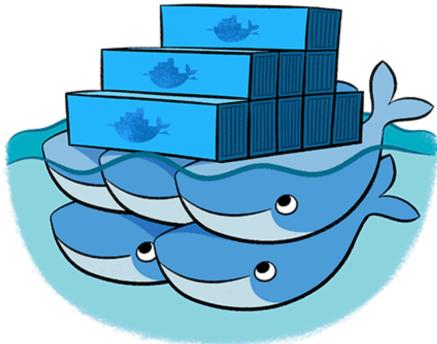
También ese diciembre, CoreOS anunció el desarrollo de rkt, su propio tiempo de ejecución de contenedores, y el desarrollo de la especificación de contenedores appc. En junio de 2015, durante la DockerCon de San Francisco, Solomon Hykes, de Docker, y Alex Polvi, de CoreOS, anunciaron la formación de la Open Container Initiative (entonces llamada Open Container Project) para desarrollar un estándar común para los formatos y tiempos de ejecución de contenedores.



También en junio de 2015, el proyecto FreeBSD anunció que Docker ya era compatible con FreeBSD, utilizando ZFS y la capa de compatibilidad de Linux. En agosto de 2015, Docker y Microsoft publicaron una "vista previa técnica" del motor Docker para el servidor Windows. Con el lanzamiento de Docker 1.8, Docker introdujo la función de confianza del contenido, que verifica la integridad y el editor de las imágenes Docker.

La confianza en el contenido es un componente fundamental para crear flujos de trabajo de confianza basados en imágenes recuperadas de los registros de Docker.

En la DockerConEU de diciembre de 2014 se anunció Docker Swarm, un gestor de clusters para Docker, y Docker Machine, una herramienta CLI para el aprovisionamiento de hosts Docker. Esto fue una clara señal de la intención de Docker de proporcionar una solución completa e integrada para ejecutar contenedores y no permitir que se limiten a proporcionar únicamente el motor Docker.



También ese diciembre, CoreOS anunció el desarrollo de rkt, su propio tiempo de ejecución de contenedores, y el desarrollo de la especificación de contenedores appc. En junio de 2015, durante la

DockerCon de San Francisco, Solomon Hykes, de Docker, y Alex Polvi, de CoreOS, anunciaron la formación de la Open Container Initiative (entonces llamada Open Container Project) para desarrollar un estándar común para los formatos y tiempos de ejecución de contenedores.

También en junio de 2015, el proyecto FreeBSD anunció que Docker ya era compatible con FreeBSD, utilizando ZFS y la capa de compatibilidad de Linux. En agosto de 2015, Docker y Microsoft publicaron una "vista previa técnica" del motor Docker para el servidor Windows. Con el lanzamiento de Docker 1.8, Docker introdujo la función de confianza del contenido, que verifica la integridad y el editor de las imágenes Docker. La confianza en el contenido es un componente fundamental para crear flujos de trabajo de confianza basados en imágenes recuperadas de los registros de Docker.

# MICROSERVICIOS Y MONOLITOS

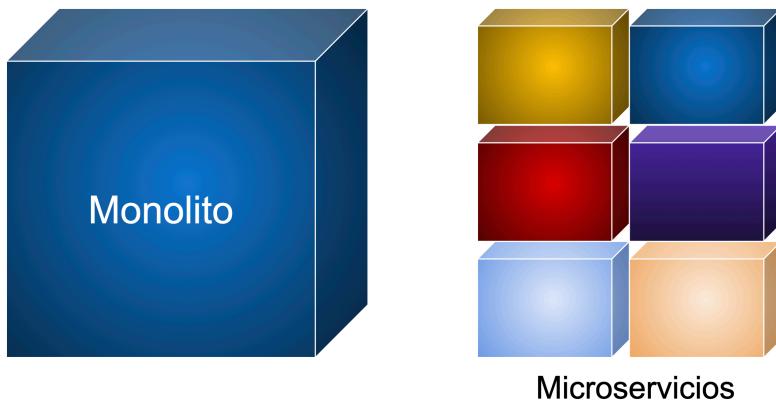


Uno de los puntos importantes del 'Curso de Docker de cero a experto' es el siguiente. Uno de los mayores casos de uso y de los más fuertes impulsores de la adopción de contenedores son los microservicios.

