CLOUD COMPUTING, PRÁCTICA 2: PAAS

Manuel Blanco Rienda

MASTER INGENIERÍA INFORMÁTICA, UGR manuelbr@correo.ugr.es

1. Introducción

El planteamiento de la práctica se basa en la creación de un servicio web, alojado en un contenedor de “docker” que accede a otro contenedor diferente para obtener un servicio de Base de Datos, necesario para desempeñar su función. El servicio web que he implementado para su despliegue en esta arquitectura es el mismo que usé en la práctica anterior.

2. Creación de la arquitectura

El diseño de la arquitectura que se ha usado en esta práctica es prácticamente el mismo que se usó en la anterior: Ubuntu 14.04 como sistema operativo sobre el que se aloja un servidor apache en el que se encuentra el servicio web y Centos 7 como sistema en el que se aloja un servidor mysql que proporciona el servicio de base de datos. Con esta variedad potencio la heterogeneidad del servicio.

2.1. Creación del contenedor Apache

Lo primero que hay que destacar es el hecho de que la imagen que usado para este contenedor contiene una versión ligera de Ubuntu 14.04 con usuario root por defecto, apache, php5 y ssh instalados y ya configurados. He obtenido esta imagen de dockerhub, la página de alojamiento de imágenes oficial de Docker.

1- Para descargar la imagen mencionada uso el comando: “docker pull eboraas/apache-php”.
2- Ahora es el momento de ejecutar el contenedor y que empiece a funcionar. Para ello utilizo el comando: “docker run -i -d -p 14010:80 --name apacheManuelBlanco eboraas/apache-php. Con este comando le proporcionamos un nombre al contenedor y re-direccionamos el puerto 14010 (que tengo asignado como alumno de la asignatura) al 80, donde espera el servidor apache.
3- Con docker exec -i -t apacheManuelBlanco /bin/bash accedo a la Shell del sistema operativo del contenedor.
4- Compruebo la disponibilidad del servicio de apache con el comando: "service apache2 status". Una nota a destacar es que en este contenedor ya entramos en modo root desde el inicio por lo que no hay que acompañar las órdenes que usamos con "sudo".
5- Instalo git con el comando "apt-get install git" (previamente se debe haber hecho “apt-get update” para actualizar los repositorios) para poder hacer el despliegue de la app web desde mi repositorio de git.
6- Obtengo la ip de la máquina que tendré que dársela al servidor de base de datos, con el comando “ifconfig” (para tenerlo disponible debo instalar antes el paquete “net-tools”).

7- Para desplegar la app ejecuto el comando en la Shell del contenedor: “https://github.com/manuelbr/Master_Cloud_Computing_2.git”, con lo cual obtengo una copia local de mi repositorio de la asignatura en github.
8- Elimino los archivos de la app por defecto que viene con apache mediante: “rm –r /var/www/html”. Con el comando “mv /Master_Cloud_Computing_2/practica_2/src /var/www/html” introducimos el servicio web en apache, que ya estará disponible desde el navegador, tal y como puede verse a continuación:

