P7: Orquestación de Contenedores con Docker Compose

1. Descripción

El objetivo del ejercicio consiste en desplegar un balanceador de carga con dos servidores WordPress usando Docker Compose.

2. Entorno de prácticas

En estas prácticas se empleará el software de virtualización VIRTUALBOX para simular los equipos GNU/Linux sobre los que se realizarán las pruebas.

3. Imágenes a utilizar

Se proporcionan scripts de instalación tanto para GNU/Linux como para Windows. Windows es bastante inestable con algunas configuraciones de la Máquina Virtual que se usarán durante la realización de las prácticas. Por ello, se recomienda encarecidamente usar Linux como sistema base.

- Script GNU/Linux: ejercicio-docker.sh (desde línea de comandos)

```
01 alumno@pc: $ sh ejercicio-docker.sh
```

- MS Windows: ejercicio-docker.ps1 (desde cmd)

```
01 Powershell.exe -executionpolicy bypass -file ejercicio-docker.ps1
```

Notas:

- Se pedirá un identificador (sin espacios) para poder reutilizar las versiones personalizadas de las imágenes creadas. Se recomienda usar como identificativo CDA_<numero_del_grupo>_<INICIALES>.
- En ambos scripts la variable $DIR_BASE específica donde se descargarán las imágenes y se crearán las MVs.
- Por defecto en GNU/Linux será en $HOME/CDA2425 y en Windows en C:/CDA2425.
- Puede modificarse antes de lanzar los scripts para hacer la instalación de las imágenes en otro directorio más conveniente (disco externo, etc)
- Si se hace desde el script anterior, se pueden arrancar las instancias VIRTUALBOX desde el interfaz gráfico de VirtualBOX o desde la línea de comandos con VBoxManage startvm <nombre MV>_<id>

David Ruano Ordás
Departamento de Informática
Lenguajes y Sistemas Informáticos

4. Credenciales de acceso

La distribución Linux incluida en la MV tiene dados de alta dos usuarios con las siguientes credenciales y permisos:

login	password	permisos
root	purple	root
usuario	purple	permite sudo

5. Entorno desplegado

El entorno desplegado para la realización de las prácticas está formado por una máquinas virtuales sobre la que se desplegarán los diferentes servicios virtualizados en contenedores Docker.

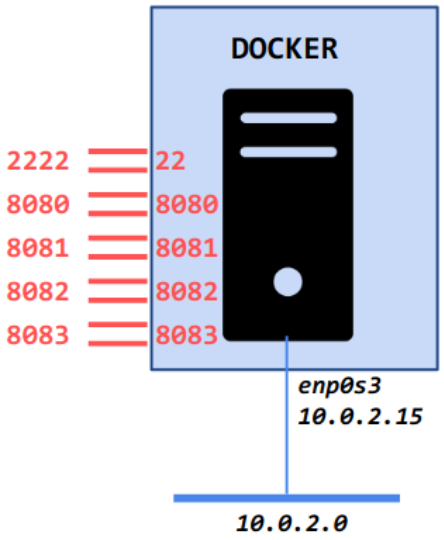

Fig 1. Entorno de trabajo.

- DOCKER tiene la dirección IP privada 10.0.2.15 y será el encargado de desplegar los contenedores que ejecuten los diferentes servicios (balanceador, wordpress, base de datos). Adicionalmente, tiene redireccionados los puertos 8080 a 8085 del anfitrión para las pruebas desde el navegador Web del anfitrión y el puerto 2222 para conexiones SSH (22).

6. Creación de imágenes Docker

Hay dos mecanismos fundamentales para crear imágenes Docker propias:

- Crear una imagen a partir de un contenedor Docker existente (y ya configurado con los ficheros requeridos), empleando el comando *docker commit*
- Crear una imagen a partir de un Dockerfile que describe sus características, empleando el comando *docker build*

6.1. Uso de Dockerfiles

Un Dockerfile es un fichero de texto donde se indican los parámetros de una imagen Docker y los comandos a ejecutar sobre una imagen base para crear una nueva imagen.

