Contenido

Comandos Básicos de Docker Contenedores

docker run

https://docs.docker.com/reference/cli/docker/container/run/

Crea y ejecuta un nuevo contenedor a partir de una imagen.

--name nombre_contenedor

--host nombre_host

-d ejecución en segundo plano

-it sesión interactiva

-e USUARIO=prueba : variables entorno

docker run -d --name my-apache-app -p 8080:80 httpd:2.4

docker create

Crea un contenedor sin ejecutarlo a partir de una imagen.

docker create --name mi_app_detenida nginx

docker ps

https://docs.docker.com/reference/cli/docker/container/ls/

Lista los contenedores en ejecución. Añade -a para ver todos, incluyendo los detenidos.

docker ps -a muestra también aquellos que están detenidos.

docker start

https://docs.docker.com/reference/cli/docker/container/start/

Inicia la ejecución de un contenedor.

-a → ver en nuestro terminal la salida

docker start -a mi_app_detenida (El parámetro -a permite ver la salida en el terminal)

docker stop

https://docs.docker.com/reference/cli/docker/container/stop/

Detiene uno o más contenedores en ejecución suavemente.

docker stop my-apache-app

Parar todos los contenedores: **docker stop $(docker ps -a -q)**

docker restart

https://docs.docker.com/reference/cli/docker/container/restart/

Reiniciar contenedor

docker restart my-apache-app

docker pause

https://docs.docker.com/reference/cli/docker/container/pause/

Pausa la ejecución de un contenedor

docker pause my-apache-app

docker unpause

https://docs.docker.com/reference/cli/docker/container/unpause/

Continua la ejecución después de estar parado

docker unpause my-apache-app

docker rm

https://docs.docker.com/reference/cli/docker/container/rm/

Elimina un contenedor

docker rm mi_app_detenida

docker rm -f [nombre_o_ID_del_contenedor]

Borrar todos los contenedores: **docker rm $(docker ps -a -q)**

docker rename

https://docs.docker.com/reference/cli/docker/container/rename/

Renombra nombre del contenedor

docker rename viejo_nombre nuevo_nombre

docker rename prueba01 prueba10

docker attach

https://docs.docker.com/reference/cli/docker/container/attach/

nos conectamos a la entrada estándar y a la salida estándar y de error de un contenedor en ejecución

docker attach my-apache-app

docker events

https://docs.docker.com/reference/cli/docker/system/events/

Ver operaciones que se producen en los contenedores

docker logs

https://docs.docker.com/reference/cli/docker/container/logs/

Ver los log de un contenedor en ejecución

docker logs my-apache-app

docker exec

https://docs.docker.com/reference/cli/docker/container/exec/

Ejecución comandos en contenedor

docker exec my-apache-app ls

docker exec -it my-apache-app bash

docker cp

https://docs.docker.com/reference/cli/docker/container/cp/

Copia ficheros a un contenedor

docker cp mi_archivo.txt my-apache-app:/destino/

docker cp ./some_file CONTAINER:/work

docker top

https://docs.docker.com/reference/cli/docker/container/top/

Visualiza procesos que se ejecutan en un contenedor

docker top my-apache-app

docker inspect

https://docs.docker.com/reference/cli/docker/inspect/

Información detallada de un contenedor

docker inspect my-apache-app

Comandos Básicos de Docker Imágenes

Dominar estos comandos es fundamental para interactuar eficazmente con Docker y gestionar tus contenedores e imágenes.

docker build

https://docs.docker.com/reference/cli/docker/buildx/build/

Construye una imagen Docker a partir de un Dockerfile y un contexto especificado.

docker build -t nombre_de_la_imagen:etiqueta .

docker build -t mi_imagen:v1.0 . (El parámetro -t asigna un nombre y etiqueta)

docker images

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

Muestra una lista de todas las imágenes Docker almacenadas localmente en tu sistema.

docker pull

https://docs.docker.com/reference/cli/docker/image/pull/

Descarga una imagen

docker pull nextcloud

docker images

Visualizar imagenes de nuestro registro

docker rmi

https://docs.docker.com/reference/cli/docker/image/rm/

Eliminar imagen

docker rmi -f [nombre_o_ID_de_la_imagen]

docker inspect

https://docs.docker.com/reference/cli/docker/inspect/

Obtener información detallada de una imagen

docker inspect httpd:2.4

Almacenamiento – Volúmenes

Permiten almacenar datos en el sistema de archivos del host, gestionados por Docker.

