ÍNDICE

Prefacio

SOBRE EL LIBRO

El libro tiene como objetivo facilitar los conocimientos necesarios para entender cómo Docker gestiona la seguridad tanto desde el punto de vista de la máquina donde ejecutamos Docker como desde el punto de vista del desarrollo y despliegue de imágenes. Ayudará a entender cómo sacar provecho de la agilidad, control y portabilidad que proporciona Docker. El libro trata de ofrecer un enfoque teórico-práctico. Casi todos los aspectos de las herramientas relacionadas con contenedores descritos en este libro están respaldados con ejemplos de cómo funcionan a nivel práctico.

OBJETIVOS DEL LIBRO

1. Utilizar Docker como plataforma de despliegue de contenedores e imágenes docker teniendo en cuenta la seguridad de las aplicaciones.
2. Crear, implementar y desplegar contenedores de forma segura con Docker.
3. Conocer y usar herramientas que nos permiten auditar la seguridad de la máquina donde ejecutamos las imágenes docker.
4. Conocer y usar herramientas que nos permiten auditar la seguridad de las imágenes docker.
5. Automatizar tareas de networking en contenedores.
6. Fomentar el interés por la investigación y la seguridad informática.

SOBRE EL AUTOR

José Manuel Ortega es ingeniero de software e investigador de seguridad centrado en las nuevas tecnologías, el código abierto y la seguridad de las aplicaciones. En los últimos años, ha mostrado interés en el desarrollo de la seguridad, especialmente dentro de los ecosistemas de Python y Java.

Actualmente desarrolla tareas de análisis y pruebas de la seguridad de las aplicaciones. Ha colaborado con la escuela oficial de ingenieros en informática e impartido conferencias a nivel nacional e internacional relacionadas con la seguridad de las aplicaciones. Más información acerca de sus conferencias y artículos escritos están disponibles en su sitio personal http://jmortega.github.io

1 INTRODUCCIÓN A LA SEGURIDAD EN DOCKER

INTRODUCCIÓN A LOS CONTENEDORES DOCKER

Docker es una plataforma abierta que permite construir y ejecutar aplicaciones distribuidas basada en contenedores que se ejecutan en Linux y funcionan tanto en máquinas físicas como virtuales simplemente usando un runtime. Está desarrollado en el lenguaje Go y usa librerías del sistema operativo, así como funcionalidades del Kernel de Linux en el que se ejecuta. Consta de un motor con API y un cliente que pueden ejecutarse en la misma máquina o en máquinas separadas.

Los contenedores aportan las características y herramientas concretas que se necesitan en la actualidad, donde la portabilidad, escalabilidad, alta disponibilidad y los microservicios en aplicaciones distribuidas son cada vez más utilizados. Cada vez se desarrollan menos aplicaciones monolíticas y más basadas en módulos o en microservicios, que permiten un desarrollo más ágil, rápido y a la vez portable. Empresas tan importantes como Netflix, Spotify o Google usan arquitecturas basadas en microservicios en muchos de los servicios que ofrecen y dichas organizaciones utilizan Docker para simplificar y acelerar su proceso de desarrollo e implementación de aplicaciones. Los atributos pueden tener un valor constante, en cuyo caso no se preguntará por su valor en el momento de la inserción del bloque.

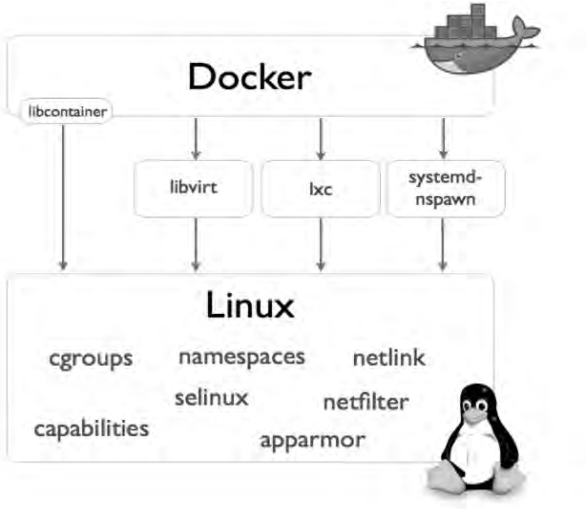
Los contenedores Docker son portátiles en entornos de desarrollo, prueba y producción que se ejecutan localmente en dispositivos físicos o virtuales máquinas, en centros de datos o en diferentes proveedores de servicios en la nube.