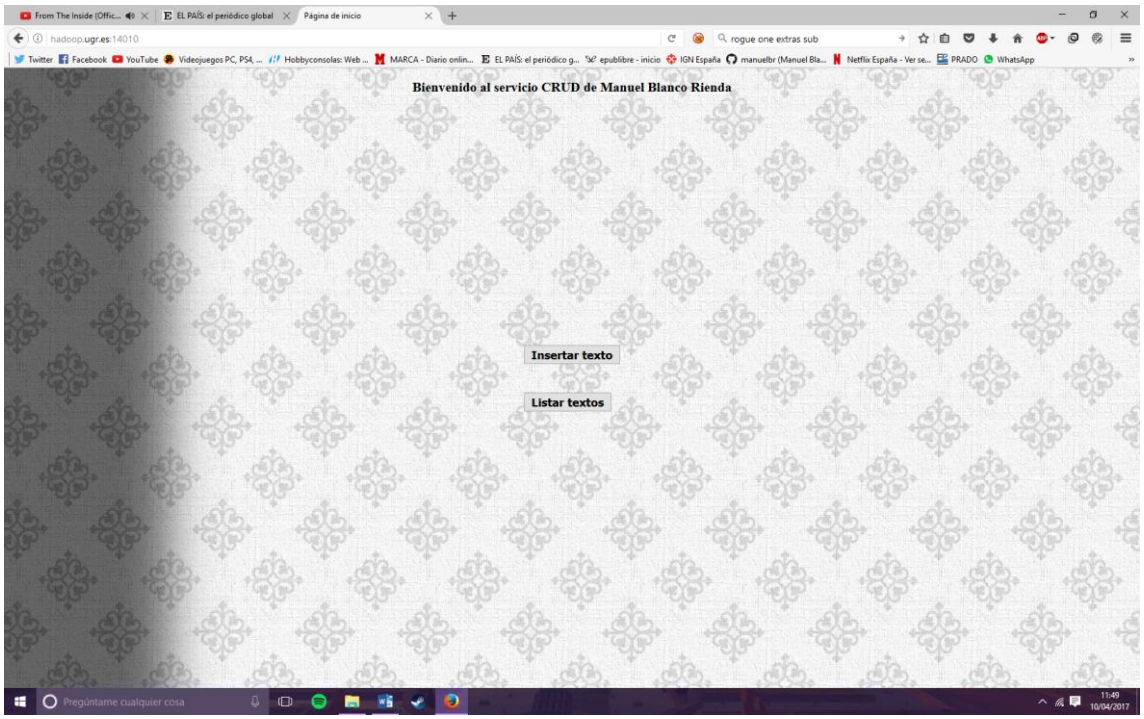

Figura 1: Muestra del funcionamiento del servicio web en el contenedor de apache

2.2. Creación del contenedor MySql con Centos 7

Lo primero que hay que destacar es el hecho de que la imagen que usado para este contenedor contiene una versión ligera de Centos 7 con usuario root por defecto y mysql instalado y ya configurado. He obtenido esta imagen de dockerhub, la página de alojamiento de imágenes oficial de Docker. El nombre de dicha imagen es: “jdeathe/centos-ssh-mysql” y la url donde puede obtenerse es: “https://hub.docker.com/r/jdeathe/centos-ssh-mysql/”.

1- Nos bajamos la mencionada imagen con mysql que deseamos. Esto lo hago con el comando "docker pull jdeathe/centos-ssh-mysql".
2- Arrancamos el contenedor con la imagen que nos hemos bajado con el comando "docker run -i -d -p 14011:3306 --name "mysqlManuelBlanco" --env "MYSQL_ROOT_PASSWORD=contraseñar" jdeathe/centos-ssh-mysql", en el que redirigimos al puerto 3306 las entradas por el puerto 14011 (el que tengo asignado como alumno para mi máquina secundaria) de centos,

le damos el nombre a la máquina: "mysqlManuelBlanco" y le especificamos la contraseña que tendrá el usuario root (la misma que se especifica en los archivos fuente de la aplicación web).

3- Pasamos a entrar dentro de la máquina creada con el comando "docker exec -it mysqlManuelBlanco /bin/bash".
4- Comprobamos que el servicio mysql esté corriendo con "service mysqld status".
5- Con el comando mysql -u root -p entramos en la consola de mysql para configurar las tablas que necesitaremos.
6- Introducimos “CREATE DATABASE textos;” en la shell de mysql, para crear la tabla desde la que haremos referencia en la web.
7- Por último, garantizaremos el acceso al servidor mysql desde la ip del servicio web que obtuve en el apartado anterior, con el comando “GRANT ALL ON *.* to root@'ip' IDENTIFIED BY 'contraseña';”
8- Para aplicar los cambios en los privilegios, en la misma Shell de mysql en la que hemos tecleado los anteriores comandos uso: “FLUSH PRIVILEGES;”. Desde este momento podremos ver que el servicio web lleva a cabo sus funciones correctamente desde “http://hadoop.ugr.es:14010”:

