Training

[Docker for Beginners](https://training.play-with-docker.com/beginner-linux/)

* Arrancar un contenedor:
  + **docker container run** alpine hostname
    - Arranca un contenedor alpine y ejecuta el comando hostname.
  + docker container run --interactive --tty --rm ubuntu bash
    - Arrancar un contenedor de Ubuntu interactivo (interactive) y lo borra al finalizar (rm). Aquí se puede ejecutar comando, por ejemplo ls, ps aux (listar los procesos activos)
    - Para salir ejecutar “exit”.
* Listar todos los contenedores:
  + docker container ls –all
* docker container run --detach --name mydb -e MYSQL\_ROOT\_PASSWORD=my-secret-pw mysql:latest
  + –detach 🡪 Corre el contenedor en background.
  + --name 🡪 nombre de la BBDD.
  + -e 🡪 Para especificar una variable de entorno, en este caso e MYSQL\_ROOT\_PASSWORD y asignarle la password.
* docker container logs mydb
  + Ver el log del container mydb.
* docker container top mydb
  + Para ver los procesos que se ejecutan dentro del contenedor mydb.
* docker exec (docker container exec [Options] CONTAINER COMMAND [ARG...]
  + Permite ejecutar comando dentro de un contenedor
  + docker exec -it mydb mysql --user=root --password=$MYSQL\_ROOT\_PASSWORD –version
    - Ejecuta el comando para ver la versión, pasando los parámetros de usuario y password dentro del contenedor mydb.
  + Esto mismo se puede realizar en varios pasos:
    - docker exec -it mydb sh
      * Con esto indicamos que queremos ejecutar comando (exec) dentro del contenedor mydb y quedarnos dentro de este (-it interactive, tty )
    - mysql --user=root --password=$MYSQL\_ROOT\_PASSWORD –version
      * Ejecutar el comando mysql con los parámetros necesarios para ver la versión instalada.
    - Exit.
      * Salimos del container.
* Crearnos imagenes de un contenedor a partir de un dockerfile
  + Dockerfile
    - Es un fichero de texto donde se especifican los pasos para crear una imagen Docker, una especie de plantilla de comandos necesarios para construir un contenedor.
    - Por ejemplo:
      * FROM nginx:latest 🡪 Aquí se especifica la imagen base de la que queremos partir, para este ejemplo usamos na imagen del servidor web/proxy nginx más reciente.
      * COPY index.html /usr/share/nginx/html -🡪 copia el fichero index.html a la ruta especificada
      * COPY linux.png /usr/share/nginx/html
      * EXPOSE 80 443 🡪 Indica a Docker en que puertos va a estar escuchando el nuevo contenedor.
      * CMD ["nginx", "-g", "daemon off;"] 🡪 ejecuta un comando, para este caso el de ngin con sus parámetros
  + Pasos:
    - Nos creamos una variable de entorno con el id de nuestro Docker
      * export DOCKERID=pGeradoIgletio
    - Crear la imagen a partir del fichero dockerfile.
      * **docker image build** --tag $DOCKERID/linux\_tweet\_app:1.0 . (-tag es para darle un nombre)
* Una vez hemos creado la imagen, Podemos crear un contenedora partir de esta.
  + docker container run \
    - --detach \
    - --publish 80:80 \ 🡪 host\_port:container\_port 🡪 se publica en el Puerto 80 en el contenedor y en el host, para que lo que llegue al puerto 80 en el host se redireccione al del contenedor.
    - --name linux\_tweet\_app \
    - $DOCKERID/linux\_tweet\_app:1.0
  + Con esto, ya podemos ver la web.
  + Para borrarlo
    - docker container rm --force linux\_tweet\_app
* Para montar (--mount) el path donde te encuentras en un contenedor. Esto es para realizar un mapeo de un directorio de Docker host en una imagen. A partir de esto, cualquier cambien en el path del host se reflejará inmediatamente en el contenedor. Esto sirve para cuando realicemos cualquier cambien en el código, no tengamos que estar modificando la imagen, si cambiamos en esta ejemplo el html, el código automáticamente se actualiza.
  + docker container run \
    - --detach \
    - --publish 80:80 \
    - --name linux\_tweet\_app \
    - --mount type=bind,source="$(pwd)",target=/usr/share/nginx/html \
    - $DOCKERID/linux\_tweet\_app:1.0
  + Con esto no estamos cambiando la imagen, ya que el path esta enlazado, pero la imagen no tiene realmente los ficheros actualizados.
* Para actualizar una imagen:
  + Hay que crear una nueva versión de la imagen inicial.
    - docker image build --tag $DOCKERID/linux\_tweet\_app:2.0 .
    - Docker solamente añade las modificaciones, en este caso el html, no re crea una nuevo.
* Consutlar todas las imágenes:
  + docker image ls
* Una vez finalizados los pasos anteriores, la imagen que hemos creado solamente la tenemos en nuestro local, para subirlo a mi usuario, hay que logarse y ejecutar los siguientes comandos:
  + docker login
  + Teclear el usuario y password de nuestro Docker
  + Para subirlo a remote: (estilo gift)
    - **docker image push** $DOCKERID/linux\_tweet\_app:1.0
    - docker image push $DOCKERID/linux\_tweet\_app:2.0

[Application Containerization and Microservice Orchestration](https://training.play-with-docker.com/microservice-orchestration/)

* Vamos a crear tres microservicios que se comuniquen entre si. El primero de ellos consistirá en un servicio que lea de un html y devuelva todos los enlaces que encuentra. Este se recorrerá el fichero y buscará el literal <a> y los devolverá a un segundo servicio que los vaya acumulando. El terceros servirá para visualizar todos los enlaces, creo?.
* Nos creamos una imagen para poder ejecutar Python, para eso:
  + Bajamos el repositorio de git
    - git clone <https://github.com/ibnesayeed/linkextractor.git>
  + Entramos en la carpeta y nos cambiamos de rama
    - cd linkextractor
    - git checkout step1
  + Creamos una imagen con Dockerfile
    - docker image build -t linkextractor:step1 (-t sirve para asignarle una etiqueta)