El comando docker build construye una nueva imagen (de nombre -t <nombre>) siguiendo las instrucciones del fichero Dockerfile.

- *docker build* recibe como parámetro el directorio donde se encuentra el Dockerfile y otros ficheros o directorios que puedan ser necesarios para la creación de la nueva imagen (mediante las opciones COPY y ADD se incorporarán a la nueva imagen). Ejecuta las instrucciones del Dockerfile una a una. Cada instrucción ejecutada crea una imagen intermedia (que normalmente se desecha) y aporta una nuevo layer a la imagen Docker resultante.
- La imagen resultante estará disponible en el repositorio de imágenes Docker local.
    - Con el comando docker save <imagen> se puede volcar una imagen a un fichero .tar (que después se puede volver a importar con docker load)
    - Es posible publicar imágenes propias en Docker Hub (o en otro Docker Registry privado)
        - Es necesario tener cuenta de usuario en Docker Hub y loguearse con docker login
        - Dar un "nombre" a la imagen con docker tag
        - Y "subirla" a Docker Hub con docker pull

6.2. Formato de los ficheros Dockerfile

- **FROM:** Define la imagen base sobre la que se construirá la nueva imagen
  Sintaxis: FROM <imagen> ó FROM <imagen>:<tag>
- **LABEL:** Permite vincular metadatos (autor, descripción, fecha, etc) a la imagen
  Sintaxis: LABEL <campo> <valor> ó LABEL <campo>=<valor>

- **ENV:** Define variables de entorno que estarán disponibles en el contenedor (tanto durante la creación de la imagen, como durante la ejecución del contenedor)
  Sintaxis: ENV <variable> <valor> ó ENV <variable>=<valor>
- **WORKDIR:** Establece el directorio donde se ejecutará los comandos indicados con RUN/CMD/ENTRYPOINT .
  Sintaxis: WORKDIR <ruta>
- **ADD ó COPY:** Permite agregar/copiar archivos desde el equipo local a la imagen
  Sintaxis: ADD <origen/es>... <destino> ó ADD ["<origen/es>",... "<destino>"]
- **EXPOSE:** Indica los puertos (TCP/IP) en los que escuchará el contenedor (es sólo declarativo, la redirección de puertos del anfitrión es independiente y debe hacerse manualmente al iniciar el contenedor)
  Sintaxis: EXPOSE <puerto/s>
- **VOLUME:** Establece los volúmenes a montar al ejecutar el contenedor.
  Sintaxis: VOLUME <volumen/es>
- **RUN:** Permite ejecutar comandos en la imagen base antes de ser creada (puede haber tantos RUN como sea necesario)
  Sintaxis en formato shell (ejecutado por /bin/sh -c): RUN <ejecutable con parámetros>
  Sintaxis en formato exec: RUN ["<ejecutable>", "<parámetro1>", "<parámetro2>"]
- **ENTRYPOINT:** Define nuevo "punto de entrada" que es el comando que ejecuta por defecto el contenedor al iniciarse (salvo que especifique otro al crear el contenedor mediante docker run --entrypoint <comando>) [sólo se permite un ENTRYPOINT en cada Dockerfile]
  Sintaxis: ENTRYPOINT ["<ejecutable>", "<parámetro1>", "<parámetro2>"] .
- **CMD:** Define la acción a ejecutar por defecto al crear el contenedor (salvo que se especifique otro comando con docker run) [sólo se permite un CMD en cada Dockerfile]. Si hay definido un ENTRYPOINT , la semanatica de CMD cambia y se interpreta su valor como parámetros que se pasarán al comando del ENTRYPOINT-
  Sintaxis en formato exec: CMD ["<ejecutable>", "<parámetro1>", "<parámetro2>"].

