Universidad de Granada

Cloud Computing: Servicios y Aplicaciones

Uso de contenedores (Docker)

Marvin Matías Agüero Torales

maguero@correo.ugr.es

Curso 2016-2017

Sumario

Objetivo

Familiarizarse con el uso de una plataforma PaaS y desarrollar habilidades de despliegue de contenedores y configurar aplicaciones sencillas en los mismos.

Enunciado

1. Crear un contenedor docker (A) con APACHE, SSL, y PHP5[1].
   1.1. ¿Cuál es el puerto SSL?
   1.2. ¿Cómo redirigir el puerto SSL a vuestro puerto asignado?
2. Crear un contenedor docker (B) con MySQL
3. Crear una página en el servidor Web (A) que se conecte al servicio MySQL en el contenedor(B)
4. Duplicar los contenedores A y B y discutir o mostrar qué pasaría si uno de ellos cayera.
5. Desplegar un servicio OwnCloud o NewCloud en otro contenedor (o eliminar alguno de los ya utilizados) y chequear su correcto funcionamiento almacenando archivos.

Despliegue de un contenedor en Docker

Configuración

Para este trabajo se desplegaron seis contenedores: tres para el servicio Web (CA), el balanceador de carga (CA0) y dos esclavos (CA1 y CA2); dos para el servicio de Sistema de Gestión de Bases de Datos (SGBD) (CB), un máster (CB1) y un esclavo (CB2); y uno para probar el servicio de Owncloud (CC).

Cabe destacar que tenemos cinco puertos públicos, de 14000 a 14004, y exactamente los mismos en el anfitrión hacia los contenedores, pero dentro de los contenedores podemos usar los típicos (80, 443, 22, etc.).

Contenedor B (Servicio SGBD)

Para el SGBD montaremos un servidor MySQL maestro (CB1) y otro esclavo (CB2). De esta manera dispondremos de un clúster de replicación maestro/esclavo de dos nodos en funcionamiento, siempre para una base de datos (BD) dada.

Primero creamos los directorios de persistencia para los contenedores, Docker nos permite guardar las configuraciones y datos en el anfitrión, útil si se perdemos el contenedor o queremos crear otro a partir de este.

```
mkdir mysql-persistence-master
```

1 Se sugiere para el contenedor (A) el uso de PHP5, en esta memoria se trabajará con **Perl5**.

```
mkdir mysql-persistence-slave
```

CB1

Creamos el contenedor maestro, con datos de identificación, credenciales de replicación y la creación de la BD con credenciales.

```
docker run -d --name mm.mysql-server \
  -v ~/mysql-persistence-master:/bitnami/mysql \
  -e MYSQL_ROOT_PASSWORD=master.pass.123 \
  -e MYSQL_REPLICATION_MODE=master \
  -e MYSQL_REPLICATION_USER=replmuser \
  -e MYSQL_REPLICATION_PASSWORD=replm.pass.123 \
  -e MYSQL_USER=webu -e MYSQL_PASSWORD=webu.123 \
  -e MYSQL_DATABASE=base1 \
  bitnami/mysql:latest
```

Ahora debemos volcar el archivo schema.sql a la BD creada en el paso anterior, que contiene la creación de tablas y la inserción de registros para la aplicación.

```
docker exec -i mm.mysql-server mysql -u webu -pwebu.123 -S
/opt/bitnami/mysql/tmp/mysql.sock base1 <./schema.sql
```

CB2

Una vez creado el maestro, procedemos a crear el contenedor esclavo de MySQL, con datos de identificación, credenciales de replicación, referencia al maestro y la copia de la BD con credenciales.

```
docker run -d --name mm.mysql-slave --link mm.mysql-server:master \
  -v ~/mysql-persistence-slave:/bitnami/mysql \
  -e MYSQL_ROOT_PASSWORD=slave.pass.123 \
  -e MYSQL_REPLICATION_MODE=slave \
  -e MYSQL_REPLICATION_USER=replsuser \
  -e MYSQL_REPLICATION_PASSWORD=repls.pass.123 \
  -e MYSQL_MASTER_HOST=master \
  -e MYSQL_MASTER_ROOT_PASSWORD=replm.pass.123 \
  -e MYSQL_USER=webu -e MYSQL_PASSWORD=webu.123 \
  -e MYSQL_DATABASE=base1 \
  bitnami/mysql:latest
```

De esta manera ya tenemos creado el esquema esclavo para maestro/esclavo de MySQL.