Los microservicios son una forma de desarrollar y componer sistemas de software de manera que se construyen a partir de componentes pequeños e independientes que interactúan entre sí a través de la red. Esto contrasta con la forma tradicional monolítica de desarrollar software, en la que hay un único gran programa, normalmente escrito en C++ o Java.

Cuando se trata de escalar un monolito, normalmente la única opción es escalar, donde la demanda extra se gestiona utilizando una máquina más grande con más RAM y potencia de CPU. Por el

contrario, los microservicios están diseñados para escalar hacia afuera, donde la demanda adicional se maneja mediante el aprovisionamiento de múltiples máquinas en las que se puede distribuir la carga. En una arquitectura de microservicios, sólo es posible escalar los recursos necesarios para un servicio concreto, centrándose en los cuellos de botella del sistema. En un monolito, hay que escalar todo o nada, lo que supone un desperdicio de recursos.



En términos de complejidad, los microservicios son un arma de doble filo. Cada microservicio individual debe ser fácil de entender y modificar. Sin embargo, en un sistema compuesto por docenas o

cientos de estos servicios, la complejidad general aumenta debido a la interacción entre los componentes individuales.

La naturaleza ligera y la velocidad de los contenedores hacen que sean especialmente adecuados para ejecutar una arquitectura de microservicios. En comparación con las máquinas virtuales, los contenedores son mucho más pequeños y rápidos de desplegar, lo que permite a las arquitecturas de microservicios utilizar el mínimo de recursos y reaccionar rápidamente a los cambios en la demanda.

# DOCKER: CI/CD



Uno de los puntos importantes del 'Curso de Docker de cero a experto' es el siguiente. CI/CD es un método para distribuir aplicaciones a los clientes con frecuencia mediante el uso de la automatización en las etapas del desarrollo de aplicaciones. Los principales conceptos que se atribuyen a CI/CD son la integración continua, la distribución continua y la implementación continua. CI/CD es una solución para los problemas que puede generar la integración del código nuevo a los equipos de desarrollo y de operaciones (también conocida como "Integration Hell").

**CONTINUOUS INTEGRATION (CI)**



**CONTINUOUS DELIVERY (CD)**

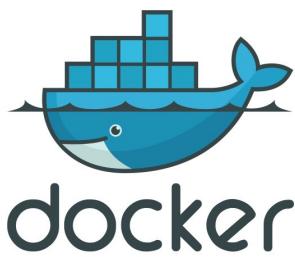
LIBERACIÓN AUTOMÁTICA AL REPOSITORIO

**CONTINUOUS DEPLOYMENT (CD)**

DESPLIEGUE AUTOMÁTICO HACIA PRODUCCIÓN

En concreto, CI/CD incorpora la automatización continua y el control permanente en todo el ciclo de vida de las aplicaciones, desde las etapas de integración y prueba hasta las de distribución e implementación. Este conjunto de prácticas se conoce como "canales de CI/CD", y cuenta con el soporte de DevOps.

La sigla CI/CD tiene diferentes significados. La "CI" en CI/CD siempre se refiere a la integración continua, que es un proceso de automatización para los desarrolladores. Si la CI tiene éxito, los cambios del código nuevo en una aplicación se diseñan, se prueban y se combinan periódicamente en un repositorio compartido. Esto soluciona el problema de que se desarollen demasiadas divisiones de una aplicación al mismo tiempo, porque podrían entrar en conflicto entre sí.



La "CD" en CI/CD se refiere a la distribución o la implementación continuas, los cuales son conceptos relacionados que suelen usarse indistintamente. Ambos conceptos se refieren a la automatización de las etapas posteriores del canal, pero a veces se usan por separado para explicar la cantidad de automatización que se está incorporando.

Por lo general, la *distribución* continua se refiere a los cambios que implementa un desarrollador en una aplicación, a los que se les realizan pruebas de errores automáticas y que se cargan en un repositorio (como GitHub o un registro de contenedor), para que luego el equipo de operaciones pueda implementarlos en un entorno de producción en vivo. Es una solución al problema de la poca visibilidad y comunicación entre los equipos comerciales y de desarrollo. Con ese fin, el propósito de la distribución continua es garantizar que la implementación del código nuevo se lleve a cabo con el mínimo esfuerzo.

