DigitalHouse>

Certified Tech Developer
The Ultimate Degree

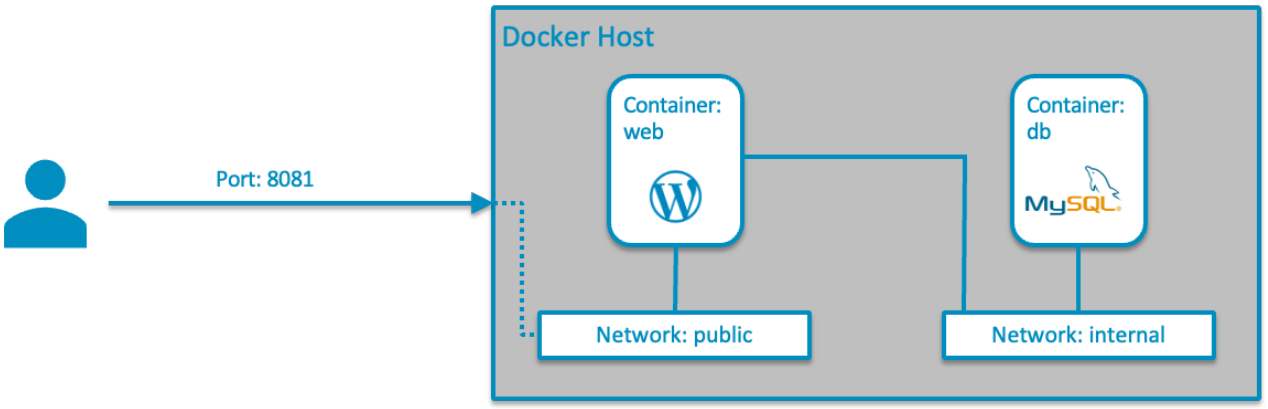
Infraestructura I

Objetivos

Construir un ambiente de WordPress con Docker que nos permitirá ver la tecnología en funcionamiento y entender su potencial.

¿Qué vamos a construir?

Un ambiente que se ve de la siguiente manera:



Para ellos vamos a:

- Construir 2 redes de tipo “bridge”, la primera llamada “public”, y la segunda llamada “internal”.
    - La red “internal” debe permitir solo tráfico entre los containers conectados a ella, y no permitir tráfico externo.
- Crear un container llamado “db” con la imagen de MySQL que debe estar conectado a la red “internal”.
- Crear un container llamado “web” con la imagen de WordPress que debe estar conectado a ambas redes.

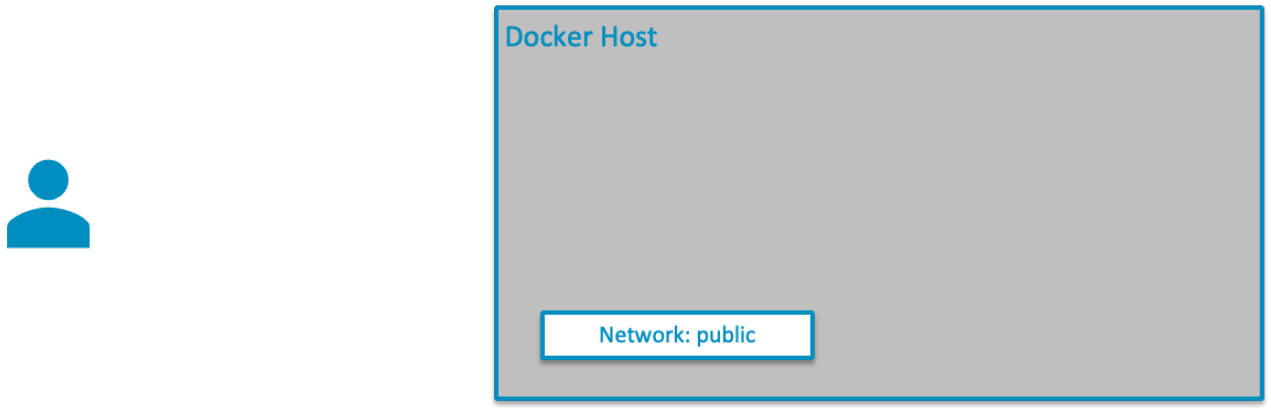
    - Este contenedor debe estar expuesto en el puerto 8081 del host para poder visitar el sitio de WordPress.
- Tanto el container “db” como el container “web” deben estar configurados de manera que puedan hablar el uno con el otro —vamos a ver que ambos servicios se pueden configurar mediante variables de entorno—.

