Docker

> ⚠ **DOCUMENTO EN DESARROLLO** ⚠

Overview

Docker permite empaquetar las aplicaciones en **contenedores** que incluyen todo lo necesario para que se puedan ejecutar en un **entorno aislado**. Cada contenedor almacena el código fuente de la aplicación, los archivos de configuración y todas las dependencias de software que necesita. Esta estrategia permite que las aplicaciones se puedan ejecutar de la misma manera sobre cualquier infraestructura que tenga soporte para Docker, tanto de forma local como en la nube.

Con la tecnología de contenedores para aplicaciones, ya no es necesario preocuparse por el software que está instalado en la máquina donde se ejecuta el contenedor, porque **todo lo que la aplicación necesita** está incluido dentro del propio contenedor. Esta forma de trabajar resuelve el problema de *'funciona en mi máquina'*, donde una aplicación puede funcionar correctamente en el entorno de desarrollo, pero tiene errores en el entorno de producción porque los dos entornos no son idénticos y contienen versiones de software diferentes.

Cada vez hay más equipos de desarrollo y operaciones que están utilizando la tecnología de contenedores Docker en sus flujos de trabajo. Esto ha permitido acelerar el proceso de desarrollo de las aplicaciones, ha facilitado la forma de distribuirlas y ha acelerado la automatización del despliegue en producción.

Máquina virtual vs contenedor

Una **máquina virtual (VM)** es un entorno que emula la misma funcionalidad de una máquina física. Una máquina virtual utiliza los **recursos** que se le han asignado, como por ejemplo su propia CPU, memoria, interfaz de red, almacenamiento y su propio sistema operativo.

Las máquinas virtuales (VM) se crean y se ejecutan sobre un software llamado **hipervisor** o ***virtual machine monitor*** **(VMM)**. El hipervisor se ejecuta en la máquina física y actúa como una **capa intermedia** entre el hardware de la máquina anfitriona o *'host'* y la máquina virtual. El hipervisor se encarga de **gestionar y distribuir** los recursos de la máquina física entre las máquinas virtuales.

Es posible crear varias máquinas virtuales sobre una misma máquina física. Cada una de las máquinas virtuales estará aislada del resto, tendrá sus propios recursos y contará con su **propio sistema operativo**, que no necesariamente será el mismo que el de la máquina anfitriona.

Un **contenedor** se puede definir como una **unidad estándar de software** que permite empaquetar el código fuente de una aplicación y todas sus dependencias, para que se pueda **distribuir y ejecutar** de forma rápida y fiable en diferentes entornos.

También se puede definir como un proceso que ha sido aislado de todos los demás procesos de la máquina anfitriona en la que está ejecutando. Aunque es posible tener más de un proceso en un contenedor, las buenas prácticas recomiendan ejecutar solo un proceso por contenedor.

Los contenedores deben cumplir con los **estándares abiertos** de la industria de los contenedores software desarrollados por la **OCI (*Open Container Initiative*)**.

La principal diferencia entre una máquina virtual y un contenedor es que la **máquina virtual necesita un sistema operativo** completo para poder funcionar mientras que un **contenedor no lo necesita** ya que comparte el kernel del sistema operativo de la máquina en la que se está ejecutando.

Por lo tanto, **los contenedores requieren menos recursos** que las máquinas virtuales. Con el mismo hardware es posible tener un mayor número de contenedores que de máquinas virtuales.

Además, los contenedores son más livianos y arrancan más rápido que las máquinas virtuales.

Por último, **un contenedor se puede ejecutar dentro de una máquina virtual** pero no al revés.