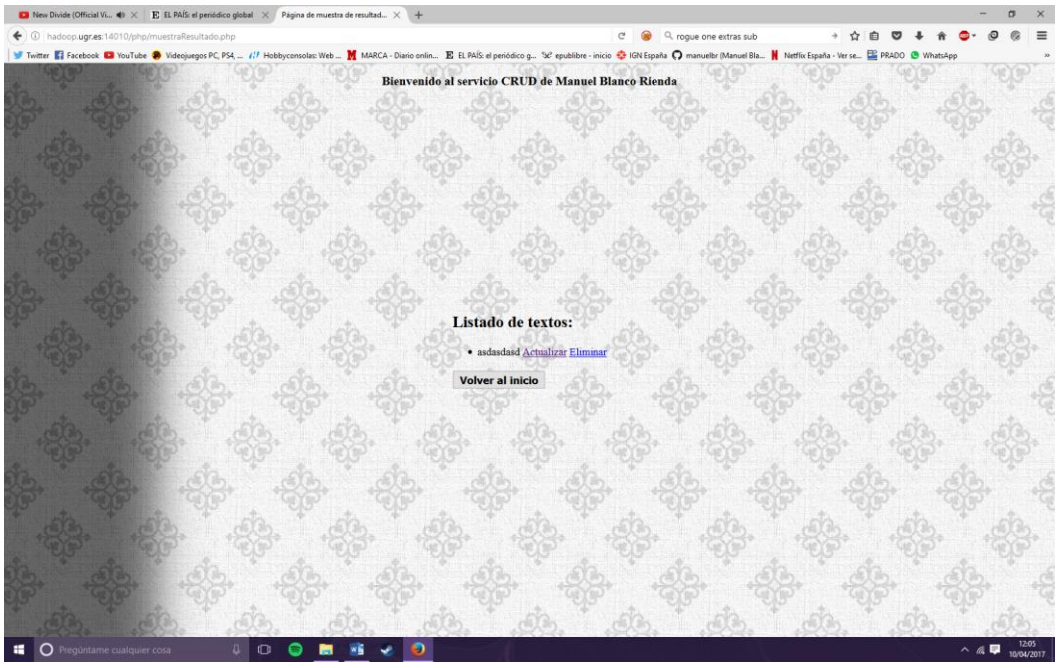

Figura 2: Muestra del funcionamiento del servicio de base de datos en el contenedor

3. Descripción de la aplicación web

Tal y como se ha dicho anteriormente, la aplicación consiste en un servicio de lectura, escritura, actualización y eliminación de cadenas de texto, es decir: un CRUD. Su funcionalidad es la siguiente: cualquier usuario puede acceder a la app a través de la dirección: “hadoop.ugr.es:15065” y a partir de ahí puede insertar textos o listar los que ya hay en la base de datos. Si accedemos al listado, cada uno de los ítems que lo componen pueden ser actualizados (sustituidos) por un nuevo texto o bien ser eliminados de la base de datos.

En cuanto a la arquitectura software, la aplicación en sí se encuentra alojada en un servidor apache dentro de un contenedor con Ubuntu 14.04 y php5. A través de un servidor mysql alojado en un contenedor Centos 7 y ssh, la aplicación obtiene acceso a la base de datos que necesita para funcionar: textos. Es la propia aplicación, una vez conectada a la base de datos la que crea la única tabla que se almacena: textos (se llama igual que la base de datos), la cual consta de solo dos campos: id (entero, llave primaria de la tabla y auto-incrementable) y texto (tipo text no nulo).