Contenedor A (Servicio Web)

Para el servicio Web utilizaremos Docker con Apache, SSL y Perl5 configurados en el mismo (CA1). Luego, lo replicaremos (CA2), para disponer de los dos contenedores sobre otro, un balanceador de carga (CA0).

CA1

Primero clonaremos la aplicación, disponible en Github[2], para hacerla persistente con el contenedor.

```
git clone https://github.com/mmaguero/Dancer-Plugin-SimpleCRUD appWeb
```

Ahora crearemos el directorio para utilizar la persistencia de configuración de Apache que provee Docker.

```
mkdir apache-persistence
```

Copiamos el Dockerfile creado previamente, con la imagen a utilizar y todo lo necesario, en este caso: Perl5, PerlDancer, Dancer::Simple::CRUD y ya hacemos EXPOSE sobre los puertos necesarios. Montamos la imagen:

```
docker build -t mmaguero/dancer-plugin-simplecrud .
```

Creamos el contenedor a partir de la imagen creada, publicamos el puerto 80 al 14000 de nuestro anfitrión, al igual que el 443 al 14001, lo vinculamos al contenedor CB1 y le damos un nombre de host.

```
docker run -d --name mm.apache -p 14000:80 -p 14001:443 \
 -v ~/appWeb/example:/home/bitnami -v ~/apache-persistences:/bitnami/apache \
 -h "hadoop.ugr.es" --link mm.mysql-server:mysqldb \
 mmaguero/dancer-plugin-simplecrud
```

Ahora debemos preparar a Apache para que pueda despachar peticiones SSL y que pueda ser fronted de la app dancer. Apache utiliza mod_proxy, y es necesario contar con .cert y .key del dominio para SSL (más información en [2]).

```
docker exec -i -t mm.apache vim /opt/bitnami/apache/conf/httpd.conf
    #/Listen 80 to 443
    #/localhost:80 #hadoop.ugr.es
    #/<Virtualhost> #add proxy_reverse
    <<COMMENT
    NameVirtualHost *:80
    <VirtualHost *:80> # 443 for SSL
```

2 Ir a https://github.com/mmaguero/Dancer-Plugin-SimpleCRUD

```
 ServerName hadoop.ugr.es
 #enabled for SSL
 #SSLEngine on
 #SSLCertificateFile /bitnami/apache/conf/bitnami/certs/server.crt
 #SSLCertificateKeyFile /bitnami/apache/conf/bitnami/certs/server.key
 <Proxy *>
  Order deny,allow
  Allow from all
 </Proxy>
 RewriteEngine on
#dancer app
 ProxyPass        / http://hadoop.ugr.es:14000/
 ProxyPassReverse / http://hadoop.ugr.es:14000/
</VirtualHost>
<IfModule mod_proxy.c>
      ProxyRequests Off
      <Proxy *>
           AddDefaultCharset off
            Order deny,allow
            Allow from all
      </Proxy>
      ProxyVia On
</IfModule>
COMMENT
```

Guardamos los cambios, salimos y reiniciamos para poder impactar las modificaciones.

```
docker restart mm.apache
```

Ya tenemos la nueva configuración, ahora debemos configurar (archivo YAML) la app dancer: debemos indicar el host de la BD, usuario, contraseña y otras cosas más, tales como nombre de la aplicación, etc.

```
docker exec -i -t mm.apache vim /home/bitnami/mysql/config.yml
```

Corremos la aplicación en segundo plano, como nota, cada vez que reiniciemos el contenedor debemos levantar este servicio.

```
docker exec -i -t mm.apache perl /home/bitnami/mysql/simplecrud-example.pl --port 14003 --daemon
```

En el navegador ya podemos escribir http://anfitrion:14000/ o https://anfitrion:14001/ y acceder a la app dancer.

CA2

Ahora que tenemos corriendo el CA1, pasamos a replicarlo, como el balanceador trabajará sobre el puerto 80, habilitamos sólo este puerto en este caso. Adicionalmente podemos enlazarlo al CA1.

```
docker run -d --name mm.apacherepl \
 -h "hadoop.ugr.es" -p 14002:80 \
 -v ~/appWeb/example:/home/bitnami -v ~/apache-persistences:/bitnami/apache \
 --link mm.mysql-server:mysqldb --link mm.apache:apache-master \
  mmaguero/dancer-plugin-simplecrud
```

Gracias a utilizar el mismo directorio de persistencia, tenemos los mismos datos de configuración de la app y de Apache del CA1 (por lo tanto levantamos la app en el mismo puerto)

```
docker exec -i -t mm.apacherepl perl /home/bitnami/mysql/simplecrud-example.pl --port 14003 –daemon
```

En el navegador ingresamos http://anfitrion:14002/ y accedemos a la app dancer.