6.3. Ejemplo de uso de Dockerfiles

```
01 usuario@docker:~$ mkdir miapache
02 usuario@docker:~$ nano miapache/Dockerfile
```

```
03 FROM debian:latest
04 EXPOSE 80
05 RUN apt-get update && apt-get install -y apache2 && apt-get clean
06 ADD index.html /var/www/html/
07 CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

```
01 usuario@docker:~$ nano miapache/index.html
```

```
01 <html>
02     <body>
03         <h1>Prueba index.html por defecto</h1>
04     </body>
05 </html>
```

```
01 usuario@docker:~$ docker build -t cda/apache:v1.0 miapache
```

```
01 => [internal] load build definition from Dockerfile
02 => [1/3] FROM docker.io/library/debian:latest
03 ...
04 => [2/3] RUN apt-get update && apt-get install -y apache2 && apt-get clean
05 ...
06 => [3/3] ADD index.html /var/www/html/
07 ...
08 => exporting to image
09 => => exporting layers
10 => => writing image
11 => => naming to docker.io/cda/apache:v1.0 Successfully built c8c685b91440
12 usuario@docker:~$ docker images
13 REPOSITORY TAG IMAGE ID CREATED SIZE
14 cda/apache v1.0 854f4c556fac 3 minutes ago 253MB
```

```
01 usuario@docker:~$ docker run -d -p 8080:80 --name servidor1 cda/apache:v1.0
02 usuario@docker:~$ docker ps
```

```
01 CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
02
03 NAMES
04
05 f21d893c261a cda/apache:v1.0 "/usr/sbin/apache2ct..." 7 seconds ago Up 6
06 seconds
```

Acceder a http://localhost:8080

```
usuario@docker:~$ docker stop servidor1
usuario@docker:~$ docker rm servidor1
```

```
nano miphp/start.sh
```

```
#!/bin/bash
sed -i "s/__HUECO__/$MENSAJE/g" /var/www/html/phpinfo.php
/usr/sbin/apache2ctl -D FOREGROUND
```

```
usuario@docker:~$ nano miphp/phpinfo.php
```

```
<html>
    <body>
        <h1>Prueba PHP</h1>
        <h2> Mensaje: __HUECO__ </h2>
        <h2> PHPInfo </h2>
        <?php echo phpinfo();?>
    </body>
</html>
```

```
usuario@docker:~$ docker build -t cda/php:v1.0 miphp
```

```
=> [internal] load build definition from Dockerfile
=> [1/5] FROM docker.io/cda/apache:v1.0
...
=> [2/5] RUN apt-get update && apt-get install -y libapache2-mod-php php &&
apt-get clean
...
=> [3/5] ADD phpinfo.php /var/www/html/
...
=> [4/5] ADD start.sh /root
...
=> [5/5] RUN chmod +x /root/start.sh
...
=> exporting to image
=> => exporting layers
=> => writing image
=> => naming to docker.io/cda/php:v1.0
```

```
usuario@docker:~$ docker images
```

```
REPOSITORY TAG IMAGE ID CREATED SIZE
cda/php v1.0 9a67661322de 2 minutes ago 286MB
cda/apache v1.0 854f4c556fac 11 minutes ago 253MB
```

```
usuario@docker:~$ docker run -d -p 8081:80 --name servidor2 cda/php:v1.0
usuario@docker:~$ docker run -d -p 8082:80 -e MENSAJE="saludos de Pepe"
--name servidor3 cda/php:v1.0
usuario@docker:~$ docker ps
```

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS

NAMES

c3a8aa97b844 cda/php:v1.0 "/root/start.sh" 6 seconds ago Up 5 seconds
0.0.0.0:8082->80/tcp, :::8082->80/tcp servidor3
2a8fea52b99f cda/php:v1.0 "/root/start.sh" 15 seconds ago Up 14 seconds
0.0.0.0:8081->80/tcp, :::8081->80/tcp servidor2
```

Desde el anfitrión, acceder a http://localhost:8081/phpinfo.php y http://localhost:8082/phpinfo.php

```
usuario@docker:~$ docker stop servidor1 servidor2 servidor3
usuario@docker:~$ docker rm servidor1 servidor2 servidor3
```

7. Despliegues multicontenedor

La herramienta docker compose (en versiones anteriores *docker-compose*) permite desplegar un conjunto de contenedores a partir de las descripciones aportadas en un fichero docker-compose.yml

