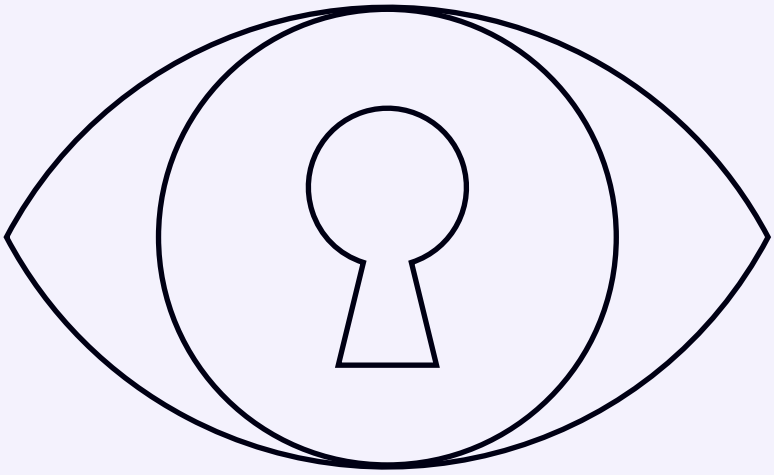

Ciberseguridad en Entornos de las Tecnologías de la Información

Docker y los contenedores de software

Módulo: Puesta en producción segura

UF: Implantación de sistemas seguros de desplegado de software

edix

ÍNDICE

DOCKER Y LOS CONTENEDORES DE SOFTWARE

Docker es un **proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.**

Es una **plataforma de virtualización a nivel de sistema operativo** que permite crear una aplicación y empaquetarla en un contenedor junto con sus dependencias. Esto garantiza que la aplicación funcione sin problemas en cualquier otra máquina o entorno, independientemente de sus configuraciones. Esta característica permite promocionar de manera más rápida entre entornos.

Docker consta de **cuatro componentes:**

- Docker Client y Daemon.
- Imágenes.
- Registros de Docker.
- Contenedores.

Docker emplea **características del kernel de Linux** (*namespaces* y *cgroups*) para construir contenedores sobre un sistema operativo y automatizar la implementación de la aplicación en los contenedores. **El software que alojan estos contenedores se conoce como Docker Engine.**

Básicamente, Docker funciona de manera muy similar a una máquina virtual (VM). Sin embargo, hay una gran diferencia entre ambos. Mientras una máquina virtual crea un sistema operativo virtual completo, Docker permite que las aplicaciones usen el mismo kernel de Linux del sistema en el que se ejecutan.

Solo necesita enviar la aplicación y sus dependencias (las que no comparta con el host) dentro de un contenedor para poderlo ejecutar. El proceso de "contenedorización" no solo reduce el tamaño de la aplicación, sino que también aumenta la capacidad de aprovechamiento de los recursos y facilita la escalabilidad.

1. ¿POR QUÉ SE USA?

Docker ofrece muchas ventajas a las empresas. Es capaz de **reducir los costes operativos en infraestructura y mantenimiento.** Al mismo tiempo, reduce el *time to market* permitiendo ofrecer nuevas funcionalidades de manera rápida y escalable.

Otras de las ventajas que podemos destacar de Docker son:

- **Uso eficiente de los recursos del sistema.** Las aplicaciones en contenedores requieren menos memoria (gracias al tamaño reducido de la aplicación al compartir librerías con el kernel de sistema operativo). Esto permite que las aplicaciones puedan crearse, usarse y destruirse para liberar recursos rápidamente. La capacidad que nos ofrecen los contenedores de empaquetar

aplicaciones y su bajo consumo nos permite desplegar multitud de contenedores dentro de un *host*, reduciendo el coste en infraestructura y aumentando la capacidad de carga de trabajo de la aplicación, optimizando el uso del procesamiento. Además, se produce un ahorro considerable en licencias de SO, ya que se necesitarán menos instancias para ejecutar la misma carga que con las máquinas virtuales.

- **Ciclos de entrega de software más rápido.** En un mercado tan competitivo y cambiante, las empresas tienen que responder rápidamente a las necesidades del mercado. Esto significa, no sólo ser capaces de adaptarse a las necesidades cambiantes de sus clientes, si no a la actualización constante de los productos, servicios, aplicaciones, librerías que su cliente utiliza. Estas dos tareas se pueden realizar usando contenedores de Docker. Con contenedores podemos desplegar, en poco tiempo, una nueva funcionalidad, así como una versión anterior al que se le agrega una nueva funcionalidad. También nos permite dar "marcha atrás" a una versión anterior. Lo que nos permite ciclos de entrega más rápidos.

