Imágenes

Antonio Espín Herranz

Contenidos

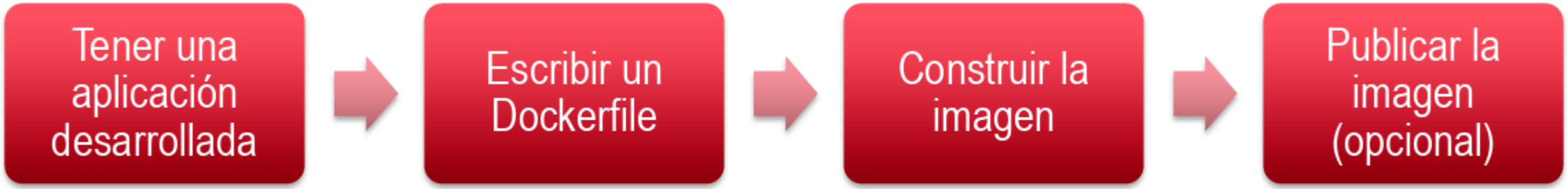
- Introducción.
- Listar imágenes
- Buscar imágenes
- Copias de seguridad
- Histórico de imágenes
- Eliminar/Restaurar imágenes
- Publicar imágenes en repositorio oficial
- Publicar imágenes en repositorio local
- Buenas prácticas Dockerfile

Introducción

- Una imagen es una colección de archivos.
- Se parte de una imagen base y luego se construyen imágenes personalizadas encima.
- Un Dockerfile es un fichero que describe las instrucciones para construir una nueva imagen.
- Las imágenes están en capas y cada capa representa una diferencia de la capa anterior.

Introducción

- Pasos para dockenizar una aplicación, ejemplo de un flujo de trabajo con Docker



Introducción

- La aplicación:
  - Un “Hello world” hecho en Java con Spring MVC
    - Entrando en la ruta /greeting dentro de donde esté el WAR desplegado aparece un “Hola, mundo!”
    - Por ejemplo, si el WAR se despliega en hello-java-0.1.0, la ruta es hello-java0.1.0/greeting/
    - Se empaqueta en un WAR

Introducción

- Construir la imagen y probarla:
  - docker build –t javahello
    - Crear la imagen
  - docker images
    - Listar las imágenes, comprobar si se ha creado
  - docker run –it --rm –p 8080:8080 javahello
    - Crear un contenedor interactivo a partir de la imagen.
    - El contenedor se borrará cuando se pare el contenedor
    - Se mapea en el puerto 8080 tanto en el host de docker como en el contenedor

Introducción

- Publicar la imagen en un Docker hub (este paso es opcional)
    - Registrarse en docker hub
    - docker login
    - docker push mi_usuario/javahello
        - Publicar la imagen en docker hub

Listar imágenes

- **docker images**
  - Nombre de la imagen
  - La etiqueta
  - El id
  - Cuando se ha creado
  - Y cuanto ocupa

```
C:\Users\Anton>docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
mariadb      latest    a914eff5d2eb   3 weeks ago     336MB
nginx        latest    b52e0b094bc0   4 weeks ago     192MB
ubuntu       latest    a04dc4851cbc   5 weeks ago     78.1MB
httpd        alpine    fcd9ece3a2cc   6 weeks ago     63.8MB
httpd        latest    0de612e99135   6 weeks ago     148MB
bash         latest    a6f5c002ec83   2 months ago    14.5MB
```

Listar imágenes

- Con el comando: **docker images --help**

- Podemos ver todas las opciones del comando:
  -a, --all para mostrar todas las imágenes (es la opción que funciona por defecto)
  --digest muestra el digest de cada imagen
  -f, --filter filtrar
  --format
    Se puede mostrar en json → docker images --format json
  --no-trunc No trunca la salida
  -q, --quiet Sólo muestra los Ids
  --tree es experimental (muestra colores)

- https://docs.docker.com/reference/cli/docker/image/ls/

Buscar imágenes

- Comando: docker search <palabra_clave>
- Ejemplo: **docker search nginx**
- Devuelve todas las imágenes de nginx, con 4 columnas:

```
C:\Users\Anton>docker search nginx
NAME                                DESCRIPTION                                     STARS     OFFICIAL
nginx                               Official build of Nginx.                        20688     [OK]
nginx/nginx-ingress                 NGINX and  NGINX Plus Ingress Controllers fo…   100
nginx/unit                          This repository is retired, use the Docker o…   65
nginx/nginx-prometheus-exporter     NGINX Prometheus Exporter for NGINX and NGIN…   48
nginx/nginx-ingress-operator        NGINX Ingress Operator for NGINX and NGINX P…   2
nginx/nginx-quic-qns                NGINX QUIC interop                              1
nginx/nginxaas-loadbalancer-kubernetes                                              0
```