Sobre la arquitectura de la aplicación web, decir que se ha implementado usando php, html para el contenido de la misma y css para determinar su estilo (muy simple y sencillo, dado el objetivo de la práctica). El directorio de archivos fuente se distribuye en un archivo index.php, que conforma la página principal de la web, una carpeta de imágenes (que contiene el fondo de la web), una carpeta php que contiene los scripts en ese lenguaje y otra css, que almacena lo propio al estilo de la aplicación. El flujo que se sigue en el proceso de ejecución de la web es el siguiente:

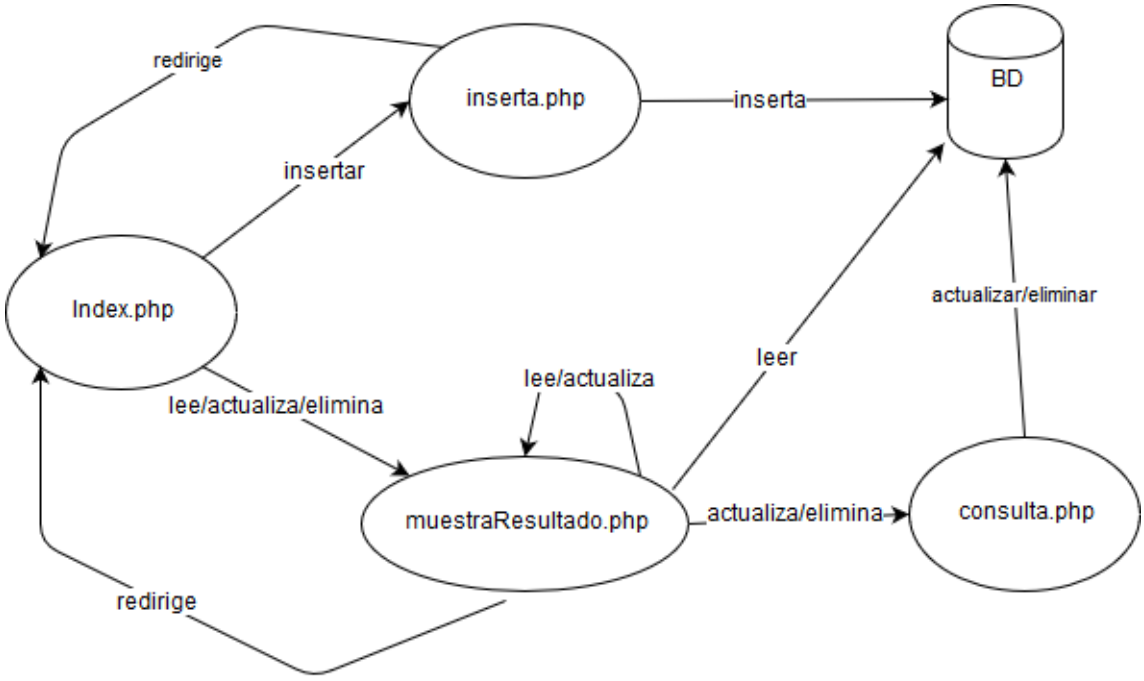

Figura 3: Flujo de procesos entre los fuentes de la aplicación

4. Análisis del comportamiento de los contenedores duplicados

Tal y como ha especificado el profesor, la finalidad de este apartado dentro de la práctica era el de usar un balanceador de carga para sobrecargar el sistema y ver como se comportaba el mismo con dicho balanceador. Sin embargo, tal y como ha explicado el profesor, el sistema que estamos utilizando en la asignatura para las prácticas, no está preparado para este tipo de "experimentos" y es por ello que esto se deja de lado.

Sin embargo, hay un par de preguntas relacionadas con la finalidad de este apartado, que deben ser contestadas:

1 - ¿Qué pasaría si realmente se incrementa o decrementa la petición de recursos?

La respuesta a esto, es que con un solo contenedor para el servidor web y un solo contenedor para el sistema de gestión de bases de datos, lo normal es que si las peticiones aumentan en una de las dos partes, el sistema podría colapsar, al no poder dar respuesta a tal cantidad de peticiones. En caso de que el número de peticiones fuera muy pequeño, tal vez podríamos estar desaprovechando todo el potencial de nuestro contenedor en una tarea esporádica.