Importante: Dependiendo de la versión de Docker instalada el comando a usar en esta sección será:

- docker-compose , en versiones "antiguas" de Docker (la que se instala desde los paquetes Debian)
- docker compose , en las últimas versiones recientes de Docker (es la que está instalada en los equipos físicos del laboratorio)

7.1. Uso de docker compose

Fichero en formato YAML (YAML Ain't Markup Language). Ver sintaxis YAML en https://yaml.org/. Especificación del fichero docker-compose.yml en https://docs.docker.com/compose/intro/compose-application-model/#the-compose-file
Elementos de primer nivel del objeto YAML:

- Elemento versión (opcional): versión del formato utilizada
- Elemento name (opcional): nombre/descripción del escenario
- Elemento services: lista de los contenedores/services a ejecutar, cada uno identificado por su nombre y acompañado de sus parámetros de creación (imagen, redirecciones de puertos, volúmenes, redes a usar, etc). Cada service puede tener:
    - atributo image : nombre la imagen base del contenedor
    - atributo build : detalles para la creación del contenedor, incluyendo ruta a Dockerfile para crear su imagen
    - atributos command / entrypoint : "sobreescribe" el comando de inicio por defecto definido en la
    - imagen base
    - atributo configs : parámetros de configuración adicionales
    - atributo depends_on : lista con otros contenedores de los que depende el contenedor definido (no se inicia hasta que esas dependencias se hayan iniciado)
    - atributo environment : lista con las variables de entorno a establecer en el contenedor
    - atributo networks : lista de redes a las que se conectará el contenedor
    - atributo port : lista de puertos expuestos y/o redireccionados
    - atributo volumes : lista de volúmenes a montar
    - otros: container_name , labels cpus , mem_limit`, etc
- Elemento networks (opcional): define la lista de redes a crear en el escenario descrito, vinculando a cada nombre de red sus parámetros (driver , ipam [con detalles de la subred], etc).
- Elemento volumes (opcional): define la lista de volúmenes a usar por los contenedores y los detalles de cada uno.
- Elemento configs (opcional): define la lista de "configuraciones" usadas por los contenedores declarados, con los detalles de cada una

- Elemento secrets (opcional): tipo particular de "configuraciones" con datos sensibles (contraseñas, claves, certificados, etc)
- En escenarios sencillos esa red será suficiente para la comunicación entre contenedores, en escenarios más complejos se pueden definir redes adicionales en el elemento de primer nivel networks y conectar cada contenedor a las redes mediante el atributo networks .
- En esa red por defecto creada por docker compose , la resolución de nombres DNS se puede realizar tanto por el nombre del contenedor (atributo container_name) como el nombre del service definido en el elemento de primer nivel services.

7.2. Comandos de *docker compose*

Ejecutable docker compose (en versiones anteriores docker compose) debe ejecutarse en el mismo directorio donde se ubica el fichero docker-compose.yml con la configuración del escenario (o indicar con -f la ruta del fichero con la configuración del escenario).

- *docker compose up*: Crear los contenedores (servicios) descritos en docker-compose.yml (con la opción -d crear los contenedores en modo dettach, sin mostar los logs de inicio) (con la opción -f se puede indicar el fichero YAML con la configuración en caso de no usar el nombre por defecto)
- *docker compose down*: Para y borra los contenedores y las redes (si las hubiera) creados con *docker compose up*
- *docker compose config* : Hace una comprobación de sintaxis del fichero docker-compose.yml , mostrando la "versión completa" del fichero si todo está correcto.
- *docker compose stop|restart*: Detiene/reinicia los contenedores que previamente se han lanzado con *docker compose up*.
- *docker compose run*: Inicia los contenedores descritos en el docker-compose.yml que estén parados.
- *docker compose rm*: Borra los contenedores parados del escenario.
- *docker compose pause|unpause*: Pausa los contenedores que previamente se han lanzado con docker compose up , o reanuda los previamente pausados
- *docker compose build*: En caso de haberse indicado en la sección build Dockerfile con definiciones de images usadas en el docker-compose.yml se construyen esas imágenes

- *docker compose logs <servicio (opc.)>*: Muestra los logs de todos los servicios del escenario o de uno en concreto
- *docker compose exec <servicio> <comando>*: Ejecuta un comando en el contenedor indicado, declarado como <servicio> en el docker-compose.yml
- *docker compose top*: Muestra los procesos que están ejecutándose en cada uno de los contenedores creados.