Al montar un volumen, un directorio dentro del contenedor se mapea a un directorio en el host

Persistencia de datos

Los datos en los volúmenes persisten independientemente del ciclo de vida del contenedor.

Compartir datos

Los volúmenes pueden compartirse entre múltiples contenedores fácilmente.

Copias de seguridad y migración

Facilitan la creación de copias de seguridad y la migración de datos entre hosts.

Por defecto, los volúmenes se almacenan en una parte del sistema de archivos del host gestionada por Docker (usualmente en /var/lib/docker/volumes/ en sistemas Linux).

Comandos Básicos de Docker Volúmenes

docker volume create

https://docs.docker.com/reference/cli/docker/volume/create/

Crear volumen

docker volume create nextcloud_storage

docker volume ls

https://docs.docker.com/reference/cli/docker/volume/ls/

Muestra una lista de todos los contedores.

docker volume inspect

https://docs.docker.com/reference/cli/docker/volume/inspect/

Obtener información detallada de un volumen

docker volume inspect mi_volumen_datos

docker volume rm

https://docs.docker.com/reference/cli/docker/volume/rm/

Eliminar volumen

docker volume rm mi_volumen_datos

Añadir un volumen ya existente a un contenedor:

docker run -d --name nextcloud-server -p 8080:80 -v nextcloud_storage:/var/www/html nextcloud

No se puede eliminar un volumen que se está usando. Primero hay que eliminar el contenedor y luego el volumen.

Redes en Docker: Conectando Contenedores

Las redes Docker permiten la comunicación entre contenedores, host y exterior, garantizando flexibilidad y aislamiento.

Tipos de Redes

Bridge

Nos permite que los contenedores estén conectados a una red privada

Red privada

Dos tipos de Bridge

Host

No tiene dirección IP propia

Es cómo si tuviera la dirección IP del Host Docker.

Los puertos son accesibles directamente desde el Host Docker.

No es compatible con Docker Desktop

Optimizar el rendimiento cuando se necesita un gran rango de puertos

docker run --network host ...

None

No tiene acceso a la red externa ni a otros contenedores

No configurará ninguna IP

Mapeo de Puertos

Permite que los servicios dentro de un contenedor sean accesibles desde fuera del host de Docker.

Utilizamos el parámetro -p o --publish en el comando docker run para configurar estas conexiones.

Mapeo Básico

Asigna un puerto del Host a un puerto del contenedor. -p 8080:80 (Host:8080 -> Contenedor:80)

Mapeo con IP Específica

Mapea un puerto en una dirección IP específica del Host a un puerto del contenedor. -p 192.168.1.100:8080:80

Especificar Protocolo

Define si el mapeo es para TCP (por defecto) o UDP. -p 8080:80/udp

Múltiples Protocolos

Mapea el mismo puerto del host para diferentes protocolos. -p 8080:80/tcp -p 8080:80/udp

Acceso Localmente

Restringe el acceso al servicio del contenedor solo desde el propio Host Docker. docker run -d -p 127.0.0.1:8081:80

Red Bridge por Defecto en Docker

Cuando no se especifica una red, los contenedores Docker **se conectan automáticamente a la red bridge por defecto**.

Esta red permite la **comunicación entre contenedores y con el exterior**

El direccionamiento de esta red por defecto es 172.17.0.0/16.

docker run -p 8080:80 ...

Red Bridge Definida por el Usuario

Ofrecen ventajas significativas sobre la red bridge por defecto,

Proporciona mayor control, seguridad y flexibilidad.