En cualquiera de las dos situaciones planteadas anteriormente, una posible solución podría ser el utilizar un balanceador de carga asociado a contenedores duplicados, de forma que en caso de aumentar la demanda de peticiones, el balanceador distribuyera la carga entre los contenedores duplicados, de cara a obtener un mejor rendimiento del sistema completo. Si por otro lado, el número de peticiones fuera ínfimo, sería bueno que el balanceador de carga pudiera dar de lado a los contenedores duplicados (bien "apagándolos" para que no consumieran recursos o bien dándoles otras tareas donde fueran más productivos).

2- ¿Qué pasaría si por cualquier cuestión uno de los servicios cayese?

Si no tenemos pensado un sistema de balanceo y duplicación de contenedores, el sistema entero dejaría de funcionar, ya que los dos tipos de contenedores son indispensables para el desempeño de la función del servicio web.

Lo ideal sería que los contenedores estuvieran duplicados y que en caso de que uno cayera, se redirigiera el tráfico a la copia del mismo, para dar continuidad ininterrumpida al servicio.

5. Instalación y prueba del contenedor OwnCloud

La instalación y puesta en marcha del servicio OwnCloud se ha realizado de forma muy simple con el contenedor: “owncloud”, hecho por la propia empresa de OwnCloud:

1- Para bajar el contenedor uso: “docker pull owncloud”.
2- Ahora, uso el comando “docker run -i -d -p 14012:80 --name "owncloudManuelBlanco" owncloud” para arrancar un contenedor de nombre owncloudManuelBlanco y con el puerto 80 recibiendo todas las entradas desde el 14012.
3- Desde el momento en que se ejecuta el anterior comando, ya puedo acceder de forma externa al servicio de owncloud, desde la dirección: http://hadoop.ugr.es:14012:

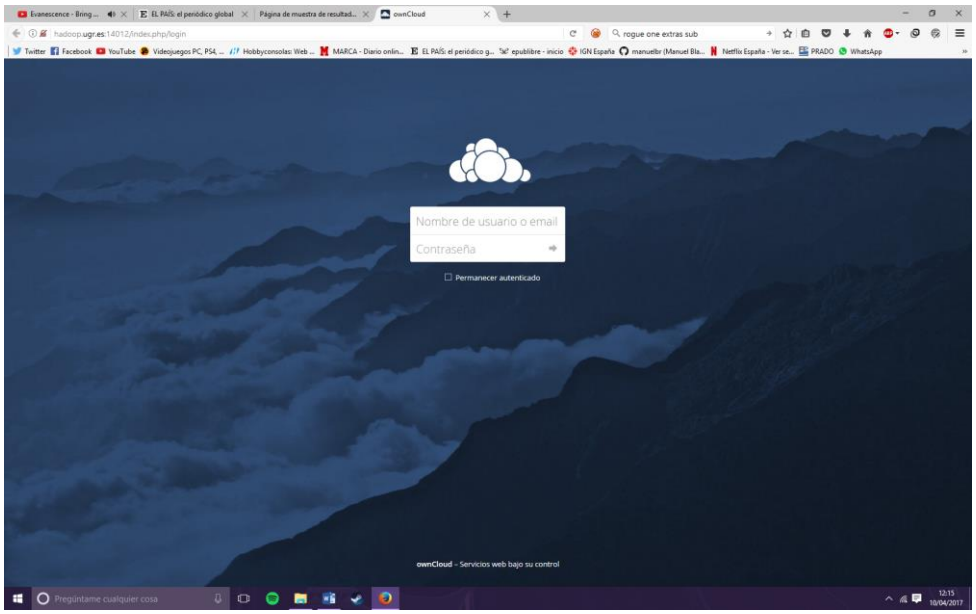

Figura 4: Pantalla de login, una vez nos hemos registrado (en una pantalla exactamente igual)