Docker es una evolución de otras de las tecnologías que se han implantado para el manejo de contenedores en Linux como **LXC (Linux Containers)**, en el que se tiene un

Kernel ligero común entre todos los contenedores, pero al mismo tiempo todas las librerías son independientes entre cada uno de los contenedores.

La documentación oficial de Docker es muy completa y dispone de muchos recursos y ejemplos de uso de todas las herramientas en la dirección https://docs.docker.com

Docker utiliza las funciones subyacentes del **Kernel de Linux** que permiten la ejecución de contenedores. El siguiente diagrama muestra los controladores de ejecución y las características del Kernel utilizados por Docker.



Docker puede limitar y controlar los recursos a los que accede la aplicación en el contenedor, generalmente usando su propio sistema de archivos como UnionFS o variantes como AUFS, btrfs, vfs, Overlayfs o Device Mapper que básicamente son sistemas de ficheros en capas. La forma de controlar los recursos y capacidades que hereda del host es mediante espacios de nombres (namespaces) y cgroups de Linux. Esas opciones que ya podemos encontrar en sistemas Unix, Docker las simplifica y las hace más usables de cara al usuario.

Otro uso común de Docker es la portabilidad de aplicaciones, imagina una aplicación que solo funciona con Python 3.6 y hacerla funcionar en un sistema Linux con Python 2.x no es una tarea trivial, piensa en lo que puede suponer en un sistema en producción actualizar Python, con contenedores sería casi automático, descargar la imagen del contenedor que ejecute esa versión de Python específica y ejecutar la aplicación.

Los contenedores permiten a los desarrolladores empaquetar grandes o pequeñas cantidades de código y sus dependencias juntas en un paquete aislado del resto. Este modelo permite la ejecución de múltiples contenedores de forma aislada.

Los contenedores Docker se construyen a partir de las imágenes del contenedor utilizando sistemas de archivos en capas proporcionando el contenedor la ejecución de la imagen. Un contenedor no es más que la instancia de una imagen en ejecución, ya sea una aplicación, servidor web, sistema operativo, lo que esté ejecutando la imagen. Las imágenes se gestionan y distribuyen desde registros como **Docker Cloud** y **Docker Trusted Registry** como sitios de confianza que crean y despliegan aplicaciones.

Docker proporciona una forma de ejecutar aplicaciones aisladas de forma segura en un contenedor, empaquetado con todas sus dependencias y bibliotecas. Como su aplicación siempre se puede ejecutar con el entorno que espera en la imagen de construcción, las pruebas y la implementación son más simples, ya que su compilación será totalmente portátil y estará lista para ejecutarse según lo diseñado en cualquier entorno.

COMPARATIVA MAQUINA VIRTUAL VS CONTENEDOR

El **contenedor** utiliza los recursos del Kernel en el que se ejecuta, por lo tanto, el entorno de ejecución de una aplicación generalmente se ejecuta en espacios de nombres protegidos y grupos de control. Desde el punto de vista de la seguridad, incluso cuando el contenedor se ejecuta con privilegios de root administrador, las posibilidades de que un atacante escale privilegios de un contenedor al host u otros contenedores son remotas.

Los contenedores comparten el mismo Kernel de Linux, son independientes de la plataforma, lo que los hace portátiles para cualquier entorno. Otros beneficios del uso de contenedores incluyen encapsulación y escalabilidad. La **encapsulación** es una característica que permite empaquetar todo lo que necesita la aplicación a nivel de dependencias y variables de entorno dentro del contenedor. Los contenedores también son **escalables**, lo que significa que pueden reducirse o ampliarse dinámicamente en función de los recursos que necesite el contenedor para ejecutarse en cada momento.

Estas son algunas de las diferencias entre las VM y los contenedores Docker:

- Docker está orientado a la aplicación, mientras que las VM están orientadas al sistema operativo.

- Los contenedores Docker comparten un sistema operativo con otros contenedores Docker. Por el contrario, las máquinas virtuales tienen su propio sistema operativo administrado por un hipervisor.
- Los contenedores Docker están diseñados para ejecutar un proceso principal, no para administrar múltiples conjuntos de procesos.