Aislamiento y Seguridad Mejorados

Puedes separar grupos de **contenedores en redes distintas**, evitando que servicios que no deben comunicarse tengan acceso entre sí.

Resolución DNS Integrada

los contenedores pueden encontrarse por su **nombre de servicio**, sin necesidad de usar direcciones IP estáticas

Mayor Control y Flexibilidad

Puedes configurar la red a tu medida y conectar o desconectar contenedores sin tener que reiniciarlos.

Es fundamental utilizar redes bridge definidas por el usuario en entornos de producción.

Comandos Básicos de Docker Redes

Dominar estos comandos es fundamental para interactuar eficazmente con Docker y gestionar tus contenedores e imágenes.

docker port

https://docs.docker.com/reference/cli/docker/container/port/

Muestra los puertos mapeados para un contenedor específico.

docker port my-apache-app

docker network ls

https://docs.docker.com/reference/cli/docker/network/ls/

Lista todas las redes Docker disponibles en el sistema.

docker network create

https://docs.docker.com/reference/cli/docker/network/create/

Crea una nueva red definida por el usuario con el driver especificado.

docker network create --subnet=172.20.0.0/16 mired04

docker network create --driver bridge mi_red_produccion

docker network create -d bridge my-bridge-network

docker network inspect

https://docs.docker.com/reference/cli/docker/network/inspect/

Muestra información detallada sobre una red específica.

docker network inspect mi_red_produccion

docker network rm

https://docs.docker.com/reference/cli/docker/network/rm/

Elimina una o más redes definidas por el usuario.

docker network rm mi_red_produccion

docker network connect

https://docs.docker.com/reference/cli/docker/network/connect/

Conecta un contenedor a una red existente.

docker network connect mi_red_produccion my-apache-app

docker network disconnect

https://docs.docker.com/reference/cli/docker/network/disconnect/

Desconecta un contenedor de una red.

docker network disconnect bridge my-apache-app

docker network disconnect multi-host-network container1

Dockerfile: Construyendo Imágenes

El comando docker build es esencial para crear imágenes de Docker a partir de un archivo Dockerfile. Este archivo define una serie de instrucciones para ensamblar la imagen capa a capa, encapsulando la aplicación y sus dependencias.

-t, --tag

Asigna un nombre y, opcionalmente, una etiqueta (nombre:tag) a la imagen. Esto facilita su identificación y versión.

-f, --file

Especifica la ruta al Dockerfile si no se llama Dockerfile o si no está en el directorio raíz del contexto de construcción.

--no-cache

Fuerza una nueva construcción de la imagen, ignorando cualquier capa en caché previamente generada. Útil para asegurar que se utilicen las últimas dependencias.

--build-arg

Permite pasar variables de tiempo de construcción al Dockerfile, que pueden ser usadas para configurar aspectos de la imagen durante su creación.

Comandos Principales del Dockerfile

El Dockerfile es un archivo de texto que contiene todas las instrucciones necesarias para construir una imagen Docker. Dominar estas instrucciones es clave para crear imágenes eficientes y funcionales.

FROM

Define la imagen base para la construcción de tu nueva imagen. Es la primera instrucción en cualquier Dockerfile.

RUN

Ejecuta comandos durante el proceso de construcción de la imagen, creando una nueva capa en el proceso.

COPY

Copia archivos o directorios locales desde el contexto de construcción a la imagen Docker.

WORKDIR

Establece el directorio de trabajo para cualquier instrucción RUN, CMD, ENTRYPOINT, COPY o ADD posterior.

EXPOSE

Informa a Docker que el contenedor escucha en los puertos de red especificados en tiempo de ejecución.

CMD

Define el comando predeterminado que se ejecutará cuando se inicie un contenedor a partir de la imagen.

ENV

Establece variables de entorno que estarán disponibles dentro del contenedor en tiempo de ejecución.

Cada una de estas instrucciones contribuye a la definición de la imagen, permitiendo un control granular sobre su contenido y comportamiento.