- **Permite la portabilidad de la aplicación.** Un contenedor puede instanciarse y ejecutarse en cualquier máquina que tenga Docker instalado. Esto permite que las aplicaciones embebidas en contenedores puedan promocionarse sin problemas de configuración o rendimiento entre los diferentes entornos.

- **Mejora la productividad de los desarrolladores.** Gracias a los contenedores de Docker, los desarrolladores no tienen que crear el entorno cada vez que las aplicaciones se tienen que ejecutar en un entorno o máquina diferente, dejando atrás la frase "en

mi máquina funciona" y permitiéndoles ser más productivos. La estandarización de los entornos gracias a los contenedores también facilita la tarea al equipo de DevOps, pudiendo acelerar los tiempos y la frecuencia de las implantaciones.

- **Aumenta la eficiencia operativa.** Los contenedores de Docker mejoran la agilidad operativa de las empresas. Simplificando la cadena de suministros y automatizando la gestión de múltiples aplicaciones en un único modelo operativo centralizado. Tener un modelo operativo único implica una reducción en el tiempo promedio empleado en la resolución de incidencias, lo que tiene un impacto directo en la satisfacción del cliente.

2. ¿QUÉ ES UN CONTENEDOR?

Un contenedor es un conjunto de uno o más procesos que están aislados del resto del sistema. Un contenedor no es más que una instancia en ejecución de una imagen.

Un **contenedor de Docker lo podemos descomponer en:**

- Una imagen de Docker.
- Un entorno de ejecución.
- Un conjunto de instrucciones a ejecutar.

3. ¿CÓMO FUNCIONA DOCKER?

Para entender cómo funciona Docker debemos conocer las **partes fundamentales de las que se compone:**

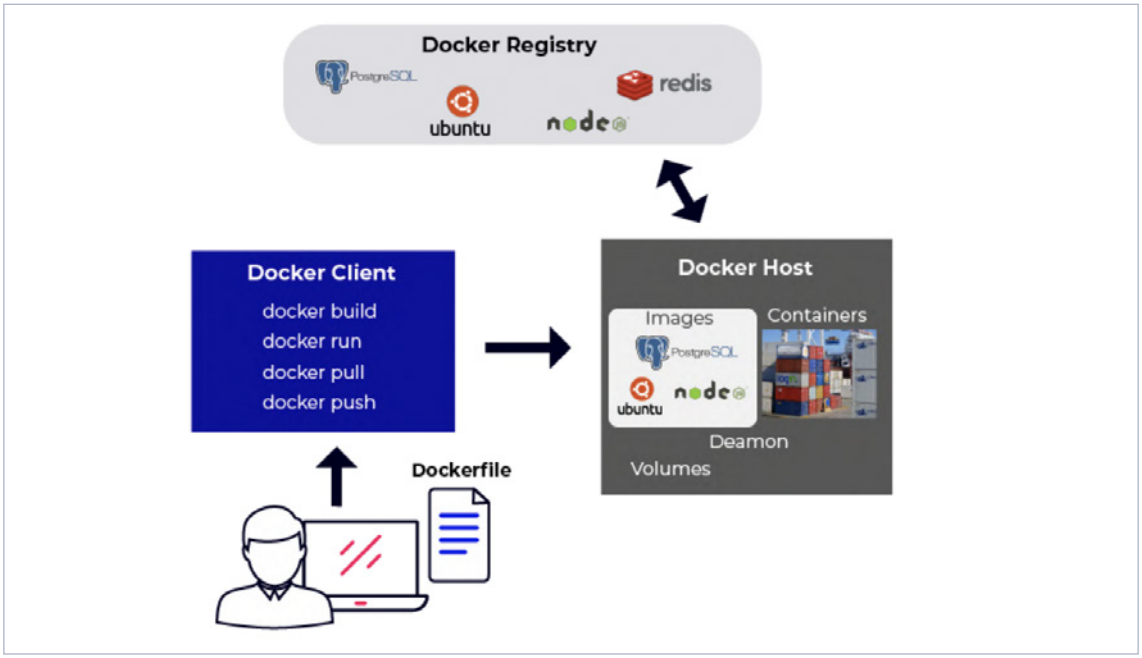

Figura 1. Partes de Docker

'Docker Engine'

El motor de Docker es la **capa sobre la que funciona Docker.** Es una herramienta ligera que administra contenedores, imágenes, *builds*, etc. Se ejecuta de forma nativa en sistemas Linux y está compuesto por:

1. Un 'Docker Daemon' que se ejecuta en el *host*.
2. Un Docker Client que se comunica con el Docker Daemon para ejecutar comandos ('docker run', 'docker ps', 'docker rm', etc.).
3. Una API REST para interactuar con el 'Docker Daemon' de forma remota.