Instrucciones

Lo primero que vamos a hacer es crear la red llamada “public”. Para ello, vamos a utilizar el siguiente comando:

```
docker network create public
```

Ejecutando este comando llegamos a este estado:



Tip: Si analizamos el comando que acabamos de correr, vamos a ver que está compuesto por cuatro términos:

- **docker**: para invocar al ya famoso cliente de Docker o Docker CLI (Command Line Interface).
- **network**: este término engloba todas las operaciones relacionadas con las redes de Docker.
- **create**: utilizando este término, dentro del contexto de “network”, estamos indicando que deseamos crear una red.
- **public**: finalmente, este término es un parámetro del comando “create” e

indica el nombre que queremos asignarle a esa red que estamos creando.

Para conocer más sobre los comandos que están bajo el grupo de “network”, podemos ejecutar:

```
docker network --help
```

Para conocer más sobre los parámetros que podemos utilizar para el comando “**create**”, podemos ejecutar:

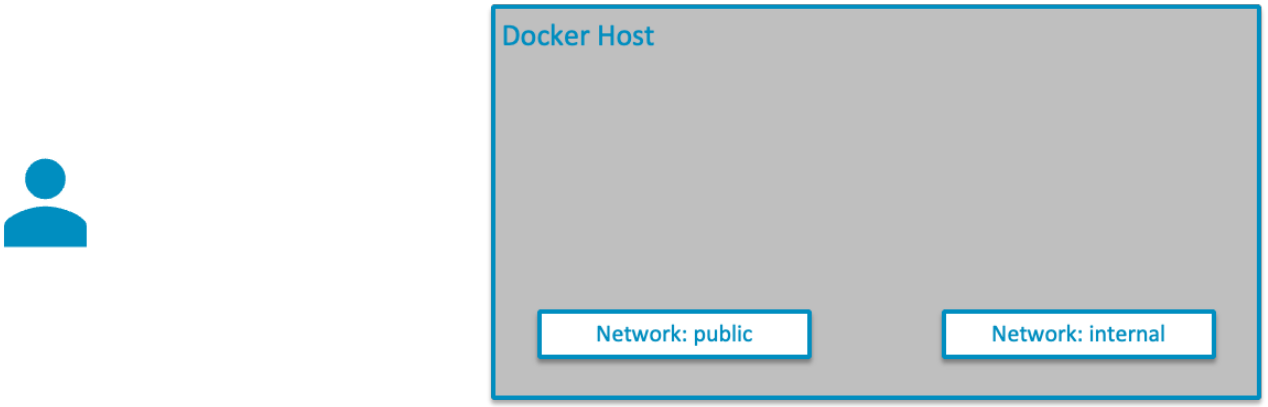
```
docker network create --help
```

Nos queda un último detalle. Dijimos que las redes que íbamos a crear iban a ser de tipo “bridge”, pero ¿cómo sabe el comando docker network create que la red debe ser de este tipo? Es simple, salvo que se indique lo contrario, todas las redes que se crean en Docker son definidas como “bridge” por defecto.

A continuación, vamos a proceder a la creación de la segunda red, pero esta vez el comando va a ser un tanto distinto:

```
docker network create internal --internal
```

Una vez ejecutado este comando, nos encontramos en el siguiente estado:



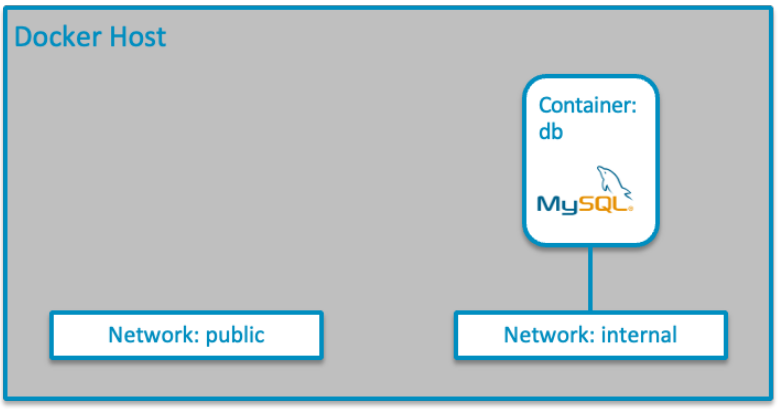
Tip: Si analizamos este segundo comando que acabamos de correr, vamos a ver que está compuesto por cinco términos, los cuatro primeros son iguales al comando anterior (el cuarto término varía, pero eso es porque le vamos a asignar un nombre distinto a la red), pero aparece un nuevo término:

- **--internal**: este término es también un parámetro del comando "create" e indica que la red que vamos a crear no va a permitir que los contenedores que conectemos a ella sean accesibles por fuera de la red. Solo podrán ser accedidos por otros contenedores conectados a la misma red.