La *implementación* continua (la otra "CD") hace referencia a la liberación automática de los cambios que implementa el desarrollador desde el repositorio

hasta la producción, para que los clientes puedan usarlos. Así se resuelve el problema de sobrecargar a los equipos de operaciones con procesos manuales que retrasan la distribución de las aplicaciones. Esto se basa en los beneficios de la distribución continua y automatiza la siguiente etapa del canal.

### **¿Cuál es la diferencia entre CI y CD (y la otra CD)?**

CI/CD puede especificar ya sea las prácticas relacionadas de integración y distribución continuas solamente, o las tres prácticas vinculadas de integración continua, distribución continua e implementación continua. Para complicarlo un poco más, el término "distribución continua" también abarca los procesos de la implementación continua.

En realidad, no vale la pena profundizar en la semántica. Solo debe recordar que la integración y la distribución continuas son un proceso que suele percibirse como una canalización e implica incorporar un alto nivel de automatización permanente y supervisión constante al desarrollo de las aplicaciones.

El significado de los términos varía en cada caso y depende de la cantidad de automatización que se haya incorporado a la canalización de integración y distribución continuas. Muchas empresas comienzan con la incorporación de la CI, y luego van automatizando la distribución y la implementación como parte de las apps cloud nativas, por ejemplo.

## **Integración continua**

El objetivo del desarrollo de las aplicaciones modernas es contar con múltiples desarrolladores que trabajen de forma simultánea en distintas funciones de la misma aplicación. Sin embargo, si una empresa fusiona todo el código fuente diversificado en un solo día (conocido como el "Merge Day"), las tareas resultantes pueden ser tediosas y manuales, y pueden tomar mucho tiempo. Esto sucede porque, cuando un desarrollador que trabaja de forma aislada implementa un cambio en una aplicación, existe la posibilidad de que este cambio entre en conflicto con aquellos cambios implementados simultáneamente por otros

desarrolladores. Este problema puede agravarse aún más si cada desarrollador personaliza su propio entorno de desarrollo integrado (IDE) local, en lugar de que el equipo acuerde un IDE basado en la nube.

La integración continua (CI) ayuda a que los desarrolladores fusionen los cambios que introducen en el código para incorporarlos a una división compartida (o "rama") con más frecuencia, incluso diariamente. Una vez que se fusionan los cambios implementados por un desarrollador en una aplicación, se validan con el desarrollo automático de la aplicación y la ejecución de distintos niveles de pruebas automatizadas (generalmente, pruebas de unidad e integración) para verificar que los cambios no hayan dañado la aplicación. Esto significa probar todo, desde las clases y el funcionamiento hasta los distintos módulos que conforman toda la aplicación. Si una prueba automática detecta un conflicto entre el código nuevo y el actual, la CI facilita la resolución de esos errores con frecuencia y rapidez.

## Distribución continua

Después de la automatización de los diseños y las pruebas de unidad e integración de la CI, la

distribución continua automatiza la liberación de ese código validado hacia un repositorio. Por eso, para que el proceso de distribución continua sea eficaz, es importante que la CI ya esté incorporada a su canal de desarrollo. El objetivo de la distribución continua es tener una base de código que pueda implementarse en un entorno de producción en cualquier momento.

En la distribución continua, cada etapa (desde la fusión de los cambios en el código hasta la distribución de los diseños listos para la producción) implica la automatización de las pruebas y de la liberación de código. Al final de este proceso, el equipo de operaciones puede implementar una aplicación para que llegue a la etapa de producción de forma rápida y sencilla.