'Docker Client'

'Docker Client' es la **interfaz que le ofrece Docker al usuario final para comunicarse con el 'Docker Daemon'.** El usuario nunca se comunica directamente con el' Docker Daemon'. Se puede ejecutar desde la máquina host, pero no es obligatorio. Puede ejecutarse desde otra máquina. En posteriores apartados veremos las operaciones que puede ejecutar el usuario con 'Docker Client'.

'Docker Daemon'

El 'Docker Daemon' es realmente quien **ejecuta los comandos enviados por 'Docker Client':** cómo construir, arrancar o parar contenedores. El 'Docker Daemon' se ejecuta desde el *host*.

'Dockerfile'

Un 'Dockerfile' es el **fichero donde se escriben las instrucciones para construir una imagen de Docker.** En posteriores apartados veremos en mayor detalle qué es y cómo se construye un 'Dockerfile'.

'Docker Images'

Las 'Docker Images' son **plantillas de solo lectura con un conjunto de instrucciones escritas en su 'Dockerfile' para construir el contenedor.** En la 'Docker Images' se define tanto la aplicación a empaquetar como las dependencias que necesita para arrancar.

La imagen Docker se construye utilizando un 'Dockerfile'. Cada instrucción en el 'Dockerfile' añade una nueva capa a la imagen. Las **capas** representan una porción del sistema de archivos de la imagen que añade o reemplaza los que tiene la imagen base de la que parte. Las capas son clave para hacer que los contenedores sean ligeros. Docker utiliza un sistema de archivos llamado 'Union File Systems' para lograrlo.

'Union File Systems'

Docker usa 'Union File Systems' para **construir una imagen**. Podríamos verlo como un sistema de archivos apilable, donde cada archivo o directorio del sistema de archivos separado en ramas, puede superponerse formando un único sistema de archivos. Esta operación es transparente para el contenedor.

Se controla mediante punteros las diferentes versiones de un mismo recurso en cada capa. De tal manera, que si se necesita modificar una capa (por algún cambio en el 'Dockerfile'), se hace una copia de la original y esa capa es la que se modifica. Así es como los sistemas de archivos simulan ser "escribibles" sin realmente permitir la escritura. En otras palabras: un sistema de "copia sobre escritura".

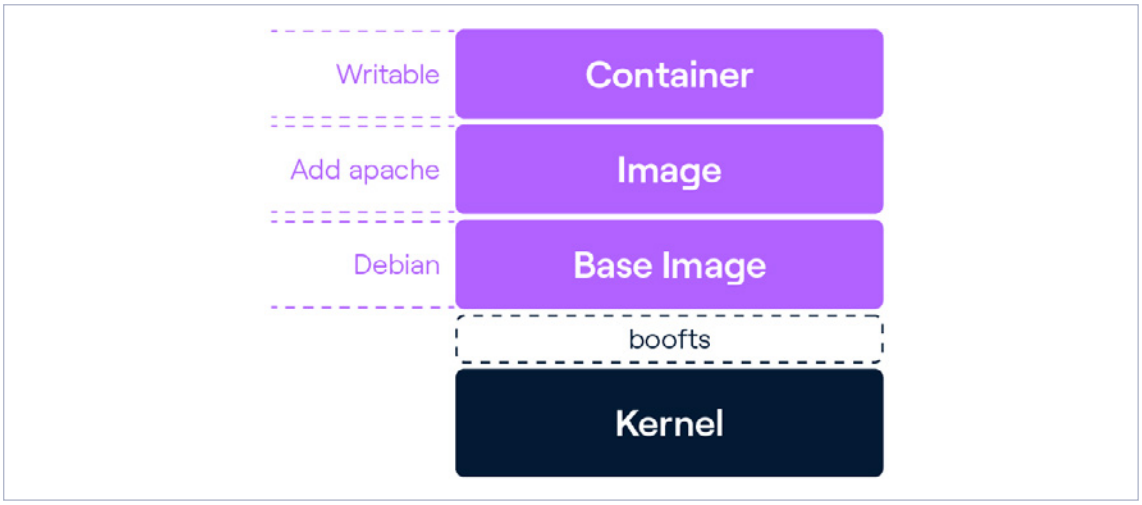

Figura 2. Docker

Los sistemas de capas ofrecen dos beneficios principales:

1. Evitan la duplicación: evita la duplicación de archivos al crear o ejecutar contenedores, haciéndola más rápida y eficiente.