CA0

Ya con los dos contenedores creados, haría falta un balanceador de carga; de esta manera tenemos a la app corriendo como un servicio, redireccionada a Apache en cada contenedor, y a su vez este balanceador redirecionará a uno u otro contenedor:

CA1>>SimpleCRUDDancer (14003)→Apache (80,443) → IP Pública (1400/1)

=> CA0>>Nginx (1403) → IP Pública (1403)

CA2>> SimpleCRUDDancer (14003)→Apache (80) → IP Pública (14002)

Creamos el balanceador de carga.

```
docker run -d --name mm.nginxld -p 14003:14003 nginx
```

Ahora ingresamos al contenedor, sobre-escribimos el archivo de configuración para poder hacer el balanceo de los contenedores CA1 y CA2 desde el CA0.

```
docker exec -i -t mm.nginxld /bin/bash
        echo "upstream servers {
        server hadoop.ugr.es:14000; #CA1
        server hadoop.ugr.es:14002; #CA2
```

```
}
server {
listen 14003;
location / {
proxy_pass http://servers;
}
}" > /etc/nginx/conf.d/default.conf
```

Salimos y reiniciamos el contenedor.

```
docker restart mm.nginxld
```

Ahora podemos ingresar en el navegador a la dirección http://anfitrion:14003/ y vemos que nos redirigirá a una u otro contenedor, podemos hacer *stop* de alguno para comprobarlo.

Aplicación web

Objetivo

El objetivo de la aplicación web es de un administrador de usuarios, dar de alta usuarios, editarlos o borrarlos, vincularlos a una empresa y poder consultarlos, escribiendo notas sobre los mismos. Es un CRUD muy simple accesible desde cualquier dispositivo con conexión a Internet.

Funcionalidad

Las funcionalidades de esta aplicación son las esperadas de un CRUD: Crear, Leer, Actualizar y Borrar: en este contexto de personas o usuarios. Tiene una pantalla de login simple (si se quiere probar, en el archivo config.yml de la aplicación se pueden ver las credenciales) que ingresa a un listado de usuarios dados de alta, a la derecha de la tabla se pueden editarlos o borrarlos, también esta pantalla permite filtrar los datos para todas las columnas, se puede ordenar clicando las cabeceras de columnas; en le menú se puede cerrar sesión o agregar un nuevo usuario, volver a home.

Arquitectura software

La aplicación es un CRUD simple realizado sobre Perl, el framework Dancer y el plugin SimpleCRUD. Utiliza MVC, algunos patrones de diseño y puede funcionar con una arquitectura cliente/servidor y microservicios (la MV1 despliega el servicio web y MV2 el de datos). Al usar Bootstrap 4 es *responsiva*, pudiéndose acceder desde cualquier plataforma que cuenta con un navegador e Internet.

Base de datos

El SGBD utilizado por la aplicación es MySQL, motor relacional usado en ambientes web ampliamente, en su versión 5.7. Se ha montado una base de datos con dos tablas relacionadas.