## **Implementación continua**

La última etapa de la canalización consolidada de integración y distribución continuas es la implementación continua, que automatiza el lanzamiento de una aplicación a la producción, ya que es una extensión de la distribución continua, la

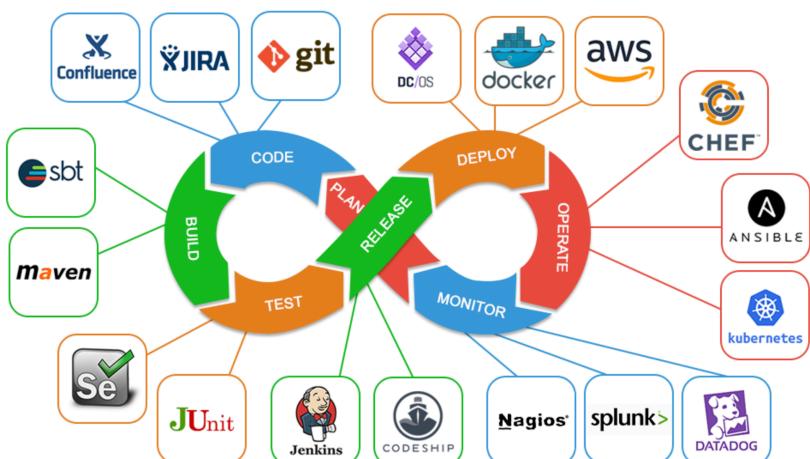
cual automatiza el lanzamiento de una compilación lista para la producción a un repositorio del código. Debido a que no hay una entrada manual en la etapa del canal anterior a la producción, la implementación continua depende, en gran medida, del correcto diseño de la automatización de pruebas.

En la práctica, la implementación continua implica que el cambio implementado por un desarrollador en una aplicación pueda ponerse en marcha unos cuantos minutos después de escribirlo (en el supuesto de que haya pasado las pruebas automatizadas). Esto facilita mucho más el proceso de recibir e incorporar continuamente los comentarios de los usuarios. En conjunto, todas estas prácticas de CI/CD relacionadas hacen que la implementación de una aplicación se lleve a cabo con menos riesgos, ya que es más fácil liberar cambios en las aplicaciones en fragmentos pequeños, en vez de hacerlo todo de una sola vez. Sin embargo, también deben realizarse muchas inversiones iniciales, ya que las pruebas automatizadas deben diseñarse para que se adapten a las distintas etapas de prueba y liberación en el canal de la CI/CD. [1]

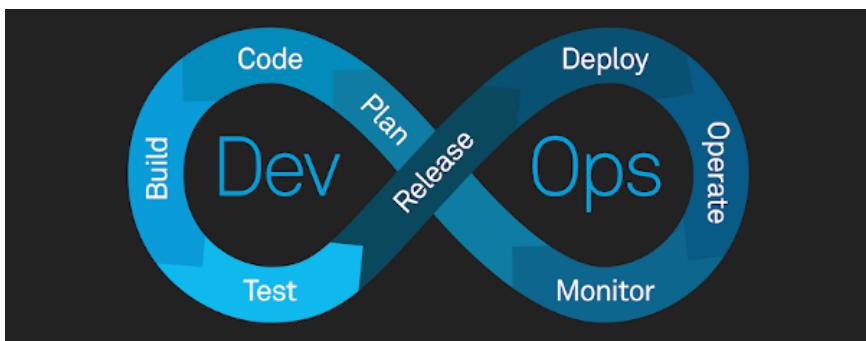
# DOCKER: DEVOPS



Uno de los puntos importantes del 'Curso de Docker de cero a experto' es el siguiente. El término "DevOps" es una combinación de las palabras "development" (desarrollo) y "operations" (operaciones), pero representa un conjunto de ideas y prácticas que van más allá de ambos conceptos, ya sea que estén juntos o separados. DevOps incluye sistemas de seguridad, maneras de trabajar en colaboración, análisis de datos, entre otras características. ¿Pero qué es?



DevOps describe los enfoques para agilizar los procesos con los que una idea (como una nueva función de software, una solicitud de mejora o una corrección de errores) pasa del desarrollo a la implementación, en un entorno de producción en que puede generar valor para el usuario. Estos enfoques requieren que los equipos de desarrollo y operaciones se comuniquen con frecuencia y aborden su trabajo con empatía hacia sus compañeros de equipo.



También es necesario contar con capacidad de ajuste y tener flexibilidad a la hora de preparar los sistemas. Con DevOps, es posible brindar una mayor potencia a aquellos que más la necesitan, gracias al autoservicio y la automatización. Los desarrolladores, que generalmente realizan codificaciones en un entorno de desarrollo estándar, trabajan en estrecha colaboración con los equipos de operaciones de TI para agilizar el diseño, las pruebas y el lanzamiento de los sistemas de software, sin comprometer la confiabilidad.