7.3. Despliegue de contenedores con volúmenes compartidos

```
usuario@docker:~$ mkdir ejemplo
usuario@docker:~$ cd ejemplo/
usuario@docker:~/ejemplo$ nano docker-compose.yml
```

```
services:
   contenedor1:
      container_name: apache1
      image: cda/php:v1.0
      restart: always
      hostname: apache1
      ports:
         - "8081:80"
      volumes:
         - ./comun_html:/var/www/html/extra
   contenedor2:
      container_name: apache2
      image: cda/php:v1.0
      restart: always
      hostname: apache2
      environment:
         MENSAJE: "Hola desde Apache2"
      depends_on:
         - contenedor1
      ports:
         - "8082:80"
      volumes:
         - ./comun_html:/var/www/html/extra
```

```
usuario@docker:~/ejemplo$ mkdir comun_html
usuario@docker:~/ejemplo$ echo "Servido por: <?php echo gethostname(); ?>" >
comun_html/index.php
usuario@docker:~/ejemplo$ docker compose config #Valida la configuracion
usuario@docker:~/ejemplo$ docker compose up -d
```

```
[+] Running 3/3
```

```
02  ✔ Network ejemplo_default Created
03  ✔ Container apache1 Started
04  ✔ Container apache2 Started
```

```
01  usuario@docker:~/ejemplo$ docker ps
```

```
01  CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS  NAMES
02
03  941d4f9b1f9b cda/php:v1.0 "/root/start.sh" 38 seconds ago Up 36 seconds
04  0.0.0.0:8082->80/tcp, :::8082->80/tcp apache2
05  b84a7928d6b3 cda/php:v1.0 "/root/start.sh" 38 seconds ago Up 36 seconds
05  0.0.0.0:8081->80/tcp, :::8081->80/tcp apache1
```

```
01  # Consultar los log de arranque de los contenedores/services
02  usuario@docker:~/ejemplo$ docker compose logs contenedor1
03  usuario@docker:~/ejemplo$ docker compose logs contenedor2
```

```
01  # Ejecutar el shell dentro del contenedor
02  usuario@docker:~/ejemplo$ docker exec -it apache1 /bin/bash
```

```
01  root@apache1:/# apt-get install -y -q iproute2 iputils-ping
02  ...
03
04  root@apache1:/# ip address
05  1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
06  default qlen 1000
07  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
08  inet 127.0.0.1/8 scope host lo
09  valid_lft forever preferred_lft forever
10  68: eth0@if69: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
11  state UP group default
12  link/ether 02:42:ac:15:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
13  inet 172.XX.0.2/16 brd 172.21.255.255 scope global eth0
14  valid_lft forever preferred_lft forever
15
16  root@apache1:/# ping 172.XX.0.3
17  PING 172.XX.0.3 (172.XX.0.3) 56(84) bytes of data.
18  64 bytes from 172.XX.0.3: icmp_seq=1 ttl=64 time=0.416 ms
19  root@apache1:/# exit
```

Desde el anfitrión, acceder a

- http://localhost:8081/extra y a http://localhost:8082/extra (se verá el contenido de la carpeta compartida comun_html , montada /var/www/html/extra de cada contenedor)
- http://localhost:8081/phpinfo.php y a http://localhost:8082/phpinfo.php (se verá el resultado de la establecer la variable MENSAJE de la imagen base)

```
#Eliminar escenario
usuario@docker:~/ejemplo$ docker compose down
```

```
usuario@docker:~/ejemplo$ docker compose down
[+] Running 3/3
✔ Container apache2 Removed
✔ Container apache1 Removed
✔ Network ejemplo_default Removed
```

```
usuario@docker:~/ejemplo$ docker ps
```

