Taller “Introducción a Docker”
Explicación escenarios Docker en Katacoda

Escenarios disponibles en https://www.katacoda.com/courses/docker/
Documentación del taller disponible en https://github.com/sergarb1/TallerIntroduccionDocker

Enlaces previos para conocer Docker:

- Docker: https://es.wikipedia.org/wiki/Docker_(software)
- Docker vs Hypervisor: https://munzandmore.com/2015/cc/docker-container-vs-virtualization
- Video ¿Que es Docker?: https://www.youtube.com/watch?v=1LRzlUoyZg4
- Instalar Docker: https://docs.docker.com/get-docker/
  - Para Windows 10 Home: https://github.com/docker/toolbox
- Interfaz gráfica Kitematic: https://github.com/docker/kitematic/releases

1) Deploying Your First Docker Container

a) `docker search redis`
Realiza una busquda en Docker Hub de imagenes con la palabra clave redis

b) `docker run -d redis`
-d indica “deatached mode”, no enlazado a nuestra terminal
Usa para el contenedor la imagen redis. Al no indicar version, coge “latest”
Al no indicar un nombre (se indica con --name), le pone uno al azar.

c) `docker ps`
Permite ver los contenedores en ejecución. También nos da información adicional, como el mapeo de puertos.

d) `docker inspect  “id/name”`
Permite ver mas información de un contenedor o su id

e) `docker logs  “id/name”`
Permite ver mensajes que el contenedor a escrito en la entrada estandar o en la entrada de error.

f) `docker run -d --name redisHostPort -p 6379:6379 redis:latest`
--name=db establece nombre del conntenedor. Si no se da, se pone una aleatorio.
Lanza el contenedor con el nombre “redisHostPort” y enlaza el puerto del contendor 6379 al puerto de la maquina 6379. OJO, crea de nuevo el contenedor cada vez que se haga “run”.

El formato es `ip:hostPort:containerPort`

- Importante 1: por defecto, el puerto esta mapeado a 0.0.0.0, es decir, a todas las IP, pero se pued e especificar una, por ejemplo 127.0.0.1:6379:6379

- Importante 2: si solo pones -p 6379, enlaza con el puerto 6379 del contenedor, pero en el host elige un puerto libre aleatorio.

g) `docker port redisDynamic 6379`
Nos permite averiguar a que IP/Puerto del host esta enlzado el puerto 6379 del contenedor.
Recordemos también puede verse con "`docker ps`".

h) `docker run -d --name redisMapped -v /opt/docker/data/redis:/data redis`

`-v /opt/docker/data/redis:/data`
Enlaza los directorios (llamados volumenes) del host y del contenedor. Esto es útil ya que los contenedore sson sin estado y si eliminas y creas un contenedor, pierdes la información.

En este ejemplo, el directorio /data del contenedor se enlaza con /opt/docker/data/redis de la maquina local.

Importante: los volumenes se hacen al crear el contenedor, no pueden crearse en caliente.

i) `docker run -it ubuntu bash`

-i enlaza con STDIN y -t crea una pseudo terminal
Al crear el contenedor, ejecuta el comando final, que es "bash". Este comandos, nos crearia una shell y accederiamos al contenedor.

- Importante: docker run crea contenedores, si hacemos esta orden de nuevo, estamos haciendo un nuevo contenedor, no accediendo con una shell al contenedor antiguo.
Para ejecutar una orden (o una shell) en un contenedor en funcionamiento, ejecutariamos "`docker exec -it id/nombre env`"

2) Deploy Static HTML Website as Container

a) Contenido del fichero "Dockerfile"

```
FROM nginx:alpine
COPY . /usr/share/nginx/html
```

Este contenido será usado cuando ejecutemos el comando "docker build"

b) "`docker build -t webserver-image:v1 .` "
Usando el ejemplo anterior:
Docker build construye una imagen (Importante IMAGEN! No contenedor)

-t nos permite dar un "nombre" a nuestra image y una etiqueta de version.

El "." al final, indica en que directorio del host, hacemos el build del contenedor.

Las instrucciones del Dockerfile indicaban que la imagen creada se basara en "nginx:alpine" y se le aplicara la opcion de copiar todo el directorio actual de "build" a la carpeta de la imagen /usr/share/nginx/html

c) `docker run -d -p 80:80 webserver-image:v1`
Partiendo de la imagen anterior, podria crear un contenedor con dicha imagen sirviendo en el puerto 80.

3) Building Container Images

a) Contenido del Dockerfile

```
FROM nginx:1.11-alpine
COPY index.html /usr/share/nginx/html/index.html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

FROM indica la imagen base
COPY copia un fichero del directorio actual del host a la imagen (importante, debe indicarse el fichero final en el destino)
EXPOSE indica que la imagen permite el enlazado del puerto 80
CMD indica que al crearse el contenedor, se lanzara la orden nginx -g daemon off.

b) `docker build -t my-nginx-image:latest .`
Comando para crear la imagen en base al Dockerfile del directorio actual

c) `docker run -d -p 80:80 my-nginx-image:latest`
Comando para crear un contenedor a partir de la imagen anterior enlazado con puerto 80

4) Dockerizing Node.js applications

a) Contenido Dockerfile

```
FROM node:10-alpine
RUN mkdir -p /src/app
WORKDIR /src/app
```

RUN indica un comando a ejecutar como si fuera lanzado desde una shell.
WORKDIR establece como directorio base “/src/app”, directorio desde el cual se lanzaran todas las ordendes que mandemos al contenedor.

b) Añadimos al Dockerfile