Una de las principales diferencias entre un contenedor Docker y una VM es que un contenedor está diseñado para ejecutar un proceso. En el caso del sistema operativo Linux existe un proceso de inicio que se suele llamar "init" o "systemd", cuyo objetivo es administrar el mantenimiento de todos los demás procesos que se ejecutan en ese sistema operativo. Esto es diferente de una VM ya que no tiene un proceso de inicialización.

Los procesos en contenedores se están ejecutando en la misma instancia del Kernel de Linux que el sistema operativo del host. Incluso se muestran en la salida del comando ps en el servidor Docker. Eso es completamente diferente de un hipervisor donde la profundidad del aislamiento del proceso generalmente incluye ejecutar una instancia completamente separada del sistema operativo para cada máquina virtual.

Los contenedores tienen varias **ventajas** con respecto a las máquinas virtuales:

- Los contenedores comparten recursos con el sistema operativo anfitrión, lo que los hace más eficientes. Los contenedores se pueden iniciar y detener en cuestión de segundos.
- La portabilidad de los contenedores tiene el potencial de eliminar toda una clase de errores causados por cambios en el entorno de ejecución. El uso de contenedores trata de poner fin al dicho "en mi máquina local funciona".
- Los desarrolladores pueden descargar y ejecutar aplicaciones complejas sin necesidad de dedicar tiempo adicional en tareas de configuración o preocupándose por los cambios requeridos en su sistema. A su vez, los desarrolladores de dichas aplicaciones pueden evitar preocuparse por las diferencias entre los diferentes entornos y la disponibilidad de dependencias.

El enfoque de contenedores se puede resumir en las siguientes características:

- Se elimina el concepto de hipervisor y máquina virtual.
- Docker Engine se ejecuta en la parte superior del sistema operativo host.
- El motor Docker ejecuta aplicaciones junto con sus dependencias como procesos aislados que comparten el núcleo del host.
- Arrancar y detener un contenedor lleva segundos en lugar de minutos.

La principal diferencia entre un contenedor y una máquina virtual es que los contenedores comparten el Kernel de la máquina host con todos los demás contenedores desplegados en ella, mientras que las máquinas virtuales tienen muchos sistemas operativos que se ejecutan en una máquina con la ayuda de un hipervisor. Esta característica de los contenedores los hace ligeros en peso en comparación con las máquinas virtuales y también ayuda en la gestión efectiva de la memoria. Los contenedores tienen una serie de ventajas, entre las que destacan:

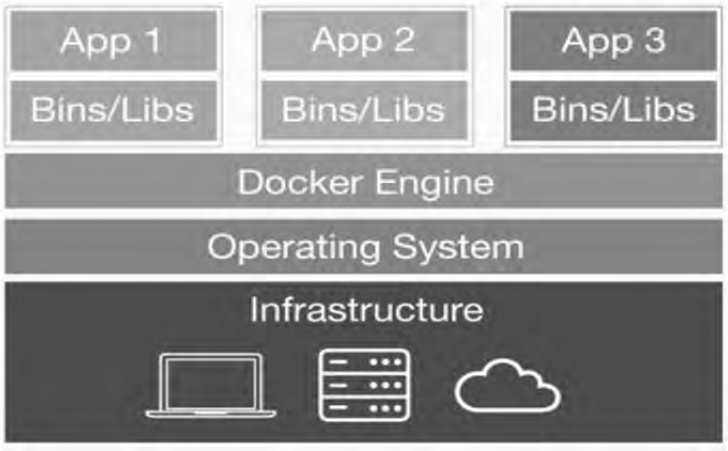
- Los contenedores ofrecen una alternativa para ejecutar aplicaciones directamente en el host que puede hacer que las aplicaciones sean más rápidas, más portátiles y más escalables.
- Por lo general, los contenedores requieren memoria en megabytes, mientras que una máquina virtual puede ocupar varios gigabytes. Por lo tanto, un solo servidor puede alojar un número mucho mayor de contenedores, en comparación con la cantidad de máquinas virtuales que puede alojar.
- Como los contenedores son ligeros, el tiempo de arranque se reduce drásticamente.
- Los contenedores también proporcionan modularidad en el desarrollo. En lugar de ejecutar una aplicación completa en un único contenedor, se puede dividir en módulos y ejecutar en diferentes contenedores, por ejemplo, podemos tener en un contenedor el frontend y en otro la base de datos de una aplicación. Esto no solo aumenta el rendimiento de una aplicación, sino que también hace que el mantenimiento del código más fácil.