Buscar imágenes

- Se pueden buscar imágenes oficiales:

```
C:\Users\Anton>docker search --filter "is-official=true" nginx
NAME      DESCRIPTION                     STARS     OFFICIAL
nginx     Official build of Nginx.        20688     [OK]
```

- Se pueden buscar imágenes que tengan al menos 10 estrellas

```
docker search --filter "stars=10" ubuntu
```

- Filtros con dos condiciones: oficiales y al menos con 50 estrellas

```
docker search --filter "is-official=true" --filter "stars=50" nginx
```

Copias de seguridad

- Crea un fichero **tar** con la imagen:
  - docker save –o fichero.tar nombre_imagen
  - **docker save –o backup.tar nginx**

- Cargar la imagen respaldada:
  - **docker load –i backup.tar**

- Crear un nuevo contenedor de la imagen restaurada:
  - **docker run -it <new_image_name>**

Histórico de imágenes

- Comando: **docker history id_img**
- docker history b52e (vale con los 4 primeros caracteres)

```
C:\Users\Anton>docker history b52e
IMAGE          CREATED       CREATED BY                                      SIZE      COMMENT
b52e0b094bc0   4 weeks ago   CMD ["nginx" "-g" "daemon off;"]                0B        buildkit.dockerfile.v0
<missing>      4 weeks ago   STOPSIGNAL SIGQUIT                              0B        buildkit.dockerfile.v0
<missing>      4 weeks ago   EXPOSE map[80/tcp:{}]                           0B        buildkit.dockerfile.v0
<missing>      4 weeks ago   ENTRYPOINT ["/docker-entrypoint.sh"]            0B        buildkit.dockerfile.v0
<missing>      4 weeks ago   COPY 30-tune-worker-processes.sh /docker-ent…   4.62kB    buildkit.dockerfile.v0
<missing>      4 weeks ago   COPY 20-envsubst-on-templates.sh /docker-ent…   3.02kB    buildkit.dockerfile.v0
<missing>      4 weeks ago   COPY 15-local-resolvers.envsh /docker-entryp…   389B      buildkit.dockerfile.v0
<missing>      4 weeks ago   COPY 10-listen-on-ipv6-by-default.sh /docker…   2.12kB    buildkit.dockerfile.v0
<missing>      4 weeks ago   COPY docker-entrypoint.sh / # buildkit          1.62kB    buildkit.dockerfile.v0
<missing>      4 weeks ago   RUN /bin/sh -c set -x     && groupadd --syst…   117MB     buildkit.dockerfile.v0
<missing>      4 weeks ago   ENV DYNPKG_RELEASE=1~bookworm                   0B        buildkit.dockerfile.v0
<missing>      4 weeks ago   ENV PKG_RELEASE=1~bookworm                      0B        buildkit.dockerfile.v0
<missing>      4 weeks ago   ENV NJS_RELEASE=1~bookworm                      0B        buildkit.dockerfile.v0
<missing>      4 weeks ago   ENV NJS_VERSION=0.8.9                           0B        buildkit.dockerfile.v0
<missing>      4 weeks ago   ENV NGINX_VERSION=1.27.4                        0B        buildkit.dockerfile.v0
<missing>      4 weeks ago   LABEL maintainer=NGINX Docker Maintainers <d…   0B        buildkit.dockerfile.v0
<missing>      4 weeks ago   # debian.sh --arch 'amd64' out/ 'bookworm' '…   74.8MB    debuerreotype 0.15
```

Eliminar / Restaurar imágenes

- Para borrar una imagen:
  - docker rmi <id_imagen>
  - **docker rmi nginx:lastest**

  - Si está utilizada por un contenedor
  - **docker rmi –f <image_id>**

  - Borrar imágenes no utilizadas:
  - **docker image prune**

  - Borra varias: **docker rmi id_img1 id_img2 id_img3**
  - **docker rmi $(docker images –q)**

- Para restaurar la imagen a partir del fichero **tar**:
  - docker load –i fichero.tar
  - **docker load –i backup.tar**

Convertir un contenedor en una imagen

- **Pasos**:
  1. Buscar el contenedor
  2. Crear la imagen del contenedor (commit)
  3. Verificar si está la imagen nueva
  4. Utilizar la nueva imagen