En este punto ya deberíamos tener creadas nuestras redes. Vamos a instanciar nuestro primer contenedor, la base de datos.

```
docker run -d --name db --network internal \
-e MYSQL_ROOT_PASSWORD=my-secret-pw \
-e MYSQL_USER=wpuser \
-e MYSQL_PASSWORD=my-secret-pw \
-e MYSQL_DATABASE=wordpressdb \
mysql:5.7
```

Llegado este punto, la implementación se verá de esta manera:



Tip: Este comando tiene muchísimos términos. Vamos a analizarlos:

- **docker**: nuevamente, para invocar al cliente de Docker o Docker CLI (Command Line Interface).
- **run**: este término indica que vamos a ejecutar un nuevo contenedor.
- **-d**: utilizando este término le estamos indicando a Docker que el nuevo container lo vamos a ejecutar en modo “de-attached”, lo que significa que el contenedor se va a ejecutar en segundo plano.
- **--name**: con este parámetro le asignamos un nombre a nuestro contenedor, en este caso “db”.
- **--network**: utilizamos este parámetro para indicar a qué red vamos a conectar nuestro nuevo contenedor.
- **-e**: lo utilizamos para definir variables de entorno dentro del contenedor. Qué es lo que hacen estas variables, cuáles son las variables que debemos definir, o cómo las consumen las aplicaciones que funcionan dentro del contenedor, dependerá enteramente de la aplicación. Para conocer más al respecto podemos visitar la página de Docker Hub de la imagen que vamos a ejecutar. Todas las instrucciones asociadas con las variables disponibles estarán allí documentadas.
- **mysql:5.7**: es el nombre de la imagen (mysql) y tag (5.7) de la imagen que vamos a ejecutar.

Para conocer más sobre los parámetros que podemos utilizar para el comando

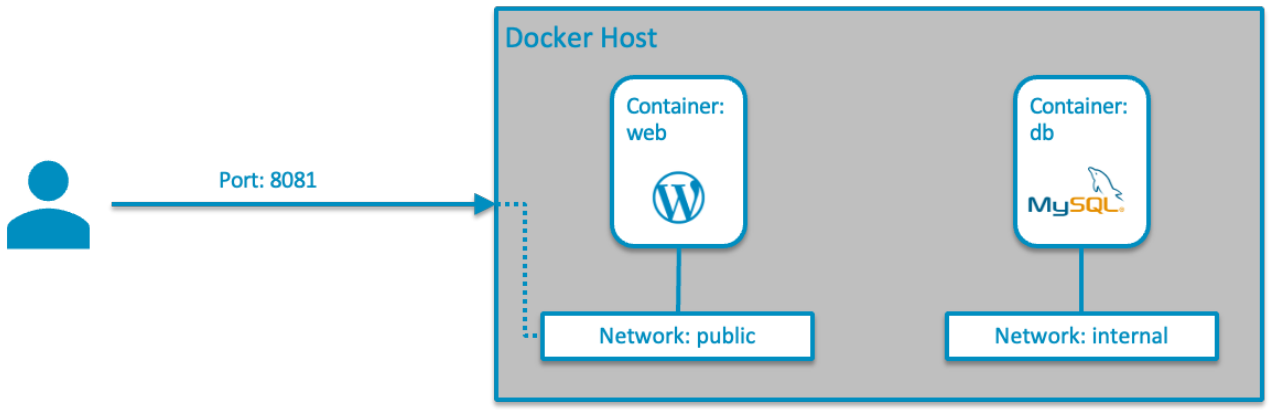
"**run**", podemos ejecutar:

```
docker run --help
```