2. Segregación de capas: hacer un cambio es mucho más rápido, ya que cuando cambias una imagen (modificando sentencias del 'Dockerfile') solo se actualiza la capa a la que afecta el cambio.

'Volúmenes'

Los volúmenes son la **parte de datos de un contenedor y se inician en su fase de creación.** Son la forma que tiene el contenedor de persistir y compartir los datos y están separados del 'Union File Systems' predeterminado y existen como directorios y archivos normales del sistema de archivos del *host*. Por lo que, si un contenedor se borra o actualiza, sus volúmenes permanecerán intactos. Un mismo volumen puede ser usado por varios contenedores.

'Contenedores Docker'

Un contenedor Docker, como se ha dicho en apartados anteriores, no es más que una **instancia en ejecución de una imagen.**

4. ANATOMÍA DE CONTENEDORES

Arquitectura

'Docker Engine' proporciona una interfaz REST que puede usar cualquier otra herramienta como por ejemplo Docker CLI, Docker para MacOs o para Windows y, también, Kubernetes.

Por debajo de 'Docker Engin'e se encuentra **'Containerd' y 'runC'** para gestionar el ciclo de vida de los contenedores. Para hacer esta gestión, 'Containerd' y 'runC', aprovechan funcionalidades del kernel de Linux como 'cgroups', 'namespaces' y 'UnionFS'.

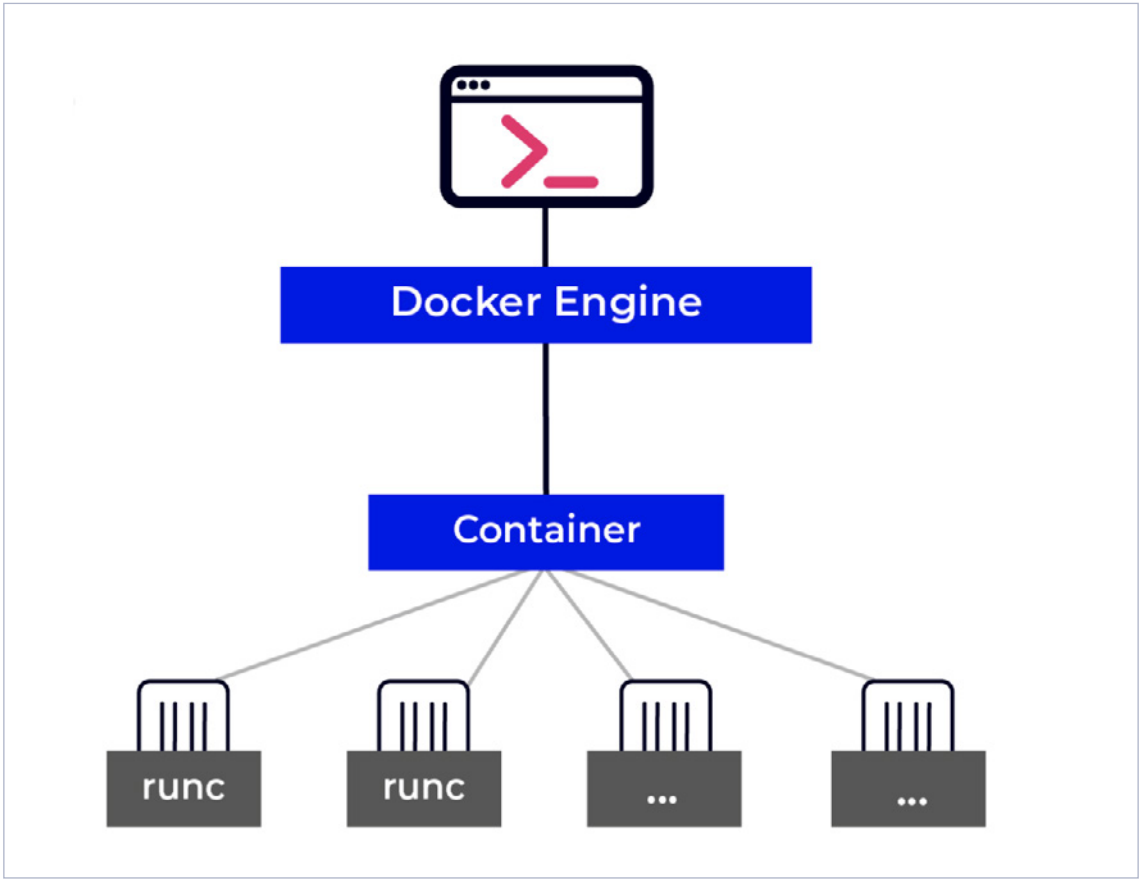

Figura 3. Arquitectura

'Namespaces'