- Mejor utilizar un **dockerfile**

- **Ejemplo**
  1. docker ps –a
  2. docker commit id_cont nombre_img:etiqueta
     docker commit 123abc mi-imagen:1.0
  3. docker images
  4. docker run –d mi-imagen:1.0

Convertir un contenedor en una imagen

- Se puede añadir un mensaje de información y el autor en el comando commit

- docker **commit -m** "Configuración personalizada" **-a** "Autor" <container_id> <nombre_de_imagen>

Etiquetado

- Para añadir etiquetas a las imágenes:
  - Listar las imágenes así obtenemos el ID de la imagen.
  - docker **tag** <imagen_id> <nombre>:<etiqueta>

  - **docker tag 123abc mi-aplicación:v1.0**

  - Si no se indica la etiqueta de la imagen, docker asume que es **latest**.

Publicar imágenes en repositorio Global

- En **Docker Hub** la imagen tiene que venir precedida del nombre del usuario.
  - docker build –t usuario/image:v1 **.**

- **Va a Docker Hub**
  - **docker login**
  - Nos pedirá user / pass
  - Para subir la imagen:
  - **docker push usuario/image:v1**
  - **Comprobar si está en Docker Hub**
  - Para descargar: **docker pull usuario/image:v1**

Publicar imágenes en repositorio Local

- Un registro Docker propio, usando **Docker Registry**.

- Un registro privado en una plataforma como Azure Container Registry o AWS Elastic Container Registry.

- Si usas **Docker Registry** para crear tu propio registro, ejecuta este comando para iniciarlo:

Publicar imágenes en repositorio Local

- Antes de subirla hay que etiquetarla con la URL del registro privado:
  - docker tag <nombre_imagen>:<etiqueta> <url_registro>/<nombre_imagen>:<etiqueta>

- Ejemplo
  - docker tag miaplicacion:latest localhost:5000/miaplicacion:latest

Publicar imágenes en repositorio Local

- Subirla con push:
  - **docker push <url_registro>/<nombre_imagen>:<etiqueta>**

- Ejemplo:
  - **docker push localhost:5000/miaplicacion:latest**

- Verificar la imagen:
  - **curl http://localhost:5000/v2/_catalog**

- Para descargar la imagen:
  - **docker pull <url_registro>/<nombre_imagen>:<etiqueta>**

Publicar imágenes en repositorio Local

- Si el registro privado requiere autenticación, **iniciar sesión** antes de subir imágenes:
  - docker login <url_registro>
  - Pedirá un **usuario** y una **contraseña** configurados en el registro.

Dockerfile

- Es un fichero de texto que se **utiliza para definir la imagen de un contenedor Docker**.

- Se compone de instrucciones para construir una imagen personalizada.

- Dockerfile nos sirve para:
  - Desarrollar aplicaciones: Crear un entorno estandarizado con las versiones exactas de software y dependencias necesarias para ejecutar tu aplicación.
  - Automatizar la configuración y construcción de ent
  - Microservicios: Diseñar imágenes ligeras y optimizadas para cada servicio.
  - Testing y debugging: Configurar entornos de prueba controlados.

DockerFile comandos I

`FROM` : Especifica la imagen base que se usará para construir la nueva imagen. Por ejemplo, `FROM ubuntu:20.04` .

`WORKDIR` : Define el directorio de trabajo dentro del contenedor donde se ejecutarán los comandos posteriores. Ejemplo: `WORKDIR /app` .

`COPY` : Copia archivos o directorios desde el sistema host al contenedor. Ejemplo: `COPY . /app` .

`ADD` : Similar a `COPY` , pero también puede descargar archivos de URL y descomprimir archivos. Ejemplo: `ADD app.tar.gz /app` .

`RUN` : Ejecuta comandos durante la construcción de la imagen, como instalar dependencias. Ejemplo: `RUN apt-get update && apt-get install -y python3` .

`CMD` : Define el comando por defecto que se ejecutará al iniciar el contenedor. Ejemplo: `CMD ["python", "app.py"]` .

`ENTRYPOINT` : Similar a `CMD` , pero más fijo y permite agregar argumentos. Ejemplo: `ENTRYPOINT ["python"]` .

DockerFile comandos II

`ENV` : Configura variables de entorno dentro del contenedor. Ejemplo: `ENV PORT=8080` .

`EXPOSE` : Indica los puertos en los que el contenedor está configurado para escuchar. Ejemplo: `EXPOSE 8080` .

`VOLUME` : Define puntos de montaje para datos persistentes. Ejemplo: `VOLUME /data` .

`ARG` : Permite pasar variables al construir la imagen. Ejemplo: `ARG VERSION=1.0` .

