Despliegue de aplicaciones web

Tema 6. Comunicación entre contenedores

1. Introducción

En esta unidad se va a introducir el concepto de redes. Con redes se refiere a las posibles relaciones que pueden establecerse:

- para conectar múltiples contenedores entre sí permitiendo comunicación entre ellos.
- para conectar un contendor en ejecución a la máquina local. Por ejemplo, realizar una petición HTTP a otro servicio corriendo en la máquina anfitriona o conectar el contenedor con una BBDD que se encuentra en la máquina local.
- para conectar con internet desde el interior del contenedor.
- etc.

Cuando se hace uso de Docker, es recomendable que un contenedor se centre en una sola cosa. Por ejemplo, si se trata de una aplicación que accede a una base de datos, la aplicación en sí, con todas su ficheros y dependencias, tendría su propia imagen con su propio contenedor a ejecutar, mientras que habría otra imagen(y su correspondiente contenedor) para la base de datos.

Para que una aplicación corriendo en un contenedor pueda comunicarse con la red, no hay que hacer nada en especial. Ya se ha visto que es necesario indicar el puerto por el que le van a llegar peticiones, pero no es necesario realizar nada especial para que pueda acceder a la red.

No pasa lo mismo para poder acceder a algún servicio corriendo en la máquina host o para establecer comunicación con otros contenedores.

docker network ls

Con este comando es posible ver las distintas redes de docker, que de inicio tiene tres predefinidas:

```
NETWORK ID     NAME      DRIVER    SCOPE
4c59ff237549   bridge    bridge    local
23b8745a4cb6   host      host      local
448b9fd7318f   none      null      local
PS D:\DAW\2º DAW\Despliegue de aplicaciones
```

Si no se configura de otra manera, los contenedores se conectan a la red de tipo bridge (cuyo direccionamiento por defecto es 172.17.0.0/16), y para exponer algún puerto al exterior tienen que usar la opción -p para mapear puertos.

Si el contenedor se conectara a la red host, éste sería accesible usando la misma IP que la máquina host. En este caso, no sería necesario indicar los puertos con -p pero si habría que indicar la red con la opción --network y host. Por ejemplo:

```
docker run --rm --name dockervueHost --network host soniagg/vue_cli
  Starting up http-server, serving dist
  Available on:
    http://127.0.0.1:8081
    http://192.168.65.3:8081   ip del equipo
    http://172.17.0.1:8081     ip red docker (host)
  Hit CTRL-C to stop the server
```

Podemos acceder directamente al puerto 80 del servidor para ver la página web.

La red none no configurará ninguna IP para el contenedor y no tiene acceso a la red externa ni a otros contenedores. Tiene la dirección loopback y se puede usar para ejecutar trabajos por lotes.

docker network create

Además, es posible configurar nuevas redes, por ejemplo para crear una red de tipo bridge, ejecutando "docker network create mired":

```
NETWORK ID     NAME      DRIVER    SCOPE
4c59ff237549   bridge    bridge    local
23b8745a4cb6   host      host      local
0c8b6aa514de   mired     bridge    local
448b9fd7318f   none      null      local
```

Hemos visto que se puede arrancar un contenedor en una red concreta usando la opción --network. Si la red es de tipo bridge, pero no es la red bridge por defecto, habrá que indicar el nombre de la red y el puerto.

docker network inspect

Con este comando es posible visualizar las características de una red pasando como parámetros el nombre de la misma.

Dependiendo de la red que se esté usando (la red puente por defecto o una red definida por el usuario) el mecanismo de enlace entre contenedores será distinto.

2. Comunicación con la máquina host

Si existe una base de datos, un servidor web, etc. corriendo en la máquina local(pero no en un contenedor), es posible conectarse desde un contenedor a los servicios que se ofrezcan.

Cuando se accede a un servicio web local desde el navegador, se usa la dirección en sí o la palabra "localhost". El problema es que si una aplicación corriendo en un contenedor quiere acceder, por ejemplo, a un servicio web en la máquina host, "localhost" no será válida, ya que con ello el contenedor entenderá **SU** localhost.

Docker provee un dominio especial(que el entenderá) y que referencia a la máquina anfitriona. Esta instrucción será traducida a la IP de la máquina host:

host.docker.internal

Por ejemplo, una conexión a mysql desde una aplicación java:

spring.datasource.jdbcUrl=jdbc:mysql://localhost:3306/db

Para que funcione desde un contenedor:

spring.datasource.jdbcUrl=jdbc:mysql:// host.docker.internal:3306/db

En este ejemplo se utiliza para una petición de tipo jdbc:mysql, pero sería lo mismo para una petición mongodb o http, por ejemplo.