Posicionate en la carpeta donde tienes el proyecto. Un archivo “Dockerfile” sin extensión alguna. Si hace falta a lo mejor un script. Ejecuta:

“*docker build -t (nombrequelequierasdaralaimagen) .*”

Ejemplo DOCKERFILE:

```
# 1. BASE Y VARIABLES INICIALES
FROM ubuntu:22.04
ENV DEBIAN_FRONTEND=noninteractive

# 2. ARGUMENTOS DE CONSTRUCCIÓN
ARG SSH_USER=user
ARG SSH_PASSWORD=pass
ARG DB_USER=dbadmin
ARG DB_PASSWORD=dbpass
ARG DB_NAME=prueba_db

# 3. VARIABLES DE ENTORNO PARA CMD / INIT
ENV SSH_USER=${SSH_USER} \
  SSH_PASSWORD=${SSH_PASSWORD} \
  DB_USER=${DB_USER} \
  DB_PASSWORD=${DB_PASSWORD} \
  DB_NAME=${DB_NAME}

# 4. INSTALACIÓN DE PROGRAMAS
RUN apt-get update && apt-get install -y \
  openssh-server \
  mysql-server \
  postgresql \
  sudo \
  && rm -rf /var/lib/apt/lists/*

# 5. CONFIGURACIÓN DE SSH Y USUARIO
RUN mkdir -p /var/run/sshd && \
  useradd -rm -d /home/${SSH_USER} -s /bin/bash -g root -G sudo -u 1000 ${SSH_USER}
&& \
  echo "${SSH_USER}:${SSH_PASSWORD}" | chpasswd
EXPOSE 22

# 6. CONFIGURACIÓN DE BASES DE DATOS (puertos y accesos)
# --- MySQL: Puerto 3307 y conexiones externas ---
RUN sed -i 's/127.0.0.1/0.0.0.0/g' /etc/mysql/mysql.conf.d/mysqld.cnf && \
  sed -i 's/port\s*=.*/port = 3307/' /etc/mysql/mysql.conf.d/mysqld.cnf
EXPOSE 3307

# --- PostgreSQL: Puerto 5433 y conexiones externas ---
RUN sed -i "s/#listen_addresses = 'localhost'/listen_addresses = '*'/g"
/etc/postgresql/14/main/postgresql.conf && \
```

```
   sed -i "s/port = 5432/port = 5433/" /etc/postgresql/14/main/postgresql.conf && \
   echo "host all all 0.0.0.0/0 md5" >> /etc/postgresql/14/main/pg_hba.conf
EXPOSE 5433

#7. SCRIPT DE INICIALIZACIÓN
COPY init.sh /init.sh

# **[NUEVO]** Instala dos2unix y convierte la codificación
RUN apt-get update && apt-get install -y dos2unix \
   && dos2unix /init.sh \
   && chmod +x /init.sh \
   && rm -rf /var/lib/apt/lists/*

# 8. CMD FINAL
CMD ["/init.sh"]

\\\\\\\\\\\\\\EL SCRIPT “init.sh”
#!/bin/bash

# Iniciar servicios
service mysql start
service postgresql start

sleep 5  # esperar a que los servicios estén listos

# MySQL: crear DB y usuario
mysql -u root -e "CREATE DATABASE IF NOT EXISTS ${DB_NAME};"
mysql -u root -e "CREATE USER IF NOT EXISTS '${DB_USER}'@'%' IDENTIFIED BY
'${DB_PASSWORD}';"
mysql -u root -e "GRANT ALL PRIVILEGES ON ${DB_NAME}.* TO '${DB_USER}'@'%'; FLUSH
PRIVILEGES;"

# PostgreSQL: crear rol y DB
sudo -u postgres psql -c "CREATE USER ${DB_USER} WITH ENCRYPTED PASSWORD
'${DB_PASSWORD}';"
sudo -u postgres psql -c "CREATE DATABASE ${DB_NAME} OWNER ${DB_USER};"
sudo -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE ${DB_NAME} TO
${DB_USER};"

# Arrancar SSH
service ssh start

# Mantener contenedor activo
tail -f /dev/null
```

Comandos Básicos de Docker Compose

Dominar estos comandos es esencial para gestionar y orquestar eficazmente tus aplicaciones multi-contenedor con Docker Compose.

docker compose build

https://docs.docker.com/reference/cli/docker/compose/build/

Construye o reconstruye las imágenes de los servicios definidos en el archivo docker-compose.yml.