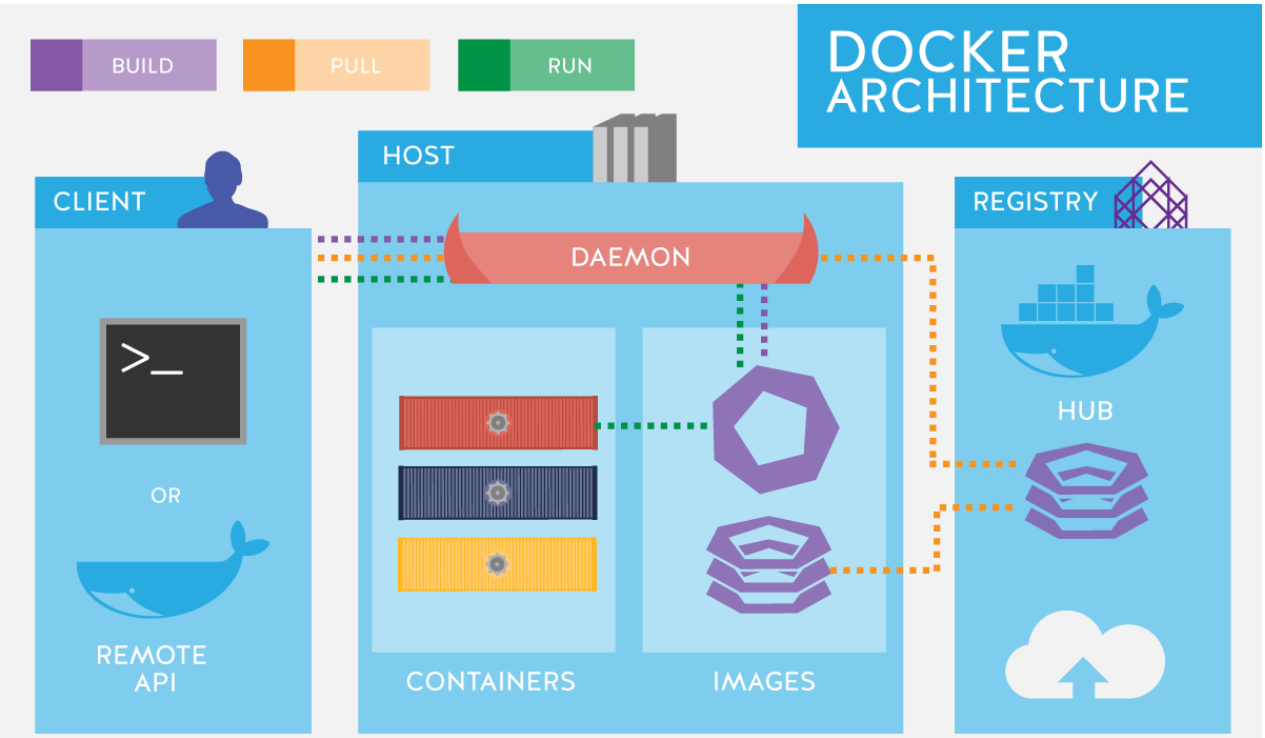
La arquitectura Docker

La arquitectura de Docker fue diseñada inicialmente de forma monolítica, pero más tarde fue rediseñada hacia una **arquitectura modular**, compuesta por diferentes componentes que pueden ser reemplazados e incluso utilizados en otros proyectos.

Cada uno de los componentes de Docker se desarrolla por separado y muchos de ellos forman parte del **"Proyecto Moby"**, un proyecto *open source* creado por la compañía Docker Inc. en 2017, en el cual se desarrollan componentes y herramientas que pueden ser utilizadas para crear productos basados en la tecnología de contenedores.

Los principales componentes de Docker que debemos conocer son:

- Cliente Docker
  - Docker CLI
  - Docker Compose
- Docker Engine
  - Docker Engine API
  - Docker Daemon
- Container Runtime
  - Containerd
  - Runc
- Docker Registry



Cliente Docker

Docker utiliza una arquitectura **cliente-servidor**, donde una aplicación cliente interactúa con un servicio llamado ***Docker Daemon***. Un mismo cliente puede comunicarse con más de un servicio ***Docker Daemon***.

La comunicación entre cliente y servidor se realiza a través de una API HTTP conocida como ***Docker Engine API***.

Las aplicaciones oficiales que se pueden utilizar como cliente son **Docker CLI (*'Command Line Interface'*)** y ***Docker Compose*** aunque cualquier aplicación cliente que haga uso de la API de ***Docker Engine*** puede ser un cliente válido.

El cliente y el servidor se pueden ejecutar en la **misma máquina** o en **máquinas separadas**. Cuando están en la misma máquina, la comunicación entre ambos se realiza a través de un **socket IPC** o un **socket TCP**. En cambio, cuando se encuentran en máquinas separadas, la comunicación se realiza mediante un **socket TCP**.

Docker CLI

Docker CLI es el cliente oficial de Docker. Es una interfaz de línea de comandos que permite a los usuarios interactuar con el servicio ***Docker Daemon***.

```
# Muestra la ayuda de Docker
$ docker help
```

El uso más habitual de ***Docker CLI*** es cuando se desea interactuar con un **único contenedor**.

Docker Compose

Docker Compose es una aplicación utilizada desde la línea de comandos que permite a los usuarios interactuar con el servicio ***Docker daemon***.

```
# Muestra la ayuda de Docker Compose
$ docker compose help
```

Esta aplicación permite definir y ejecutar aplicaciones con **múltiples contenedores**. Utiliza un archivo de configuración con formato YAML para definir los servicios, las redes y los volúmenes que componen la aplicación que se desea ejecutar.

Una de las ventajas que ofrece ***Docker Compose*** es que basta con ejecutar un solo comando para crear y ejecutar todos los servicios definidos en el archivo de configuración en formato YAML.