El valor de host.docker.internal está establecido en el fichero hosts de la máquina local, y es un valor que establece Docker en si instalación. Hay que tener cuidado porque puede que Docker establezca un valor diferente a 127.0.0.1(localhost) y habría que cambiarlo.

Esta es una funcionalidad que no se recomienda usar en entorno de producción.

EJERCICIO 1

- Desarrolla(o utiliza una que ya tengas) una aplicación que solicite datos a alguna API externa(a la API JSonPlaceHolder servirá).(Por ejemplo, el ejercicio del periódico con Vue-Router es suficiente)
- Crea una imagen para dicha aplicación, ejecuta un contenedor de la misma y comprueba que la aplicación funciona y carga los datos solicitados a la API.
- Crea un fichero .json y copia en el el contenido de la API a la que estabas llamando en el punto anterior. Arranca el servidor local json-server para que sirva los datos que acabas de incluir en el fichero .json.
- Modifica el código de tu aplicación para que, tras regenerar la imagen, la aplicación siga funcionando pero solicitando los datos al servidor local json-server.

3. Comunicación entre contenedores

Lo más óptimo sería tener todas las aplicaciones, servicios, BBDD, etc. en contendores. Cada una de estas partes necesarias para que la aplicación funcione, estaría corriendo en un contendor diferente.

Docker permite crear una red de contenedores, para después ejecutar tantos contenedores como sea necesario en la misma haciendo uso de la opción *--network nombre_red*.

Cómo crear una red, ya se vio en el primero apartado del tema. Si después todos los contenedores que vayan a formar parte de la misma red se arrancan con la opción--network y el nombre de dicha red, serán capaces de comunicarse entre sí.

Para que la aplicación corriendo en un contenedor pueda comunicarse con otra corriendo en otro contenedor de la misma red, podrá referenciarlo, desde el código de la aplicación, simplemente por el nombre del contenedor. Por ejemplo, si el contenedor de la BBDD se llama conteendor_db:

```
spring.datasource.jdbcUrl=jdbc:mysql://contenedor_db:3306/db
```

EJERCICIO 2

- Descarga el proyecto “Ejemplo NodeJS y MongoDB” del Aula Virtual.
- Esta aplicación que vas a dockerizar es una aplicación que sirve datos, no tiene parte front-end. Para probarla haremos uso de Postman:
  - Descarga Postman y ejecútalo en tu equipo.
  - Juntos vemos cómo hacer uso de Postman.
- Crea una red propia para poder arrancar en ella los siguientes contenedores.
- Necesitarás un contenedor para mongo y otro para la aplicación:
  - De mongo bastará con arrancar un contenedor de la imagen oficial, pero ten en cuenta que tienes que arrancarlo en la red correcta.
  - Modifica el código de la aplicación para que, cuando vaya a establecer la conexión con BBDD, en lugar de buscar la BBDD en localhost lo haga en el contenedor anteriormente arrancado.
  - Genera la imagen de la aplicación y arranca un contenedor.
- Una vez arrancados los dos contenedores, haz peticiones a través de Postman.

Si la petición a un contenedor viene de fuera del mismo, o de fuera de la red, por ejemplo desde el navegador, ya no se podrá hacer de esta manera recién vista, ya que Docker no podrá resolver la petición en caso de que se esté referenciando al contendor por el nombre.

Esto es así porque Docker tiene un servidor DNS incorporado que ayuda a los contenedores a resolverse entre sí utilizando sus nombres. Hay que tener en cuenta que el servidor DNS incorporado siempre se ejecuta en la dirección 127.0.0.11.

EJERCICIO 3

- Vamos a utilizar una imagen de json-server, por ejemplo: **williamyeh/json-server**.
  - En el siguiente enlace se puede ver el fichero Dockerfile de la misma y los pasos a seguir para hacer uso de ella: https://github.com/William-Yeh/docker-json-server.
  - Revisa los parámetros necesarios para arrancar un contenedor de dicha imagen. Si alguno no entiendes pregúntalo.
  - Genera un contenedor de esta imagen para que sirva los datos almacenados en el fichero json que creaste en el ejercicio 1.
  - Una vez arrancado el contenedor comprueba que es accesible desde el navegador. No pases al siguiente paso hasta que no puedas ver el contenido del fichero json en el navegador.
- Teniendo en cuenta como resuelve Docker la comunicación entre contenedores, haz las modificaciones necesarias para que la aplicación Vue del contenedor que creaste en el ejercicio 1 sea capaz de solicitar datos al json-server corriendo en el contenedor del punto anterior.