En cuanto al consumo de espacio de disco y requisitos de procesamiento, lo normal es que un contenedor no ejecute un sistema operativo completo, sino que lo que se suele hacer es utilizar un sistema operativo mínimo sin interfaz gráfica y con solo los recursos necesarios para ejecutar las aplicaciones.

La mayoría de las tecnologías modernas de contenedores pueden hacer uso de las herramientas de seguridad integradas de Linux, como las políticas AppArmor, SELinux y Seccomp, Grsecurity, grupos de control (cgroups) y espacios de nombres (namespaces) del Kernel.

DOCKER ENGINE

En el núcleo está Docker Engine, que proporciona el motor de ejecución y las principales características a nivel de orquestación, programación, redes y seguridad para construir e implementar una o varias aplicaciones de contenedores. Docker Engine se puede instalar en cualquier sistema operativo Unix ya sea en una máquina física o en un centro de datos de nube pública o privada.



Los contenedores Docker dentro de un mismo host comparten el Kernel de la máquina sobre la cual se están ejecutando. Los comandos ejecutados desde un contenedor Docker aparecen en la tabla de procesos en el host y, en muchos aspectos son similares mucho a cualquier otro proceso que se ejecute en el sistema.

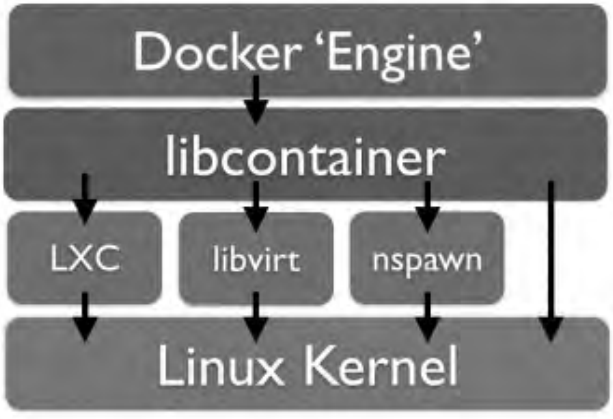
Entre los puntos a analizar podemos destacar:

- **Sistema de archivos:** el contenedor tiene su propio sistema de archivos y no puede ver el sistema de archivos del sistema host de forma predeterminada, salvo que se creen volúmenes o se monten directorios del host de forma explícita dentro del contenedor. También destacar que ficheros como /etc/hosts y /etc/resolv.conf se pueden montar de forma automática dentro del contenedor.
- **Tabla de procesos:** de forma predeterminada, los procesos dentro de un contenedor no pueden ver la tabla de procesos del host, sino que tienen su propia tabla de procesos. Desde el interior del contenedor, un proceso no puede ver ningún otro proceso ejecutándose en el host que no se haya iniciado dentro del contenedor.
- **Interfaces de red:** de forma predeterminada, el proceso Daemon de Docker define una dirección IP a través de DHCP desde un conjunto de direcciones IP privadas. Docker admite varios modos de red, como permitir que los contenedores utilicen las interfaces de red de otro contenedor, las interfaces de red del host directamente o sin interfaces de red. De esta forma, es posible exponer un puerto desde el interior del contenedor a número de puerto en el host. Es lo que se llama mapear un puerto desde el contenedor hacia el Docker host.

- **Dispositivos:** los procesos dentro del contenedor no pueden ver directamente los dispositivos en el sistema host. Nuevamente, se puede establecer una opción de privilegio especial en tiempo de ejecución del contenedor para otorgar ese privilegio.

Arquitectura Docker Engine

Docker Engine utiliza una arquitectura cliente-servidor. Un cliente Docker se comunica con el Docker Daemon, que hace el trabajo de construir, enviar y ejecutar los contenedores Docker para un servicio de aplicación específico. Todas las comunicaciones entre cliente y el Docker Daemon tienen lugar a través de un API REST y se puede asegurar con TLS. Docker está escrito en Go, y el Daemon usa varias bibliotecas y funciones de Kernel para entregar su funcionalidad. Docker utiliza su propia biblioteca de contenedores conocida como **libcontainer** para su gestión, aunque inicialmente hace uso de **LXC**.