```
COPY package.json /src/app/package.json
RUN npm install
```

COPY copia el fichero “package.json” al directorio /src/app.
- Importante 1: “package.json” indica dependencias de una aplicación Node.
RUN ejecuta el comando “npm install” desde el WORKDIR definido.
- Importante 2: “npm install” obtiene que dependencias instalar de “package.json”
- Importante 3: Diferencia CMD y RUN. CMD se lanza al crear el contenedor. RUN se lanza mientras se crea la imagen

c) Añadimos al Dockerfile

```
COPY . /src/app
EXPOSE 3000
CMD [ "npm", "start" ]
```

COPY copia todo el contenido del directorio actual del host donde esta el Dockerfile al directorio de la imagen “/src/app”.
EXPOSE 3000 indica que el puerto 3000 es el puerto a enlazar
CMD indica que al lanzarse el contenedor, se lance la orden “npm start” en el directorio defindido por WORKDIR
- Importante 1: “npm start” utiliza información definida en “package.json” para iniciar una aplicación Node, es decir, lanza lo que defina el script “start”.
En el ejemplo, el script se define dentro de “package.json” como "start": "node ./bin/www" que lo que hace en la practica es iniciar la aplicación.

d)

```
docker build -t my-nodejs-app .
docker run -d --name my-running-app -p 3000:3000 my-nodejs-app
```

Comandos para crear la imagen y posteriormente lanzar el contenedor con la aplicación Node.

e)

```
docker run -d --name my-production-running-app -e NODE_ENV=production -p 3000:3000 my-nodejs-app
```

-e define variables de entorno. En este ejemplo define dentro del contenedor la variable “NODE_ENV” con el valor “production”.

5) Create Data Containers

a)

```
docker create -v /config --name dataContainer busybox
```

Crea un contenedor de datos (contenedores usados únicamente para almacenar datos) con un volumen asociado al directorio “/config”
Se usa “busybox” como imagen base.

b)

```
docker cp config.conf dataContainer:/config/
```

docker cp copia de la maquina host el fichero “config.conf” al contenedor con nombre “dataContainer” al directorio “/config”

d)

```
docker run --volumes-from dataContainer ubuntu ls /config
```

--volumes-from coge los volumenes del contenedor “dataContainer” y los enlaza al nuevo contenedor creado.
- Importante: si el directorio /config ya existista en la imagen, este es sobre-escrito.

e) `docker export dataContainer > dataContainer.tar`
Esta orden nos permite exportar a un fichero “dataContainer.tar” un contenedor (sea de datos o sea un contenedor normal).

f) `docker import dataContainer.tar`
Esta orden nos permite importar un contenedor desde un fichero “dataContainer.tar”

6) Creating Networks Between Containers using Links

a)
```
docker run -d --name redis-server redis
docker run --link redis-server:redis alpine env
docker run --link redis-server:redis alpine cat /etc/hosts
```

La primera orden, crea el contenedor “redis-server”.

La segunda orden y tecera orden, crean un contenedor y lo enlaza con “redis-server” usando como alias interno para referirse a el “redis”.

- La segunda orden ejecuta la orden “env” para ver que algunas variables de entorno se han creado en el contenedor enlazo.
- La tercera orden ejecuta la orden “cat /etc/host” para ver que en el contenedor enlazado se han añadido entradas en “host” relacionando los contenedores.

b)`docker run --link redis-server:redis alpine ping -c 1 redis`

Otro ejemplo en el contexto anterior, que lanza un ping al alias “redis” para comprobar que responde.

7) Creating Networks Between Containers using Networks

a) `docker network create backend-network`

Creamos una red para ser usada en Docker con el nombre “backend-network”.

b) `docker run -d --name=redis --net=backend-network redis`

Creamos un contenedor con nombre redis, que se enlaza a la red creada anteriormente.

c) `docker run --net=backend-network alpine env`
`docker run --net=backend-network alpine cat /etc/hosts`

Con estas ordenes, se crean dos contenedores que se agregan a la red existente. Uno al ejecutarse lanza la orden “env” y el otro “cat /etc/hosts”. Esto nos permite ver, que al contrario que enlazando con links, con este metodo Docker no introduce variables de entorno nuevas ni modifica el fichero host.

d) `docker run --net=backend-network alpine cat /etc/resolv.conf`

Para comunicarse entre contenedores mediante nombres de contenedoresDocker, realmente se tiene un “servidor DNS enmebebido”.
Este contenedor lanza el comando “cat /etc/resolv.conf” y se ve como hace referencia al servidor embebido para resolver los nombre Docker.

e) `docker run --net=backend-network alpine ping -c1 redis`

Con esta orden se puede comprobar que el ping funciona usando nombres de contenedores Docker que son resueltos por el servidor DNS embebido por Docker.

f) `docker network create frontend-network`
`docker network connect --alias db frontend-network redis`

La primera orden crea una red “frontend-network”.
La segunda orden, conecta el contenedor con nombre “redis” a la red “frontend-network2” usando el alias “db”
La tercera orden comprueba el funcionamiento correcto del alias con un contenedor que lanza un ping.

g) `docker network ls`

Nos muestra las redes existentes en nuestro sistema Docker.

h) `docker network inspect frontend-network`

Nos proporciona información sobre la red indicada.

i) `docker network disconnect frontend-network redis`

Este comando desconecta el contenedor “redis” de la red “frontend-network”