Tablas

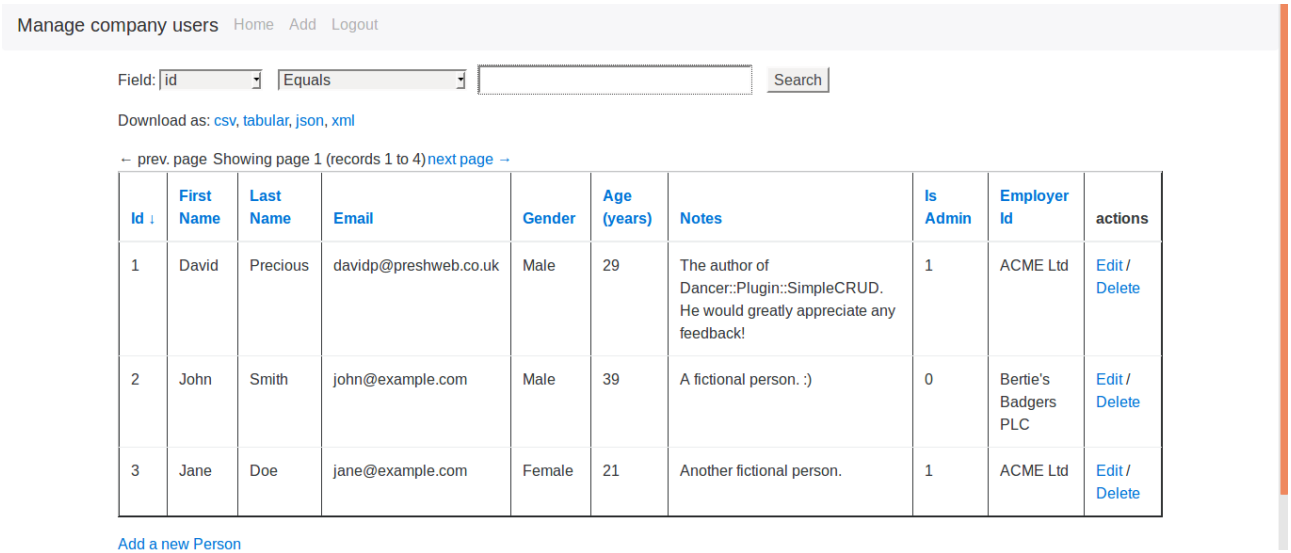
Las tablas creadas son people y employer. La primera es una tabla con datos personales básicos: nombre, edad, sexo; correo, comentarios, y su rol de usuario, además de a que empresa pertenece. Justamente la segunda tabla describe esa relación.

Manuales

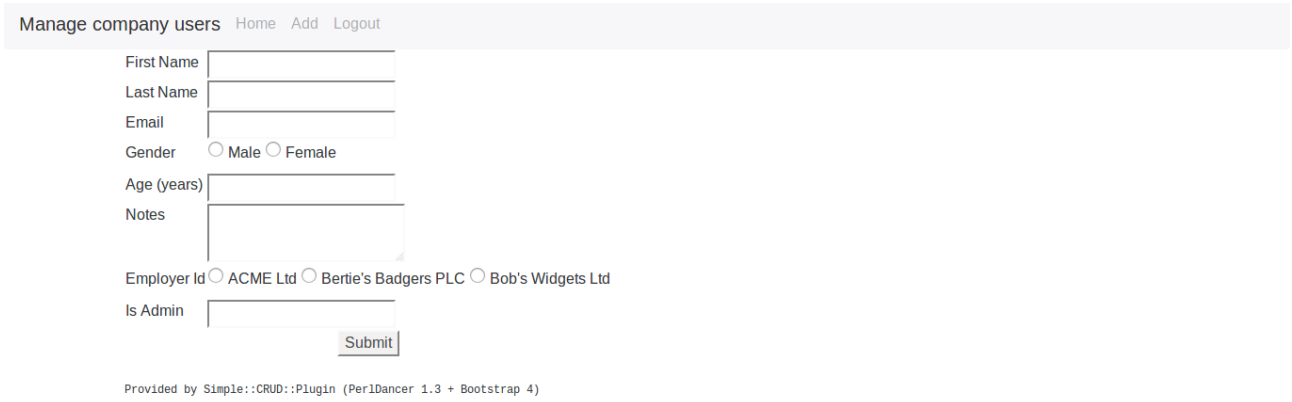
Uso de la aplicación web

Ir a la página de login en http://docker.ugr.es:15061, se pueden usar las credenciales editor@editor para ingresar.

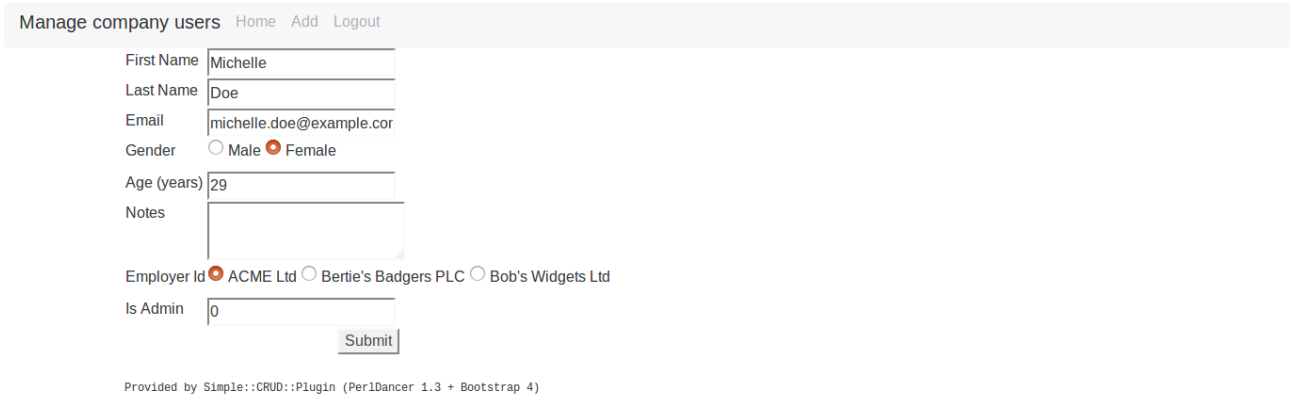
Al ingresar se observa el listado de usuarios con las distintas opciones habilitadas, como un dashboard simple de administración de usuarios.