En la actualidad existen dos versiones de ***Docker Compose***:

- **v1**: debe instalarse como una herramienta adicional y se ejecuta con `docker-compose`
- **v2**: integra el comando `compose` dentro del cliente oficial de ***Docker CLI***. Por lo tanto, la nueva versión se ejecuta con `docker compose`

Docker Engine

Docker Engine es el componente principal de Docker, responsable de crear, ejecutar y gestionar contenedores.

Tiene un diseño **modular** y está compuesto por varios componentes que cumplen con los estándares abiertos de la **OCI (*Open Container Initiative*)**:

- *Docker Engine API*
- *Docker Daemon*
- *Container runtime* (*containerd*)
- Gestión de redes (*libnetwork*)
- Creación de imágenes (*buildkit*)

- Interacción con los registros de contenedores (*distribution*)
- Soporte nativo para la orquestación de contenedores con Docker Swarm (*swarmkit*)
- Gestión de plugins

> ⚠ Estos son proyectos que cumplen los estándares abiertos y están alojados en **"Moby Project"**.

Docker Engine se ejecuta de forma nativa en los sistemas Linux y Windows Server. En otros sistemas operativos de Windows y en macOS, se ejecuta sobre una máquina virtual Linux.

Docker Engine API

(TODO)

Docker daemon

(TODO)

Container runtime

containerd

(TODO)

runc

(TODO)

Docker Registry

(TODO)

Objetos de Docker

Los principales objetos de Docker son:

- Imágenes
- Contenedores
- Volúmenes
- Redes

Imágenes

Las imágenes son **plantillas inmutables** que contienen el sistema de archivos y los parámetros necesarios para configurar un contenedor. Representan el estado inicial del sistema de archivos raíz del contenedor y pueden incluir aplicaciones preinstaladas, configuraciones y dependencias.

Las imágenes se pueden crear manualmente utilizando un archivo llamado `Dockerfile`, que contiene una serie de instrucciones para ensamblar la imagen paso a paso. También es posible obtener imágenes predefinidas desde registros públicos como ***Docker Hub***.

Docker utiliza un **sistema de capas** para construir imágenes, lo que permite reutilizar partes comunes entre diferentes imágenes y optimizar el almacenamiento y la transferencia.

A partir de una misma imagen, se pueden **crear múltiples contenedores**, lo que facilita la escalabilidad y consistencia en la implementación de aplicaciones.

Contenedores

Un contenedor es una **instancia ejecutable de una imagen**. Representa una unidad de software ligera y autónoma que incluye todo lo necesario para ejecutar una aplicación.

Puede crear, iniciar, detener, mover o eliminar un contenedor utilizando la API o la CLI de Docker. Además, es posible conectar un contenedor a una o más redes, adjuntar almacenamiento persistente o incluso crear una nueva imagen basada en el estado actual del contenedor mediante la ejecución de comandos como `docker commit`.

De forma predeterminada, un contenedor está relativamente **aislado** de otros contenedores y de la máquina *host*. Este aislamiento incluye el sistema de archivos, la red y otros recursos del sistema. Sin embargo, Docker proporciona opciones para controlar el nivel de aislamiento según las necesidades específicas.

Es importante notar que, a menos que se utilicen volúmenes o montajes de enlaces, los cambios realizados dentro de un contenedor no se preservan una vez que el contenedor se elimina.

Volúmenes

Los volúmenes son el mecanismo que ofrece Docker para gestionar la **persistencia de datos** en los contenedores. Permiten almacenar datos fuera del sistema de archivos del contenedor, asegurando que la información persista incluso si el contenedor se elimina.

Los volúmenes se pueden crear y gestionar utilizando comandos de Docker, como `docker volume create`. Una vez creados, pueden ser montados en uno o varios contenedores.

El ciclo de vida de los volúmenes es independiente del ciclo de vida de los contenedores. Esto significa que los volúmenes permanecen **intactos y disponibles** para otros contenedores incluso después de que los contenedores que los utilizaban hayan sido eliminados.

Además de los volúmenes, Docker también soporta montajes de enlaces (*'bind mounts'*), que permiten montar directorios o archivos específicos del sistema de archivos de la máquina host dentro de un contenedor.

Redes

Docker proporciona capacidades avanzadas para la creación y gestión de redes, facilitando la **comunicación entre contenedores y con sistemas externos**.

Docker soporta varios tipos de redes, incluyendo:

- ***Bridge*** → Es la red por defecto en Docker. Permite la comunicación entre contenedores en la misma máquina host.
- ***Host*** → Utiliza la pila de red de la máquina host, permitiendo que los contenedores compartan la misma dirección IP y puertos que el host.
- ***Overlay*** → Permite la comunicación entre contenedores que residen en diferentes máquinas host, comúnmente utilizada en configuraciones de clústeres y orquestación.
- ***Macvlan*** → Asigna una dirección MAC a cada contenedor, permitiendo que aparezcan como dispositivos físicos en la red.