En este punto ya tenemos las dos redes creadas y uno de los contenedores. Ahora vamos a crear un contenedor adicional para WordPress. Para ello vamos a ejecutar el siguiente comando:

```
docker create -p 8081:80 --name web --network public \
-e WORDPRESS_DB_HOST=db:3306 \
-e WORDPRESS_DB_USER=wpuser \
-e WORDPRESS_DB_PASSWORD=my-secret-pw \
-e WORDPRESS_DB_NAME=wordpressdb \
-e WORDPRESS_TABLE_PREFIX=hj \
wordpress:latest
```

Creando este nuevo container, llevamos la implementación al siguiente estado:



Tip: Este nuevo comando tiene tantos términos como el anterior con algunos cambios sutiles. Vamos a analizarlos:

- create: en lugar de llamar al comando "run", estamos llamando al comando "create". Esto lo hacemos porque queremos crear el contenedor sin iniciarlo. ¿Por qué? Porque antes de iniciarlo queremos conectarlo a otra red (vamos a ver esto en el siguiente comando que ejecutemos).
- -p: este parámetro nos permite mapear un puerto del contenedor, en este caso el puerto 80, a un puerto en nuestra estación de trabajo, en este caso el puerto 8081. De modo que una vez iniciado este contenedor, podemos visitar el puerto 8081 (por ejemplo: https://127.0.0.1:8081) e ingresar a WordPress.

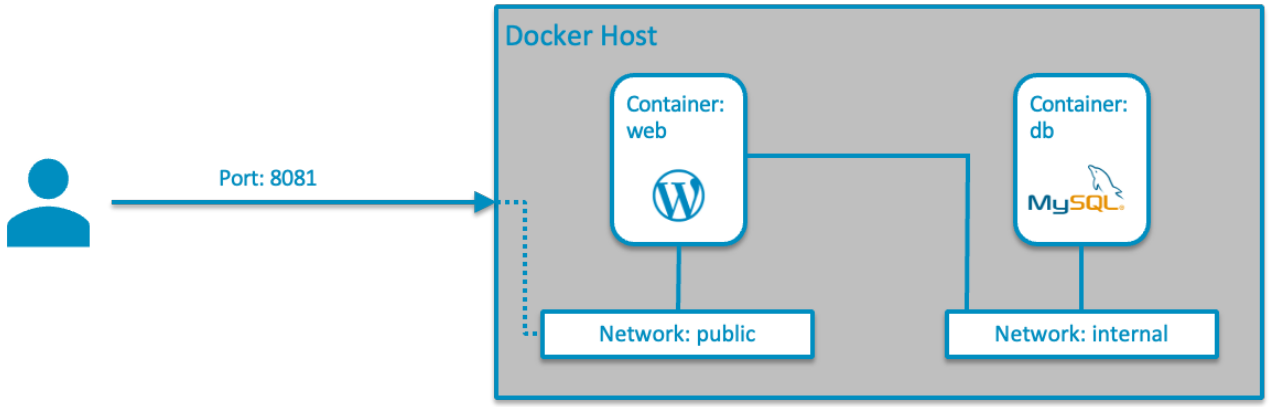
Para conocer más sobre los parámetros que podemos utilizar para el comando "create", podemos ejecutar:

```
docker create --help
```

Ahora vamos a conectar nuestro nuevo container a la red "internal". Eso lo podemos lograr utilizando el siguiente comando:

```
docker network connect internal web
```

Lo cual nos lleva al estado deseado:



Tip: Volvimos a invocar un comando del espacio "docker network", pero en este caso no fue "create", sino "connect". Analicemos qué es lo que logramos con este comando y los argumentos que le enviamos:

- connect: utilizando este comando, dentro del contexto de "network", estamos indicando que deseamos conectar un contenedor existente a una red existente.
- internal: es el nombre de la red a la cual vamos a conectar el contenedor.
- web: es el nombre del contenedor que vamos a conectar a la red.

Para conocer más sobre los parámetros que podemos utilizar para el comando "connect", podemos ejecutar:

```
docker network connect --help
```

Ahora solo nos queda iniciar el contenedor "web" lo cual podemos hacerlo con el siguiente comando:

```
docker start web
```

Habiendo completado todos estos pasos de manera satisfactoria, podemos visitar la URL http://127.0.0.1:8081 y deberíamos encontrarnos con el portal de configuración inicial de WordPress.

Finalmente, para remover todo lo que acabamos de crear, podemos ejecutar los siguientes comandos:

```
docker stop web
docker stop db
docker rm web
docker rm db
docker network rm public
docker network rm internal
docker rmi mysql:5.7
docker rmi wordpress:latest
```