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

```

8. Ejercicios

Tarea 1: Creación de una imagen personalizada para el despliegue de un entorno web con Flask y python

La tarea a desarrollar se basa en la creación de una imagen personalizada que englobe un entorno web usando el Flask y Python. Para ello se describen los siguientes pasos a realizar:

- Crear un directorio llamado flask-app.
- Dentro de flask-app, crear:
    - Un archivo app.py que tenga una simple aplicación web usando Flask

app.py

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
```

```
return "¡Bienvenido a mi aplicación Flask en Docker!"

if __name__ == "__main__":
app.run(host="0.0.0.0", port=5000)
```

- Crear el fichero requirements.txt que contenga las dependencias necesarias.

```
requirements.txt
```

```
Flask==1.1.2
Werkzeug==1.0.1
```

- Crear el Dockerfile con las siguientes especificaciones:
  - Usar python:3.8-slim como imagen base.
  - Establecer el directorio de trabajo en /app
  - Copiar requirements-txt y app.py al contenedor
  - Instalar las dependencias usando pip (pip install -r requirements.txt)
  - Definir una variable de entorno llamada FLASK_ENV con valor development
  - Exponer el puerto 5000.
  - Iniciar la aplicación Flask

- Construir la imagen
- Ejecutar el contenedor mapeando el puerto expuesto y conectacte desde localhost para comprobar que funciona correctamente

Tarea 2: Despliegue de un escenario PHPMyAdmin y MariaDB usando Docker Compose.

La tarea consiste en automatizar, mediante Docker Compose, el proceso de creación y gestión de un escenario multicontenedor siguiendo el esquema que se muestra en la siguiente figura.

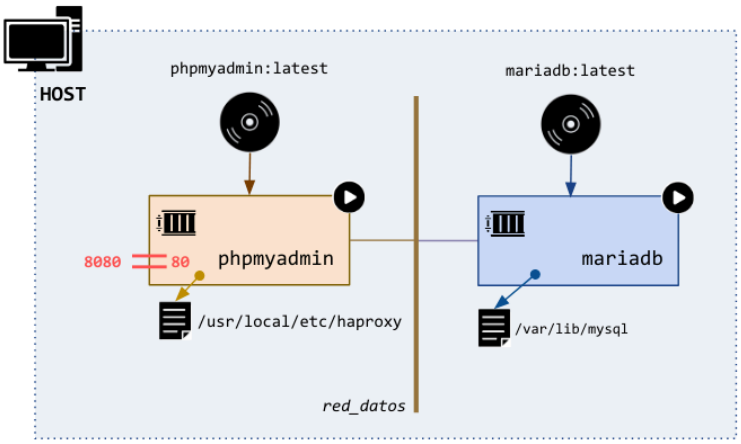

Fig 2. Entorno a desplegar.

Tal y como se puede observar en Fig 2, se deben desplegar un total de cuatro contenedores que ofrecen diferentes servicios:

- Una red docker de tipo bridge que permitirá que los contenedores phpmyadmin y mariadb se comuniquen entre sí.
- Un contenedor phpmyadmin que permita administrar la base de datos mariadb.
- Un contenedor que ejecute el servidor de base de datos mariadb

Pasos a realizar (usando docker compose):

- Crear el directorio de configuración y el fichero docker-compose.yml, junto con los demás elementos necesarios (directorios, ficheros, etc)
- Aportar los ficheros de configuración usados y (si es pertinente) explicar su contenido.

Tarea 3 : Despliegue de un cluster de balanceo de carga para un servidor WordPress (con MariaDB).