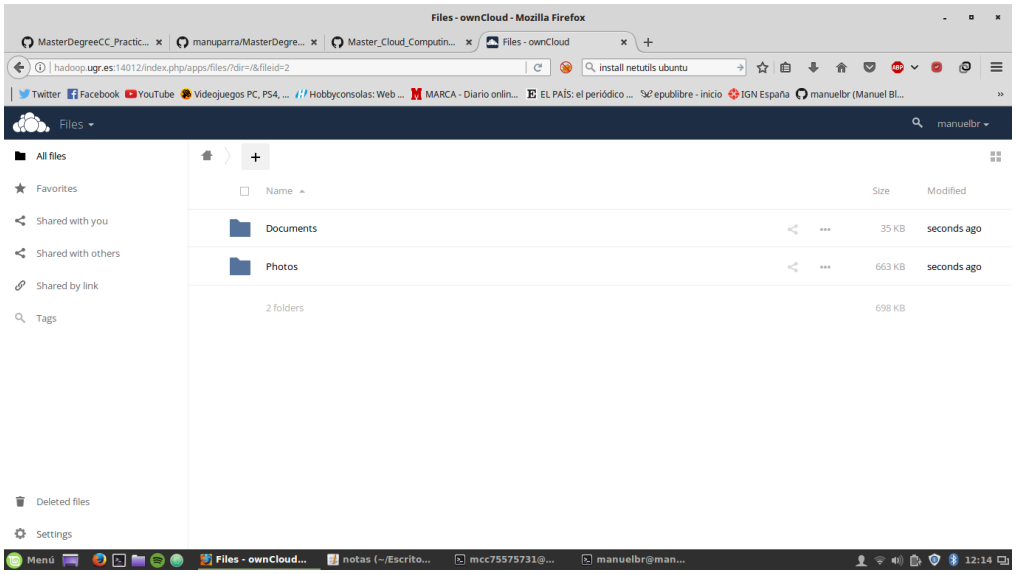

Figura 5: Pantalla de inicio de OwnCloud con el árbol de directorios que tengo

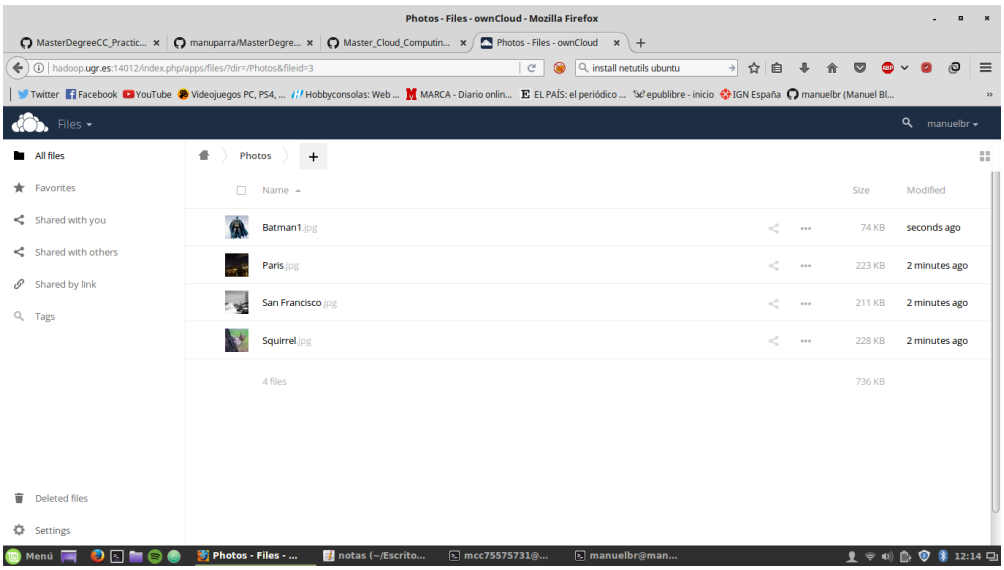

Figura 6: Pantalla con las imágenes almacenadas por defecto en la carpeta “Photos” más una que yo mismo he subido: “Batman1.png”

Con todo ello, vemos finalmente que los dos servicios implementados a través de tres contenedores diferentes funcionan perfectamente. Las direcciones donde pueden accederse a los dos servicios son:

- Servicio CRUD web: http://hadoop.ugr.es:14010/
- Servicio OwnCloud: http://hadoop.ugr.es:14012/