A través de comandos como `docker network create`, `docker network connect` y `docker network inspect`, los usuarios pueden crear redes personalizadas, conectar contenedores a ellas y obtener información detallada sobre su configuración.

Appendix: Common Docker Commands

IMAGES

Docker images are a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings

```
# Buscar una imagen en Docker Hub
docker search ${SEARCH_TERM}

# Descargar una imagen (por ejemplo Docker Hub)
docker image pull ${IMAGE}

# Construir una imagen a partir de un Dockerfile
docker image build --rm=true -t my-image:tag .

# Listar todas las imágenes instaladas localmente
docker image ls

# Listar imágenes instaladas (listado detallado)
docker image ls --no-trunc

# Mostrar información detallada sobre una imagen específica
docker image inspect ${IMAGE_ID}

# Ver historial de instrucciones de una imagen
docker image history ${IMAGE_ID}

# Etiquetar una imagen existente
docker tag ${SOURCE_IMAGE} ${TARGET_IMAGE}

# Eliminar una imagen específica
docker image rm ${IMAGE_ID}

# Eliminar todas las imágenes
docker image rm $(docker image ls -aq)

# Eliminar imágenes no utilizadas (sin contenedores asociados)
docker image prune

# Eliminar imágenes no utilizadas y sin etiqueta
docker image prune -a

# Eliminar imágenes intermedias (build cache)
docker builder prune
```

CONTAINERS

```
# Ejecutar un contenedor
docker container run -d --name my-container -p 80:80 my-image

# Listar contenedores en ejecución
docker container ls

# Listar todos los contenedores
docker container ls --all

# Listar todos los contenedores que han salido con un estado específico
docker container ls -a --filter "status=exited"

# Ver detalles del contenedor
docker container inspect ${CONTAINER_ID}

# Encontrar la IP de un contenedor
```

```
docker container inspect --format '{{ .NetworkSettings.IPAddress }}' ${CONTAINER_ID}

# Obtener el ID del contenedor para una imagen específica usando una expresión regular
docker container ls | grep wildfly | awk '{print $1}'

# Mostrar logs
docker container logs ${CONTAINER_ID}

# Mostrar últimos logs
docker container logs --tail 100 ${CONTAINER_ID}

# Ver estadísticas en tiempo real
docker container stats ${CONTAINER_ID}

# Abrir una shell interactiva en un contenedor
docker container exec -it ${CONTAINER_ID} bash

# Ejecutar un comando en el contenedor
docker container exec ${CONTAINER_ID} <command>

# Renombrar un contenedor
docker container rename ${OLD_NAME} ${NEW_NAME}

# Conectar a un contenedor
docker container attach ${CONTAINER_ID}

# Detener un contenedor
docker container stop ${CONTAINER_ID}

# Detener todos los contenedores en ejecución
docker container stop $(docker container ls -q)

# Eliminar un contenedor
docker container rm ${CONTAINER_ID}

# Eliminar contenedores que coincidan con una expresión regular
docker container ls -a | grep wildfly | awk '{print $1}' | xargs docker container rm -f

# Eliminar todos los contenedores que han salido
docker container rm -f $(docker container ls -a --filter "status=exited" -q)

# Eliminar todos los contenedores
docker container rm $(docker container ls -aq)
```

VOLUME

```
# Eliminar volúmenes huérfanos (_'dangling'_)
docker volume rm $(docker volume ls -qf dangling=true)
```

OTHERS

```
# Mostrar la versión de Docker
docker --version

# Mostrar la versión del cliente y del servidor
docker version

# Proporciona un resumen completo del estado del daemon de Docker
docker info

# Comprobar y diagnosticar problemas de Docker
docker system info

# Muestra los registros del servicio de Docker en sistemas basados en 'systemd'
```

```
journalctl -u docker

# Muestra el uso de espacio en disco por imágenes, contenedores, y volúmenes
docker system df

# Limpieza general del sistema
docker system prune

# Iniciar el daemon manualmente
dockerd

# Verifica el estado del servicio Docker en sistemas que usan 'systemd'
sudo systemctl status docker

# Reinicia el servicio Docker  en sistemas que usan 'systemd'
sudo systemctl restart docker

# Permite especificar un archivo de configuración personalizado para el daemon de Docker
docker daemon --config-file /path/to/config.json

# Actualizar Docker a la última versión disponible
sudo apt-get update && sudo apt-get upgrade docker-ce
```

Referencias

- https://docs.docker.com/
- https://docs.docker.com/language/java/
- https://cheatsheets.zip/docker
- https://github.com/wsargent/docker-cheat-sheet/tree/master/es-es
- https://github.com/collabnix/dockerlabs/blob/master/docker/cheatsheet/README.md

Licencia