La tarea consiste en realizar automáticamente mediante Docker Compose el despliegue de contenedores Docker siguiendo el esquema que se muestra en la siguiente figura:

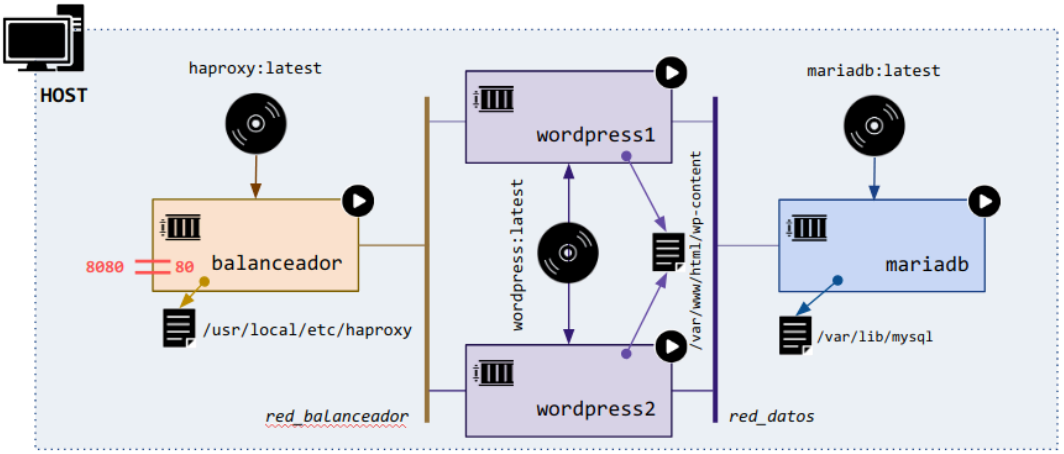

Fig 3. Entorno a desplegar.

Tal y como se puede observar en Fig 3, se deben desplegar un total de cuatro contenedores que ofrecen diferentes servicios.

- Un contenedor HAProxy haciendo de frontal (con el puerto 80 de ese contenedor redirigido al puerto 8080 del anfitrión para verificar el acceso)
- Dos contenedores con el gestor de contenidos Wordpress instalado sobre los que repartirá las peticiones el balanceador HAProxy
- Un contenedor con una base de datos MariaDB compartida por los dos Wordpress.

Tal y como se puede observar, *balanceador*, *wordpress1* y *wordpress2* deben pertenecer a la misma red ya que el *balanceador* debe poder distribuir las diferentes peticiones entre contenedores WordPress. Por otro lado, tanto wordpress1 como wordpress2 requieren acceso a la base de datos (mariadb). Esto implica que comparta red con el contenedor *mariadb*.

Pasos a realizar:

1. Crear en el anfitrión los directorios a usar con -v (volume mapping)
    a. Para /var/lib/mysql del contenedor MariaDB
    b. Para /var/www/html/wp-content de los contenedores Wordpress
    c. Para /usr/local/etc/haproxy del contenedor HAProxy
2. Crear el fichero de configuración de HAProxy (haproxy.cfg) en frontend ajustar bind a 0.0.0.0:80 ó *:80 en backend ajustar la dirección de los server usando los nombres de los contenedores Wordpress
3. Crear las redes necesarias

4. Crear los contenedores necesarios, ajustando las variables de entorno (con -e) según corresponda
    a. Wordpress:
        i. WORDPRESS_DB_HOST
        ii. WORDPRESS_DB_NAME
        iii. WORDPRESS_DB_PASSWORD
        iv. WORDPRESS_DB_USER
    b. MariaDB
        i. MARIADB_DATABASE
        ii. MARIADB_USER
        iii. MARIADB_PASSWORD
        iv. MARIADB_ROOT_PASSWORD

Tarea 4 : Entregable (memoria)

1. Formato: PDF
2. Nombre: Practica7_(apellidos_nombre).pdf
3. Describir:
    - Tarea 1:
        - Aportar el Dockerfile creado
        - Aportar los comandos utilizados (construcción de la imagen, creación/ejecución del contenedor).
        - Aportar el resultado obtenido al conectarse al contenedor desde el navegador.
    - Tarea 2 y Tarea 3:
        - Aportar los comandos utilizados, la salida que dan (si es relevante) y una breve descripción de lo que hace cada uno.
        - Añadir capturas del resultado final una vez se acceda mediante un navegador al anfitrión con la URL http://localhost:8080
        - NOTA: No es necesario completar la instalación de los Wordpress basta con constatar que se muestra la página de configuración.