Por supuesto, esto implica cambios más frecuentes en el código y el uso más dinámico de la infraestructura. Las estrategias tradicionales de gestión no logran satisfacer este aumento de la demanda. Necesitará hacer algunos cambios para obtener una ventaja competitiva.

## **Ventajas de DevOps para las aplicaciones en contenedores de Docker**

---

Estas son algunas de las ventajas más importantes que proporciona un flujo de trabajo sólido de DevOps:

- Entregar software de mayor calidad con más rapidez y un mejor cumplimiento.
- Impulsar la mejora y los ajustes continuos en fases más tempranas y de forma más económica.
- Aumentar la transparencia y la colaboración entre las partes interesadas que participan en la entrega y el funcionamiento del software.
- Controlar los costes y usar recursos aprovisionados de forma más eficaz mientras se minimizan los riesgos de seguridad.

- Conectarse y funcionar automáticamente con muchas de las inversiones existentes de DevOps, incluidas las inversiones en código abierto.

# VENTAJAS DE LOS CONTENEDORES DOCKER



## Modularidad

---

El enfoque Docker para la creación de contenedores se centra en la capacidad de tomar una parte de una aplicación, para actualizarla o repararla, sin necesidad de tomar la aplicación completa. Además de este enfoque basado en los microservicios, puede compartir procesos entre varias aplicaciones de la misma forma que funciona la arquitectura orientada al servicio (SOA).

## Control de versiones de imágenes y capas

---

Cada archivo de imagen de Docker se compone de una serie de capas. Estas capas se combinan en una sola imagen. Una capa se crea cuando la imagen cambia. Cada vez que un usuario especifica un comando, como *ejecutar* o *copiar*, se crea una nueva capa.

Docker reutiliza estas capas para construir nuevos contenedores, lo cual hace mucho más rápido el proceso de construcción. Los cambios intermedios se comparten entre imágenes, mejorando aún más la velocidad, el tamaño y la eficiencia *-Keep it small: a closer look at Docker image sizing*. El control de versiones es inherente a la creación de capas. Cada vez que se produce un cambio nuevo, básicamente, usted tiene un registro de cambios incorporado: control completo de sus imágenes de contenedor.

## Restauración

---

Probablemente la mejor parte de la creación de capas es la capacidad de restaurar. Toda imagen tiene capas. ¿No le gusta la iteración actual de una imagen? Restáurela a la versión anterior. Esto es compatible con un enfoque de desarrollo ágil y permite hacer realidad la integración e implementación continuas (CI/CD) desde una perspectiva de las herramientas.

## Implementación rápida

---

Solía demorar días desarrollar un nuevo hardware, ejecutarlo, proveerlo y facilitarlo. Y el nivel de esfuerzo y sobrecarga era extenuante. Los contenedores basados en Docker pueden reducir el tiempo de implementación a segundos. Al crear un contenedor para cada proceso, puede compartir rápidamente los procesos similares con nuevas aplicaciones. Y, debido a que un SO no necesita

iniciarse para agregar o mover un contenedor, los tiempos de implementación son sustancialmente inferiores. Además, con la velocidad de implementación, puede crear y destruir la información creada por sus contenedores sin preocupación, de forma fácil y rentable.

Por lo tanto, la tecnología Docker es un enfoque más granular y controlable, basado en microservicios, que prioriza la eficiencia.

## REFERENCIAS

[1], ¿Qué son la integración/distribución continuas (CI/CD)?, <https://www.redhat.com/es/topics/devops/what-is-ci-cd>

[2], Ventajas de los contenedores Docker,  
<https://www.redhat.com/es/topics/containers/what-is-docker>

## SOBRE EL AUTOR

Mi nombre es Joan Amengual, soy graduado en Ingeniería Telemática por la Universidad de las Islas Baleares (UIB).



Apasionado en varios campos como en el Data Science, Inteligencia Artificial (IA) o en el Blockchain. En el campo profesional estoy trabajando en el campo de la Inteligencia Artificial (IA) basada en el razonamiento y los almacenes asociativos basados en la memoria con una empresa tecnológica de Silicon Valley, USA.