Dockerfile Avanzado

Arturo Silvelo

Try New Roads

Introducción: ¿Qué es un Dockerfile?

Un **Dockerfile** es un archivo de texto que contiene instrucciones para construir una imagen Docker. Define el sistema operativo base, las dependencias, el código fuente, variables de entorno y el comando de arranque del contenedor.

Permite automatizar la creación de entornos reproducibles y portables, facilitando el despliegue de aplicaciones en cualquier sistema compatible con Docker.

Ejemplo mínimo:

```
FROM alpine:3.20
CMD ["echo", "Hola Docker!"]
```

Optimización de imágenes Docker

Optimizar imágenes Docker es fundamental para mejorar el rendimiento, la seguridad y la eficiencia en el despliegue de aplicaciones. Una imagen optimizada:

- Se descarga y despliega más rápido.
- Ocupa menos espacio en disco y consume menos ancho de banda.
- Reduce la superficie de ataque y facilita la actualización.
- Es más fácil de mantener y depurar.

Para lograrlo, es importante aplicar varias técnicas y buenas prácticas, como el uso de multi-stage builds y la optimización de capas.

¿Cómo optimizar imágenes Docker?

- Usar imágenes base ligeras (alpine, slim)
- Multi-stage builds para separar dependencias de build y producción
- Minimizar el número de capas combinando instrucciones RUN, COPY, etc.
- Eliminar archivos temporales y cachés en la misma instrucción

Multi-stage builds

Permite crear imágenes más ligeras y seguras usando varias etapas en el Dockerfile.

- Solo la última etapa contiene lo necesario para ejecutar la aplicación.
- Las etapas previas pueden tener herramientas de compilación y dependencias de desarrollo, que no se copian a la imagen final.

Ventajas:

- Reduce el tamaño de la imagen final.
- Elimina archivos y dependencias de desarrollo.
- Nombra las etapas (`AS build`, `AS produccion`) para facilitar el copiado selectivo.
- Usa imágenes base "slim" o "alpine" cuando sea posible.

> No es necesario construir todas las fases del Dockerfile. Usando la opción --target, puedes crear una imagen solo hasta una etapa concreta.

Capas en Docker: ¿Qué son?

Cada instrucción (FROM, RUN, COPY, etc.) en un Dockerfile crea una **capa**.

- Las capas son archivos intermedios que Docker almacena en cache.
- Si una capa no cambia, Docker la reutiliza en builds futuros.
- Esto acelera la construcción y ahorra espacio en disco.

Ejemplo:

```
FROM node:20  # Crea una capa
COPY . .      # Otra capa
RUN npm install  # Otra capa
```

Optimización de capas: Estrategias

Optimizar el número y el contenido de las capas en tus imágenes Docker es clave para reducir el tamaño, acelerar los builds y facilitar el mantenimiento.

Algunas estrategias recomendadas son:

- **Agrupa comandos en una sola instrucción RUN:**
  Así reduces el número de capas y aprovechas mejor la cache de Docker.
- **Elimina archivos temporales y de build en la misma RUN:**
  Si los borras en la misma instrucción donde se crean, no quedan guardados en ninguna capa.

- **Incluye un archivo `.dockerignore`:**
  Excluye archivos y carpetas que no necesitas en la imagen (tests, docs, node_modules, etc.), evitando que se copien y generen capas innecesarias.
- **El orden de las capas influye en el tiempo de build:**
  Si modificas una instrucción, todas las capas posteriores se reconstruyen. Coloca primero las instrucciones que cambian menos (por ejemplo, dependencias) y después las que cambian más (código fuente).

Variables ARG y ENV en Docker

Las variables `ARG` y `ENV` permiten personalizar tanto la construcción de la imagen como el comportamiento de los contenedores.

- **ARG** se usa para definir valores que solo existen durante el proceso de build (por ejemplo, elegir una versión de Node o una URL de descarga).
- **ENV** define variables de entorno que estarán disponibles en la imagen final y en los contenedores que se creen a partir de ella.

Estas variables ayudan a crear imágenes más flexibles, reutilizables y adaptables a diferentes entornos (desarrollo, producción, etc.).

Buenas prácticas

- Usa `ARG` para valores temporales o que no deban quedar en la imagen.
- Usa `ENV` para configuración que necesita el contenedor en ejecución.
- No pongas secretos en `ENV` ni en el Dockerfile.
- Documenta las variables y su propósito.

Gestión de secretos en Docker

Los secretos son datos sensibles (contraseñas, tokens, claves API, certificados, etc.) que permiten acceder a recursos protegidos. Es fundamental protegerlos y evitar que queden expuestos en la imagen o el código fuente.

En Docker, los secretos no deben almacenarse en la imagen. Deben gestionarse de forma externa y segura.

Riesgos de una mala gestión

- Los secretos quedan guardados en la imagen y pueden ser leídos por cualquiera con acceso.
- Si subes la imagen a un registro público, los secretos quedan expuestos.
- Los secretos pueden filtrarse en logs, capas intermedias o sistemas de CI/CD.

Buenas prácticas

- Nunca incluyas secretos en el Dockerfile ni en la imagen.
- Usa mecanismos externos: Docker secrets (Swarm), variables de entorno solo en ejecución, archivos montados como volúmenes.
- Usa archivos `.env` solo para desarrollo y nunca los subas a git.
- Documenta cómo inyectar los secretos en producción.