`LABEL` : Agrega metadatos a la imagen, como autor o versión. Ejemplo: `LABEL maintainer="tuemail@example.com"` .

`USER` : Cambia al usuario que ejecutará los comandos dentro del contenedor. Ejemplo: `USER appuser` .

`ONBUILD` : Especifica instrucciones que se ejecutarán al usar la imagen como base en otro Dockerfile.

Ejemplo

```
# Imagen base
FROM python:3.9-slim

# Establecer el directorio de trabajo dentro del contenedor
WORKDIR /app

# Copiar el archivo de dependencias (requirements.txt) al contenedor
COPY requirements.txt .

# Instalar las dependencias requeridas
RUN pip install --no-cache-dir -r requirements.txt

# Copiar los archivos de la aplicación al contenedor
COPY . .

# Especificar el comando para ejecutar la aplicación
CMD ["python", "app.py"]
```

FROM python:3.9-slim: Usamos una imagen base ligera de Python.

WORKDIR /app: Configura el directorio de trabajo dentro del contenedor.

COPY requirements.txt .: Copia el archivo `requirements.txt` para instalar las dependencias.

RUN pip install: Instala las librerías definidas en `requirements.txt`.

COPY . .: Copia todos los archivos del directorio actual al contenedor.

CMD: Define el comando por defecto para ejecutar la aplicación (`app.py` en este caso).

El fichero requirement.txt contendrá los nombres de las librerías de Python habría que instalar en el contenedor:
- pandas
- Flask
Etc.

Dockerfile (ENTRYPOINT vs CMD)

- **ENTRYPOINT**

  - Propósito: **Define el programa principal que siempre se ejecutará en el contenedor**. Es el “punto de entrada” fijo.

  - Formato: Puede escribirse en dos formas:
    - **Lista** ejecutable: ENTRYPOINT ["executable", "param1", "param2"] (forma recomendada).
    - Forma de **cadena**: ENTRYPOINT command param1 param2 (menos común y limitada a /bin/sh).

Dockerfile (ENTRYPOINT vs CMD)

- **CMD**

  - Propósito: **Especifica el comando por defecto que se ejecutará cuando se inicie el contenedor**. Puede ser sobrescrito al ejecutar el contenedor.

  - Formato: También puede usarse en dos formas:
    - Lista ejecutable: CMD ["executable", "param1", "param2"].
    - Forma de cadena: CMD command param1 param2 (limitada a /bin/sh).

  - Comportamiento:
    - Es más flexible que ENTRYPOINT, ya que puedes **sobrescribir** el **comando** al ejecutar el contenedor.
    - Si ejecutamos: docker run imagen ls (ejecuta ls en vez del comando especificado en CMD)

Buenas prácticas en un DockerFile I

- Utilizar las **imágenes ligeras** siempre que se pueda:
  - Utilizar alpine o las que terminan en Slim
  - FROM python:**3.9-slim mejor que** FROM python:3.9

- **Minimizar** el número de **capas** de la imagen. Cada instrucción ADD, COPY, etc. Crea una nueva capa.
  - Por ejemplo, se pueden unir comandos → RUN apt-get update && apt-get install –y curl …

- **Eliminar** archivos **temporales**

- Se puede utilizar **.dockerignore** (se pueden excluir archivos innecesarios)

Buenas prácticas en un DockerFile II

- Fijar **versiones** de las **librerías**: flask==2.1.0 en un archivo: requirements.txt (por ejemplo, en Python)

- Utilizar mejor **COPY** que ADD. ADD permite descomprimir archivos.

- Evitar el usuario root. Añadir otros usuarios (por seguridad):
  - **RUN** useradd –m appuser
  - **USER** appuser

- **Eliminar** las herramientas innecesarias (que no tengan sentido en producción)

- Utilizar **comentarios** dentro del fichero.
  - Las líneas se preceden con una **#**

Apéndice

- **1. docker build** – To build Docker Image from Dockerfile
- **2. docker pull** – To pull Docker Image from Docker Hub Registry
- **3. docker tag** – To add Tag to Docker Image
- 4. **docker images** – To list Docker Images
- **5. docker push** – To push Docker Images to repository
- 6. **docker history –** To show history of Docker Image

Apéndice II

- **7. docker inspect**– To show complete information in JSON format
- **8. docker save** – To save an existing Docker Image
- **9. docker import** – Create Docker Image from Tarball
- **10. docker export** – To export existing Docker container
- **11. docker load**– To load Docker Image from file or archives
- **12. docker rmi**– To remove docker images