El propio motor de Docker realiza gran parte del trabajo pesado para la seguridad de los contenedores. **El motor Docker Engine actúa como capa de protección al mismo Docker, a las bibliotecas de virtualización (como libcontainer, libvirt y LXC) y al sistema operativo del Host**. El motor Docker se ejecuta como root, por lo que debe configurar los espacios de nombres para limitar el acceso al contenedor e inicializar los contenedores con los ID de usuario correctos. Es recomendable implantar medidas adicionales de seguridad a través de las características del Kernel de Linux, como SELinux o AppArmor.

AISLAMIENTO DE DOCKER

La principal ventaja de los contenedores Docker es que puede ejecutar muchas aplicaciones que dependen de diferentes bibliotecas y entornos en un Kernel único, sin interferir nunca entre sí.

Desde el punto de vista de la seguridad, Docker proporciona una forma de ejecutar aplicaciones aisladas de forma segura en un contenedor, empaquetado con todas sus dependencias y bibliotecas.

Todas las bibliotecas y dependencias para una aplicación se encuentran en la misma imagen de Docker, y cada imagen está compuesta por muchas capas, según la arquitectura de la aplicación.

SEGURIDAD DEL KERNEL

La seguridad de Docker depende fundamentalmente de limitar el acceso del contenedor a los recursos subyacentes del sistema operativo. Mientras que el motor del contenedor debe ejecutarse con el usuario root, no es una buena práctica que los contenedores lo hagan y es necesario crear un usuario por cada contenedor en ejecución. El modelo de aislamiento de recursos para que esto suceda se basa en un mapa virtual llamado espacios de nombres, que asigna usuarios y grupos específicos a subconjuntos de recursos (por ejemplo, redes, archivos, IPC, etc.) dentro de su espacio de nombres.

Docker Engine creará un ID de usuario específico para cada contenedor en ejecución, y luego asignará ID a contenedores en tiempo de ejecución. Un contenedor se limita a los recursos asignados a su grupo. Desde el punto de vista de la seguridad, Docker proporciona una forma de ejecutar aplicaciones aisladas de forma segura en un contenedor, empaquetado con todas sus dependencias y bibliotecas.

La tecnología de contenedores Docker aumenta la seguridad creando capas de aislamiento entre aplicaciones y entre la aplicación y el host para proteger tanto el host como los contenedores que comparten el acceso al host.

Docker va un paso más allá al abordar el aislamiento, como controlar lo que los contenedores pueden ver u obtener acceso, qué recursos pueden usar y qué pueden hacer en relación con el host sistema y otros contenedores. Los contenedores Docker proporcionan **sandboxing** y aplicación de restricciones en el acceso a los recursos con **espacios de nombres y cgroups de Linux.**

PRINCIPIOS DE LA SEGURIDAD EN DOCKER

Si bien estos mecanismos de aislamiento han estado disponibles en el Kernel de Linux durante años, Docker trata de simplificar el acceso a estas capacidades, lo que permite a los administradores crear y administrar las restricciones y distribuir aplicaciones de contenedores como unidades independientes y aisladas.

Seguridad flexible

Los contenedores son unidades individuales más pequeñas que se vuelven más dinámicas o flexibles. El flujo de trabajo para mantenerlos también es más flexible. Es ideal para aplicar parches de seguridad, probar y liberar los contenedores actualizados en producción.

Limitar recursos y la exposición de información

Los contenedores pueden tener recursos limitados asignados. Esto nos ayuda a limitar la cantidad de información disponible para el sistema (y un posible atacante).

Cada contenedor obtiene los siguientes componentes:

- Pila de red
- Espacio de proceso
- Instancias del sistema de archivos

La limitación de recursos se logra mediante el uso de espacios de nombres. Los espacios de nombres son como una "vista", que solo muestra un subconjunto de todos los recursos en el sistema. Esto proporciona una forma de aislamiento: los procesos que se ejecutan dentro de un contenedor no pueden ver o afectar los procesos en otros contenedores, o el sistema host en sí mismo.

Filtrado del tráfico

Por defecto, todo el tráfico IP está permitido entre contenedores. Esto significa que pueden hacer ping entre ellos, pero también enviar otras formas de tráfico. Hubiera sido mejor si Docker hubiese aplicado un principio de "denegar todo por defecto". Esto obliga al mantenedor del contenedor a pensar qué tipo de tráfico se necesita entre contenedores individuales. Afortunadamente, el tráfico se puede filtrar y es una práctica recomendable para sistemas en producción.