Docker utiliza una función del kernel de Linux llamada 'Namespaces' para **proporcionar encapsulamiento y restringir la visibilidad al exterior de los contenedores.**

> Los 'Namespaces' son el mecanismo que utiliza docker para que los procesos que tiene dentro piensen que son una instancia independiente del resto del sistema.

Podemos restringir diferentes elementos del sistema dependiendo del tipo de 'Namespace' que utilicemos para establecer las restricciones, pudiendo restringir la visibilidad de PIDs, de redes, del sistema de archivos, etc.

'Control Groups (cgroups)'

La función 'Groups' es la encargada de **gestionar el uso de recursos en los contenedores. Esto incluye limitar y aislar recursos de dichos procesos.** El uso de 'cgroup' nos permite:

- Limitar recursos (memoria, CPU, I/O, uso de red, etc.).
- Priorizar unos grupos con respecto a otros.
- Seguimiento del uso de recursos por motivos de facturación.
- Control sobre la ejecución de los procesos.

'Union Filesystem (UnionFS)'

Como se ha explicado en el apartado anterior, la función de 'UnionFS' es la **superposición de sistemas de archivos para formar un sistema de archivos único.**

De este modo, los contenidos de directorios en las distintas capas que tienen la misma ruta en distintos sistemas aparecerán juntos en un único directorio. Construyendo un nuevo sistema de archivos virtual de la composición de muchos.

Fontanería de contenedores: 'runC' y 'Containerd'

Las dos tecnologías sobre las que funciona Docker son 'runC' y 'Containerd'.

- '**runC**' es una herramienta que funciona como cliente que permite construir y ejecutar contenedores siguiendo la especificación de *Open Container Initiative* (OCI), aportando funcionalidades de alto nivel como la distribución de las imágenes, el almacenamiento, la gestión de las redes, etc.

- Por otro lado, '**Containerd** 'es un demonio que actúa como una API Fachada para los contenedores y el SO, la fachada les provee de una capa superior de abstracción que les permite trabajar con entidades de alto nivel como son *snapshots* y contenedores.

5. KUBERNETES

Kubernetes es una **plataforma de código abierto para automatizar la implementación, escalado y operaciones de contenedores de aplicaciones en grupos de *hosts*, proporcionando infraestructura centrada en contenedores.**

Con Kubernetes, se puede responder rápida y eficientemente a las siguientes necesidades:

- Despliegue sus aplicaciones de forma rápida y predecible.
- Escalado de aplicaciones sobre la marcha.
- Despliegue de nuevas funcionalidades.
- Optimizar el uso del hardware utilizando solo los recursos que necesita.

Kubernetes puede programar y ejecutar contenedores de aplicaciones en grupos de máquinas físicas o virtuales. Sin embargo, Kubernetes también permite a los desarrolladores **"cortar el cable" a máquinas físicas y virtuales**, pasando de una infraestructura centrada en el host a una infraestructura centrada en el contenedor, que proporciona todas las ventajas y beneficios inherentes a los contenedores.

> Kubernetes proporciona la infraestructura para construir un entorno de desarrollo verdaderamente centrado en contenedores.

Kubernetes satisface una serie de **necesidades** comunes de aplicaciones que se ejecutan en producción, como:

- Ubicar conjuntamente los procesos de ayuda, facilitar las aplicaciones compuestas y preservar el modelo de una aplicación por contenedor.
- Montaje de sistemas de almacenamiento.
- Distribución de secretos.
- Control de salud de la aplicación.
- Replicado de instancias de aplicaciones.
- Autoescalamiento horizontal.
- Balanceo de carga.
- Monitoreo de recursos.

Esto **proporciona simplicidad para gestionar una Plataforma como un Servicio (PaaS) con la flexibilidad de la Infraestructura como un Servicio (IaaS)**, y facilita la portabilidad entre los proveedores de infraestructura.

edix

Creamos Digital Workers