Engineer

Docker

forma de distribuir un programa con todo lo que necesita

Qué es

Docker es un conjunto de herramientas para empaquetar un programa, sus bibliotecas, el entorno de ejecución de su lenguaje y sus archivos en un solo artefacto llamado imagen, y ejecutar después esa imagen como un proceso aislado llamado contenedor. La imagen se construye una vez a partir de una receta, el Dockerfile, y esa misma imagen se inicia en un portátil, en un servidor de compilación y en producción. Toda la promesa cabe en una frase: lo que se probó es, byte por byte, lo que se ejecuta.

Cuatro sustantivos concentran casi todo el vocabulario. Un Dockerfile es un archivo de texto con instrucciones como `FROM`, `COPY` y `RUN`. Una imagen es una plantilla de solo lectura construida con esas instrucciones y almacenada como una pila de capas, una por instrucción, de modo que una reconstrucción tras un cambio pequeño solo rehace las capas que cambiaron. Un contenedor es una instancia de una imagen en ejecución, con una delgada capa de escritura encima. Un registro, como Docker Hub o el de un proveedor de nube, es donde se suben y descargan las imágenes.

Bajo la línea de comandos hay un cliente y un servidor. El comando `docker` es un cliente: envía solicitudes mediante una API REST, normalmente a través de un socket Unix, a un demonio de larga duración llamado `dockerd`. El demonio controla las imágenes, los contenedores, las redes y los volúmenes. Para el ciclo de vida del contenedor delega en containerd, que de forma predeterminada utiliza runc, el pequeño programa que solicita al núcleo de Linux espacios de nombres y grupos de control, e inicia el proceso dentro de ellos. Un contenedor es, por tanto, un proceso corriente sobre el núcleo del host, con una visión restringida del sistema. No es una máquina virtual; en macOS o Windows, Docker Desktop ejecuta una pequeña máquina virtual Linux para disponer de un núcleo Linux que compartir.

El formato de imagen trascendió a la empresa que lo creó. En junio de 2015, Docker, CoreOS y otros lanzaron la Open Container Initiative. Docker le donó su formato de imagen y runc, y las especificaciones resultantes explican por qué una imagen construida con Docker funciona sin cambios en Podman, containerd o un servicio de nube gestionado.

Utilice el concepto en una decisión

Aplique «¿Qué es Docker?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Software empaquetado para comportarse igual en un portátil, un servidor de pruebas y producción, y ampliar su capacidad cuando aumente la actividad.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. 20 aplicaciones Docker en Google Cloud y AWS — 20 aplicaciones en contenedores desplegadas y monitorizadas

20 aplicaciones Docker en Google Cloud y AWS Póngase en contacto →

Por qué importa

La pregunta útil nunca es «Docker sí o no» en abstracto. Es si una carga concreta justifica una imagen o si un servicio sencillo en el host haría el mismo trabajo con menos piezas.

Una imagen se justifica cuando el programa arrastra consigo un entorno. Una aplicación Python o R que necesita un conjunto exacto de paquetes, un servicio heredado fijado a un entorno de ejecución antiguo, dos programas que requieren versiones incompatibles de la misma biblioteca: todos son difíciles de reproducir a mano y triviales de reproducir desde una imagen. Volver atrás consiste en iniciar la imagen anterior. Un nuevo desarrollador ejecuta un comando en lugar de seguir una página de instrucciones de instalación. La comparación entre Docker y una máquina virtual plantea el mismo equilibrio desde el otro lado: los contenedores arrancan en unos instantes y comparten un núcleo, por lo que caben decenas en un host que solo alojaría unas pocas máquinas virtuales, a cambio de un aislamiento más débil.

El coste es infraestructura real. De forma predeterminada, el demonio se ejecuta como root y gestiona la red; Docker también admite un modo sin privilegios de root. Hay un registro que mantener o pagar, imágenes base que acumulan correcciones de seguridad y requieren reconstruirse para recibirlas, y una capa de almacenamiento con reglas distintas a las del host. Un único binario estático sin dependencias gana muy poco con todo eso, y una unidad systemd aislada mediante sandbox lo ejecuta igual de bien.

La diferencia entre Docker y Podman también importa aquí. Ambos ejecutan las mismas imágenes y aceptan la mayoría de los mismos comandos. Docker utiliza un demonio, con privilegios de forma predeterminada; Podman inicia los contenedores directamente como procesos hijos del comando o de systemd. En un portátil la diferencia es pequeña. En un servidor cuyos cortafuegos y supervisión de procesos ya se han diseñado, puede decidir la elección.

