DOCKER

Imagen

Una imagen es una forma empaquetada de un programa para poderse ejecutar en un ambiente aislado, teniendo su propio ambiente de ejecución, dependencias y su código fuente.

Descargar una imagen de dockerhub

```
docker image pull IMAGE-NAME
```

Listar imagenes descargadas

```
docker image ls
```

Eliminar una o todas las imagenes

```
docker image prune -a
docker image rm IMAGE-ID
```

Renombrar una imagen

```
docker image tag TAG:VERSION NEWTAG:NEWVERSION
```

Contenedor

Un contenedor es una instancia de una imagen, un contenedor se encarga de ejecutar la imagen en un ambiente aislado de nuestra computadora, como una maquina virtual pero mas ligera.

Iniciar y ejecutar un contenedor

```
docker container run IMAGE-NAME
```

Parar de ejecutar un contenedor

```
docker container stop CONTAINER-ID
```

Ejecutar un contenedor existente

```
docker container start CONTAINER-ID
```

Listar todos los contenedores

```
docker container ls -a
```

Ver los logs de un contenedor y poder seguirlos

```
docker container logs -f CONTAINER-ID
```

Ejecutar comandos dentro del contenedor

```
docker container exec CONTAINER COMMAND
```

Eliminar o todos los contenedores

```
docker container prune
docker container rm CONTAINER-ID
```

Parametros

-f Forzar la eliminación / Seguir los logs
-d Ejecutar el contenedor en segundo plano
-p Asignar puerto host con puerto de contenedor
-a Listar todos los contenedores activos e inactivos
-e Mandar variables de entorno al contenedor
-v Asignar volumen a un directorio del contenedor
-it Ejecutar comandos en la terminal del contenedor
--name Nombrar un contenedor

Volumenes

Un volumen es una forma en que un contenedor puede almacenar información de forma persistente sin que se pierda al eliminar el contenedor y puede ser compartida entre varios

Crear un volumen

```
docker volume create NAME
```

Listar volumenes

```
docker volume ls
```

Eliminar una o todas volumenes

```
docker volume prune
docker volume rm NAME-VOLUME
```

Ejemplo de ejecucion de un contenedor

```
docker container run
  -dp 3306:3306
  -e MARIADB_USER=example-user
  -e MARIADB_PASSWORD=user-password
  -e MARIADB_ROOT_PASSWORD=root-secret-password
  -e MARIADB_DATABASE=world-db
  --name container-volume
  -v my-volume:/var/lib/mysql
  mariadb:jammy
```

Redes

Cuando se tienen mas de un contenedor y se busca una comunicación entre ellos, al estar en un entorno aislado no es posible la comunicación, es necesario crear una red en Docker para ello. Docker se encarga de asignar IP y DNS a cada contenedor conectado.

Crear una red

```
docker network create NAME-NET
```

Conectar un contenedor a una red

```
docker network connect NET CONTAINER
```

Desconectar un contenedor

```
docker network disconnect NET CONTAINER
```

Inspeccionar contenedor

```
docker network inspect NET
```

Eliminar una o todas las redes

```
docker network prune
docker network rm NET-NAME
```

--help: Recibiras ayuda de los comandos y sus descripciones posibles

By: Luis Mata :)

DOCKER

Terminal Interactiva

Al igual como una maquina virtual, puede acceder a la terminar de un contenedor, navegar por sus carpetas y a puro CMD editar los archivos, generalmente usado en bind volumes.

Acceder a la terminal de un contenedor

```
>docker container exec -it CONTAINER EXEC
>docker container exec -it db /bin/sh
```

ls	Listar el directorio
cd	Movernos entre directorios
car FILE	Ver el contenido de un archivo
vi FILE	Editar el archivo
i	Empezar a editar
esc+:wq!	Guardar y salir
exit	Salir de la terminal

Docker Compose

Es un archivo que se usa para no tener que repetir tanto todos los "container run". Asi solo se escribe una vez y se ejecuta con un solo comando
Maneja variables de entorno en un archivo .env.
Compose crea una red por defecto entre los servicios con DNS de nombre de contenedor a cada uno

Ejecutar los contenedores del compose

```
>docker compose up
```

Detener y eliminar los contenedores del compose

```
>docker compose down
```

Estructura de docker-compose.yml

docker-compose.yml

```yaml
version: '3'      #? VERSION DE DOCKER (3)
services:         #? SERVICIOS / CONTENEDORES
  db:             #? NOMBRE DEL SERVICIO
    container_name: ${MDB}  #* Nombre contenedor
    image: mongo:4.0        #* Imagen
    volumes:
      - pokeball:/data/db   #* Volumen
    environment:            #* Variables del entorno
      - MONGO_INITDB_ROOT_USERNAME=${MUSER}
      - MONGO_INITDB_ROOT_PASSWORD=${MPASS}
    restart: always         #* Volver a iniciar si truena
    command: ['--auth']     #* Comandos a ejecutar
    ports:                  #* Puertos
      - "27017:27017"

  express-db:
    container_name: express-db-container
    depends_on:             #* Si necesita esperar a otro contenedo
      - db
    image: mongo-express:1.0.0-alpha.4
    environment:
      - ME_CONFIG_MONGODB_ADMINUSERNAME=${MUSER}
      - ME_CONFIG_MONGODB_ADMINPASSWORD=${MPASS}
      - ME_CONFIG_MONGODB_SERVER=${MDB}
    ports:
      - 8080:8081
    restart: always
volumes:                    #* Declaracion de volumenes
  pokeball:                 #* Nombre del volumen
    external: true          #* Si el volumen a usar ya existe
```