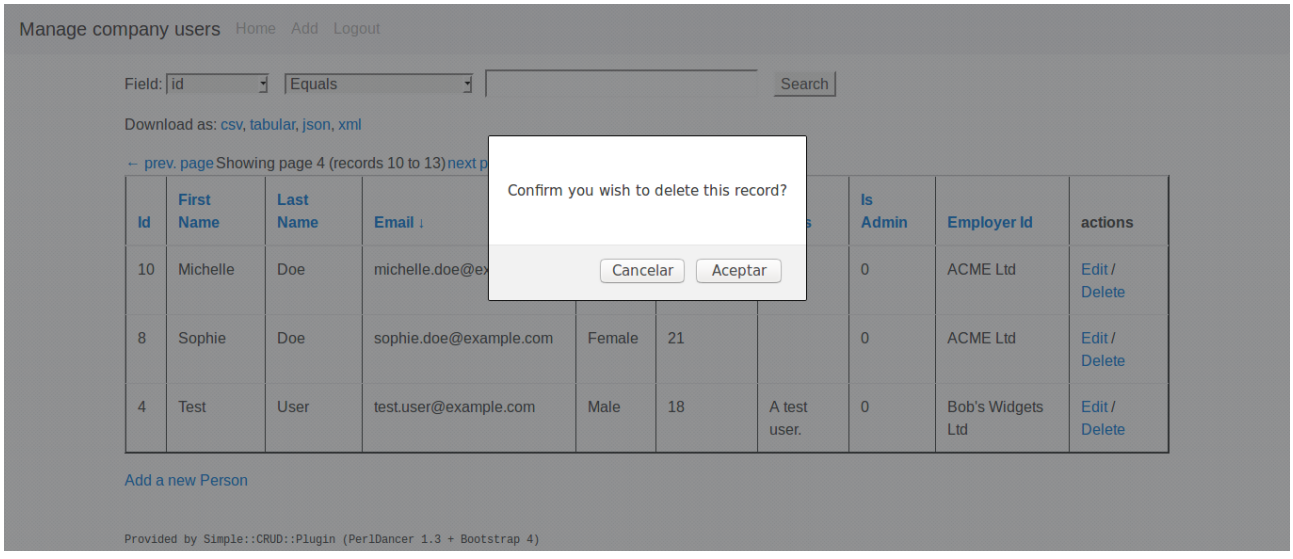


En el menú se puede agregar usuarios nuevos, volver a home o salir de la aplicación.