¿Qué hacer si tu archivo tiene otro nombre?

Si tu archivo de configuración **no se llama** docker-compose.yml o docker-compose.yaml (por ejemplo, si se llama produccion.yml), debes usar la opción -f o --file para especificarlo.

Ejemplo si tu archivo se llama produccion.yml:

docker compose -f produccion.yml up -d

docker compose up

https://docs.docker.com/reference/cli/docker/compose/up/

Crea y arranca los contenedores, redes y volúmenes definidos en el archivo Compose.

docker compose up -d

docker compose down

https://docs.docker.com/reference/cli/docker/compose/down/

Detiene y elimina los contenedores, redes y volúmenes creados por docker compose up.

docker compose ps

https://docs.docker.com/reference/cli/docker/compose/ps/

Lista los contenedores que forman parte del proyecto Docker Compose actual.

docker compose logs

https://docs.docker.com/reference/cli/docker/compose/logs/

Muestra la salida de los logs de los servicios del proyecto.

Ejemplo DOCKER COMPOSE:

```
services:
#SERVICIO DE MySQL
  mysql:
    image: mysql:latest
    container_name: mysql_server
    restart: always  # ESTO HACE QUE LA MÁQUINA SE REINICIE CUANDO SI POR ALGÚN MOTIVO SE DETIENE.
    ports:
      - "3307:3306"  # PUERTO EN EL HOST -> PUERTO DEL CONTENEDOR
    expose:
      - "3306"  # ES EL PUERTO VISIBLE PARA LOS OTROS CONTENEDORES
    environment:
      MYSQL_ROOT_PASSWORD: 12345
      MYSQL_USER: cristobal
      MYSQL_PASSWORD: 12345
      MYSQL_DATABASE: optativadb01
    volumes:
      - mysql_data:/var/lib/mysql  # VOLUMEN PARA PERSISTENCIA
    networks:
      - red_bd  # SE HA PUESTO A AMBOS EN UNA RED COMÚN.

#SERVICIO DE postgresql
  postgres:
    image: postgres:latest
    container_name: postgres_server
    restart: always
    depends_on:
      - mysql  # DEPENDS_ON HACE QUE SE ESPERE A QUE ARRANQUE PRIMERO EL CONTENEDOR ESPECIFICADO. ES DECIR, SE ESPERA A QUE "MYSQL" EMPIECE.
    ports:
      - "5433:5432"
    expose:
      - "5432"
    environment:
      POSTGRES_USER: cristobal
      POSTGRES_PASSWORD: 12345
      POSTGRES_DB: optativadb02
    volumes:
      - postgres_data:/var/lib/postgresql
    networks:
      - red_bd
```

```
networks:
  red_bd:
    driver: bridge  # SE PUSIERON A AMBOS EN UNA RED TIPO BRIDGE

volumes:
  mysql_data:
  postgres_data:
```

COMANDOS USADOS EN LOS EJERCICIOS

Crear y ejecutar contenedor:

docker run -d --name my-apache-app -p 8080:80 httpd:2.4

Comprobamos el mapeo de puertos:

docker port my-apache-app

También podemos usar:

docker inspect --format='{{range $p, $conf :=

.NetworkSettings.Ports}} {{(index $conf 0).HostPort}} -> {{$p}} {{end}}' my-apache-app