Login en docker hub

Al igual que github, Docker cuenta con repositorios publicos y privados para subir tus imagenes, con version gratuita y de paga, para poder subir tus imagenes es necesario ingresar tus credenciales de coker hub

```
>docker login
```

Para subir una imagen es necesario crear un repositorio y ahi docker te dira que nombre de imagen colocar para dar el pull

```
>docker pull USER/IMAGE:TAG
```

Dockerfile

Docker file es el archivo que nos permite crear imágenes propias de nuestro proyecto, generalmente basado en otras imagenes.

FROM: Instrucion inicial de un Dockerfile, donde marcamos la imagen base para crear nuestra propia imagen (node, python...)

```
FROM IMAGEN-BASE:TAG
FROM --platform=arm64 node:19.2-alpine3.16
```

COPY / ADD: Copea nuestros archivos locales a la imagen destino (origen / destino) (. copear todo)(No crea directorios en la imagen)

```
ADD app.js /app
COPY app.js /app
```

WORKDIR: Navegar a una carpeta dentro de la imagen.

```
WORKDIR /app
```

RUN: Ejecutar comando en la consola de la imagen.

```
RUN npm install
```

CMD: Comandos que se ejecutaran cuando la imagen se corra en un contenedor

```
CMD ["node", "app.js"]
```

Pueden ejecutarse test con RUN, al fallar la imagen no se contruira

By: Luis Mata :)

DOCKER

EXPOSE: Expone el puerto del contenedor cuando se ejecute la aplicacion dentro del mismo, para poderse mapear en el run container con -p

```
EXPOSE 3300
```

ENV: Crea variables de entorno en el contenedor (variable de sistema) que se pueden sobrescribir en el run container, por lo que declararla en el dockerfile solo es para colocar valor por defecto.

```
ENV nombre_variable valor
```

Construir la imagen

Donde tag sera el nombre y tag de la imagen y
. el directorio donde esta el dockerfile
(--no-cahe para evitar uso de cache al construir)

```
>docker build --tag cron-ticker:1.0 .
```

.dockerignore

En este archivo se marcaran todos los directorios que no seran copeados en la construccion de la imagen

.dockerignore

```
node_modules/
Dockerfile
```

Buildx

Los contructores son usados para construir imagenes para distintas arquitecturas de procesadores y funcione en todos los pc

Instalar un builder

```
>docker buildx create --name mybuilder --driver docker-container --bootstrap
```

Listar builders

```
>docker buildx ls
```

Inspeccionar builder

```
>docker buildx inspect mybuilder
```

Usar un constructor

```
>docker buildx use BUILDER
```

Construir imagenes en distintas arquitecturas

```
>docker buildx build --platform linux/amd64,linux/arm64 -t IMAGENAME .
```

Multi-stage Build

Es un proceso mas organizado para construir imagenes, donde utilizamos varias imagenes para dividir etapas en el proceso y crear con ellas una imagen final. Cada etapa tendra un alias "as" y lo usaremos de referencia cuando necesitemos algo de cierta etapa en una nueva.

Dockerfile

```
# Primera etapa "Instalacion de dependencias dev"
FROM node:19.2-alpine3.16 as test-deps
WORKDIR /app
COPY package.json ./
RUN npm install

# Segunda etapa "Ejecucion de los tests"
FROM node:19.2-alpine3.16 as tests
WORKDIR /app
COPY --from=test-deps /app/node_modules ./node_modules
COPY . .
RUN npm run test

# Tercera etapa "Dependencias de produccion"
FROM node:19.2-alpine3.16 as prod-deps
WORKDIR /app
COPY package.json .
RUN npm install --prod

# Cuarta etapa "Runner de la app en prod"
FROM node:19.2-alpine3.16 as prod-runner
WORKDIR /app
COPY --from=prod-deps /app/node_modules ./node_modules
COPY . .
CMD ["node", "app.js"]
```

Crear imagen con DockerCompose

Para usar una imagen al correr un contenedor, pero quieres crearla en el momento justo en donde se ejecuta el docker compose, puede usar build en lugar de image, ahi puedes seleccionar el dockerfile que construye tu imagen (util para desarrollo y ver los cambios rapidamente sin construir la imagen a cada rato)

```
express-db:
  build:
    context: "./"          # path del dockerfile
    dockerfile: Dockerfile # nombre de archivo
    target: dev            # etapa a ejecutar
  image: "NOMBRE NUEVO IMAGEN"
```

Usa docker compose build para generar la imagen

```
>docker compose build
```

Usa docker compose up para ejecutar los contenedores

```
>docker compose up
```

Construir solo un servicio del docker compose

```
>docker compose build servicio
```

By: Luis Mata :)