Training

# [Primera parte: Developers Stage 1](https://training.play-with-docker.com/dev-stage1/)

## [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 sí. 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 href y los devolverá a un segundo servicio para que los vaya acumulando. El terceros será una web la cual a partir de la url invocará al primer servicio y pintará gráficamente todos los enlaces.
* Primer microservicio (Step 1 y 2):
  + Nos creamos una imagen para poder ejecutar Python y tenga unas librerías que son necesarias, 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)
    - Una vez tenemos la imagen, podemos crear un container, pasándole la url donde irá a buscar todos sus enlaces.
      * docker container run -it --rm linkextractor:step1 <http://example.com/>
        + Con la url de ejemplo que le hemos pasado, va a buscar el enlace que tienen dentro y nos lo va a devolver.
  + Siguiente paso, mejorar la salida
    - Nos cambiamos a la siguiente Branch
      * git checkout step2
    - aquí tenemos actualizado el script de Python para afinar las búsquedas de las url’s, por lo que vamos a generar una nueva imagen.
      * docker image build -t linkextractor:step2 .
* Segundo microservicio (Step3).
  + Nos posicionamos en el branch step3
    - git checkout step3
  + Aquí tenemos:
    - un nuevo scrip en Python (main.py) que recoge los enlaces devueltos por el primer microservicio.
    - El dockerfile actualizado con las nuevas llamadas.
    - El endpoint donde se expone el servicio (<http://<hostname>[:<prt>]/api/<url>>). (esto se esta haciendo en el script de Python.
    - Un fichero donde se mueven las dependencias. (requirements.txt)
  + Con todo esto, generamos una nueva imagen
    - docker image build -t linkextractor:step3 .
  + Corremos el contenedor.
    - docker container run -d -p 5000:5000 --name=linkextractor linkextractor:step3 (La opción -d es para poder continuar ejecutando más comandos).
    - Según el dockerfile, al final tienen el CMD donde ejecuta el main.cp, que es el que está escuchando en el puerto 5000 + el endpoint publicado y quien devuelve el resultado previamente llamando al otro script de Python que buscaba las url’s.
* Desplegar los dos servicios en contenedores diferentes (Step 4)
  + Nos posicionamos en el branch step4
    - git checkout step4
  + Aquí tenemos:
    - Un fichero yml donde se configurará la orquestación.
      * En este fichero se configurarán los dos servicios donde se indicará el puerto y url donde se va a estar expuesto, imágenes que usarán, variable de entorno con parte del endpoint
    - La web index.php, accesible desde http://<hostname>[:<prt>]/?url=<url-encoded-url>
  + Para este paso, se van a crear dos contenedores, en uno correrá el API y en el otro la aplicación web, la cual habrá que darla un mecanismos para que puedan comunicarse. Habría varias posibilidades:
    - Realizar una mapeo de sus puertos en la máquina host y usarlo para el enrutamiento de las peticiones.
    - Poner ambos contenedores en una misma red privada para que puedan acceder directamente.
      * Docker tiene un excelente soporte de redes y proporciona comandos útiles para tratar con redes. Además, en una red Docker, los contenedores se identifican usando sus nombres como nombres de host para evitar la búsqueda de sus direcciones IP en la red privada.
    - Utiliza composite Docker.
  + Para el ejemplo se atizará el composite Docker
    - **docker-compose** up -d –build
  + Una vez ejecutado el composite , tenemos dos contenedores iniciados y ya podemos probarlo.
    - El api linkextractor\_api\_1
      * Para probarlo con una llamada directa:
        + curl -i <http://localhost:5000/api/http://example.com/>
    - linkextractor\_web\_1
      * Para lanzar la web
        + <http://ip172-18-0-21-bgs6t4s3uhdg0081o9g0-80.direct.labs.play-with-docker.com/>

Desde aquí, si tecleamos una url nos dirá todos sus enlaces bien pintados

* + Para parar los contenedores
    - docker-compose down
* Tercer microservicio REDIS(Step 5)
  + Nos posicionamos en el branch step4
    - git checkout step4
  + Aquí tenemos:
    - Se añade un nuevo Dockfile para el componente web.
    - En el yaml se añade un nuevo servicio de REDIS para cacheo.
    - Una nueva variable de entorno REDIS\_URL definida para que el API existente, ya que este es el que va a realizar las invocaciones al servicio de REDIS
  + Se arrancan los contenedores a partir del nuevo yaml.
    - docker-compose up -d –build
  + Ya tenemos los tres servicios disponibles.
* Nuevo microservicio en rubby (Step 6)
  + Se cambia el script de python por uno que hace casi lo mismo, pero en rubby

## [Deploying a Multi-Service App in Docker Swarm Mode](https://training.play-with-docker.com/swarm-stack-intro)

* Hay dos terminales, uno es el que hará de manager y el segundo de worked
* Para crear el nodo worker, hay que ejecutar en el manager lasiguiente instrucción para obtener el toker para que se puedan unos los worked
  + **docker swarm** **init** --advertise-addr $(hostname -i)
    - Como salida, te da el comando que hay que ejecutar en el segundo nodo para que se una como worker.
      * **docker swarm join** –token ….
  + Desplegar un STACK, Stack es un grupo de servicios que se despliegan juntos. Estos diferentes servicios se describen en un fichero yaml y se ejecuta:
    - docker stack deploy --compose-file=docker-stack.yml voting\_stack
  + Se puede mostrar el listado de stack que hay desplegados:
    - docker stack ls
  + Se puede listar los servicios que contiene un stack
    - docker stack services voting\_stack

# [Segunda Parte](https://training.play-with-docker.com/ops-stage2/)

## [Security](https://training.play-with-docker.com/security-seccomp/)

* Seccomp. Hace de filtro para las llamadas y manejos de los contenedores.