Acceder al log del contenedor: *docker logs (nombre del contenedor)*

Copiar objeto y meterlo en un contenedor:

docker cp index.html my-apache-app:/usr/local/apache2/htdocs/

Modificar html que está dentro de container:

echo "<h1>Curso Docker</h1>" > index.html

O te metes en el container:

docker exec -it my-apache-app bash

Evitar que un contenedor se pare:

docker run -d [imagen_o_ID] /bin/sh -c "while true; do sleep 3600; done"

docker run -d --restart always [imagen_o_ID]

docker update --restart unless-stopped [nombre_o_ID_del_contenedor]

Creación de contenedores con variables de entorno:

docker run -d --name mimariadb -e MARIADB_ROOT_PASSWORD=12345 mariadb:10.5

Para comprobar las variables usamos: *docker exec -it mimariadb env*

Limitar uso de recursos: CPU y RAM

Comprobar recursos que se están consumiendo:

docker stats

- CPU:

docker run -d --cpus 1 --name servidor_web httpd:2.4

Muestra limitación cpu:

docker inspect --format '{{.HostConfig.NanoCpus}}' servidor_web

Aunque nos dará un resultado un poco difícil de interpretar. Para visualizarlo mejor, usamos:

nano_cpus=$(docker inspect --format '{{.HostConfig.NanoCpus}}' servidor_web)

cpus=$(echo "scale=2; $nano_cpus / 1000000000" | bc)

echo "CPUs asignadas al contenedor: $cpus"

- RAM:

docker run -d --memory 512m --name servidor_web httpd:2.4

Comprobar RAM:

docker inspect --format '{{.HostConfig.Memory}}' servidor_web

o

memory_limit=$(docker inspect --format '{{.HostConfig.Memory}}' servidor_web)

memory_limit_mb=$(echo "scale=2; $memory_limit / 1048576" | bc)

echo "Límite de memoria asignado al contenedor: $memory_limit_mb MB"

- Modificar en tiempo real:

docker update --memory 1g --cpus 1.5 servidor_web

Volúmenes:

Antes de generar el contenedor, debemos crear el volumen que se le añade durante su creación.

docker volumen create nextcloud_storage

Comprobamos su creación: *docker volumen ls*

Podemos inspeccionarlo: docker volumen inspect nextcloud_storage

Ahora procedemos a crear un contenedor con el volumen creado:

docker run -d --name nextcloud-serer -p 8080:80 -v nextcloud_storage:/var/www/html nextcloud

Redes:

Crear un contenedor con tipo de red None (sin IP, completamente aislada).

docker run -it --name sinred --network none alpine sh

Levantar un contenedor nginx con puerto publicado:

docker run -d --name nginx02 -p 8080:80 nginx

Ver puertos mapeados: docker port nginx02

docker container ls --format "table {{.ID}}\t{{.Names}}\t{{.Ports}}" -a

Listar redes disponibles: *docker network ls*

Inspeccionar una red: *docker network inspect bridge*

Crear una red personalizada: *docker network create mired03*

docker network ls

docker run -d --name nginxserver03 --network mired03 nginx

docker run -d --name nginxserver03 -p 8081:80 --network mired03 nginx

Levantar un contenedor con una IP específica:

En este caso, vamos a crear primero una “network” donde estableceremos la Subnet donde se encontrará la IP estática que estableceremos más adelante.

docker network create --subnet=172.20.0.0/16 mired04

docker run -d --name nginxserver04 --network mired04 --ip 172.20.0.10 nginx

Contenedor con multiples puertos:

docker run -d --name nginxmultiport2 -p 8084:80 -p 8445:443 nginx

Desconectar y eliminar todas las redes:

docker network prune

WARNING! This will remove all custom networks not used by at least one container.

Are you sure you want to continue? [y/N] y

Eliminar todos los contenedores:

sudo docker rm -f $(sudo docker ps -a -q)

Eliminar todas las imágenes:

sudo docker image remove -f $(sudo docker images -a -q)

Eliminar TODO:

docker system prune (lo borra todo)

docker system prune -a (borra incluso más).