Cómo se aplica en la práctica

Docker aparece en diversos trabajos de este sitio, y uno de los proyectos consiste en la decisión deliberada de prescindir de él.

Entre 2022 y 2024 funcionaron veinte aplicaciones en contenedores repartidas entre Google Cloud y AWS, incluidas aplicaciones de análisis escritas en R, en las que reproducir el conjunto exacto de paquetes es la parte difícil. Las veinte se desplegaron y mantuvieron con configuración y monitorización coherentes. Eso las mantuvo estables y reproducibles, y permitió diagnosticar problemas con más rapidez entre ambas nubes.

A mayor escala, un conjunto de cien aplicaciones Docker estaba alojado en hosts Ubuntu detrás de una combinación de Apache y Nginx, y cada una necesitaba un certificado TLS válido. Un flujo basado en ACME emitía, renovaba y desplegaba automáticamente esos certificados. Las cien conservaron certificados válidos y desapareció el trabajo manual con certificados.

El tercer caso apunta en la dirección contraria. En un host cuyas reglas de cortafuegos proceden de un rol de endurecimiento, la capa de contenedores se montó sobre Podman y Quadlet en lugar de Docker, porque la red predeterminada de Docker reescribe las reglas de filtrado de paquetes del núcleo y publica los puertos de los contenedores por delante de las propias reglas del cortafuegos. El motivo se documenta junto a la elección, para que la siguiente persona que trabaje en el host sepa que se valoró la opción predeterminada y no se olvidó.

Dónde surgen los problemas

Docker facilita el primer contenedor. Los problemas llegan después, normalmente en una de estas cuatro formas.

Un puerto que bloquea el cortafuegos es accesible desde internet. De forma predeterminada, un puerto publicado se vincula a todas las direcciones del host, y las reglas de filtrado de paquetes que escribe Docker envían ese tráfico al contenedor antes de que una herramienta como ufw llegue a verlo. El cortafuegos informa de que el puerto está cerrado cuando está abierto. Si solo un proxy inverso local necesita el contenedor, publique en la dirección de bucle local:

```
# reachable from anywhere, whatever ufw reports
docker run -d -p 8080:80 nginx:1.27

# reachable only from the host itself
docker run -d -p 127.0.0.1:8080:80 nginx:1.27
```

Los datos desaparecen con el contenedor. Todo lo escrito en su propia capa de escritura se elimina al retirar el contenedor, que es precisamente lo que hace una actualización. Las bases de datos, los archivos subidos y cualquier otra cosa que deba sobrevivir se guardan en un volumen con nombre o un montaje de enlace; ese volumen necesita después su propia copia de seguridad.

Una reconstrucción produce algo distinto a la semana pasada. Una etiqueta como `python:3` o `latest` es un puntero móvil, así que el mismo Dockerfile puede construir otra imagen otro día. Fije las imágenes base a una versión concreta o a un resumen criptográfico cuando la construcción deba ser exactamente reproducible, y reconstruya deliberadamente cuando la imagen base publique correcciones de seguridad.

Una credencial viaja dentro de la imagen. Los argumentos de construcción y las variables de entorno persisten en la imagen final, y un archivo copiado en una capa permanece en ella aunque una instrucción posterior lo elimine. Quien pueda descargar la imagen puede leerlo. Pase los secretos a la construcción mediante un montaje de secretos, que los expone a un solo paso `RUN` sin escribirlos en ninguna capa.

Lecturas adicionales

- Documentación de Docker — Los manuales, las guías y la referencia oficiales del motor, la línea de comandos y Compose.
- Visión general de Docker — El cliente, el demonio, las imágenes y los contenedores explicados por el propio proyecto.

- Publicación y asignación de puertos — Por qué los puertos publicados quedan abiertos al exterior de forma predeterminada y cómo vincularlos a localhost.
- Filtrado de paquetes y cortafuegos — Cómo escribe Docker las reglas de iptables o nftables y dónde deja ufw de ver el tráfico.
- Secretos de construcción — Cómo pasar credenciales a una construcción sin dejarlas en la imagen.
- Especificación del formato de imagen de OCI — El estándar abierto que permite ejecutar la misma imagen en Docker, Podman y containerd.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es Docker?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/docker/