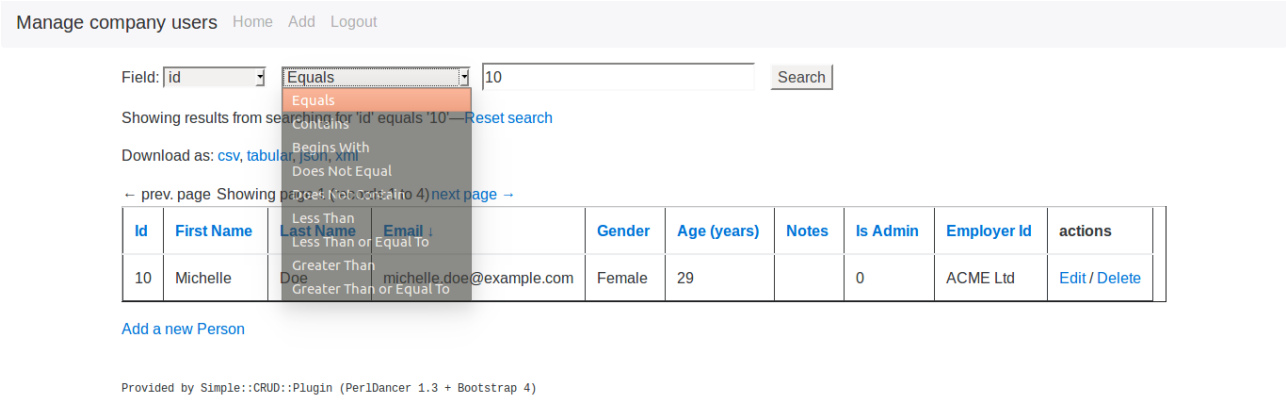


A la derecha de la tabla se puede editar o borrar usuarios existentes. Los títulos o cabeceras permiten ordenar los valores de cada columna.

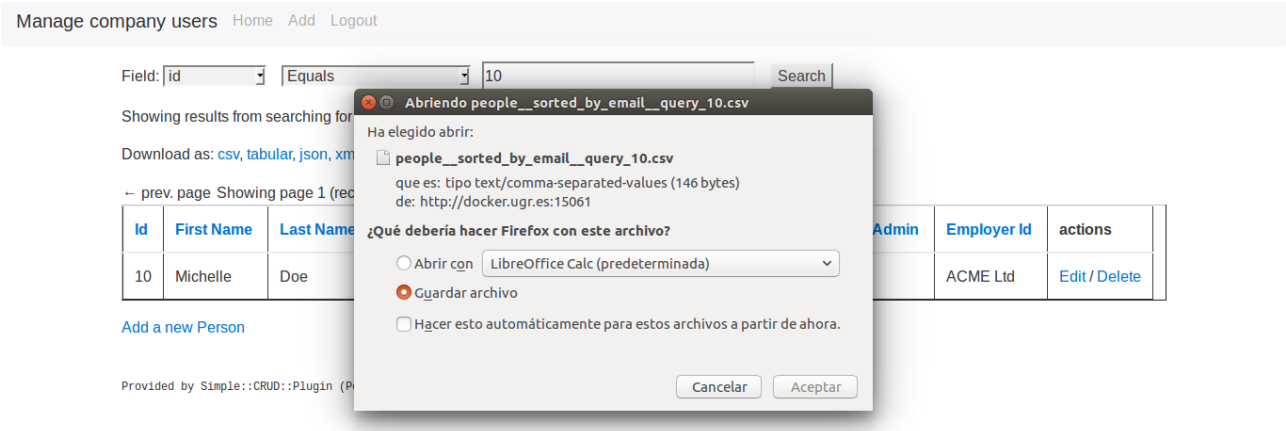




El listado también puede filtrarse seleccionando la columna y el tipo de filtro.



Además es posible descargar el listado actual en CSV, txt o JSON.



Análisis de la funcionalidad de los contenedores replicados

Como se pudo comprobar al replicar los contenedores, se logra ganar la robustez del sistema: tener siempre disponibles los servicios, tanto de SGBD como la Web. Es una actividad que Docker la tiene pensada y preparada, ofrece muchas ventajas en su uso y facilidades que en la misma creación del contenedor se logra concretar, y además ofrece otras ventajas, como los directorios de persistencia, la comunicación de los contenedores, entre otros.

Por lo tanto, una vez directamente comprobado, según lo visto más arriba, la petición de recursos no se verá afectada bajo este esquema, aunque es a una escala pequeña, Docker demuestra su robustez para esta consigna sin mucho esfuerzo. Entonces, si por cualquier cuestión uno de los servicios cayera, no se verán afectados los servicios ofrecidos, siendo todo transparente para usuarios. Si fuera a una escala mayor, aun seguiría consolidada la información, tanto del SGBD, como de la Web, puesto que se pueden agregar varios contendedores para seguir manteniendo el servicio ante la demanda.

Proceso de instalación, configuración y despliegue de Owncloud en Docker

Configuraremos un servicio de almacenamiento en al nube, denominado Owncloud, utilizaremos Docker para los contenedores, y MariaBD para el SGBD.

Contendor C

Primero crearemos el contenedor (CC1) con MariaBD, para luego enlazarlo al contendor de Owncloud (CC2). Antes que nada creamos las carpetas de persistencia de Docker:

```
mkdir mariadb-persistence

mkdir owncloud-data && mkdir owncloud-config
```

CC1

La creación de un contenedor con MariaDB es simple, en este caso decimos que el ROOT no tendrá contraseña (al ser un entorno de desarrollo no hay ningún riesgo).

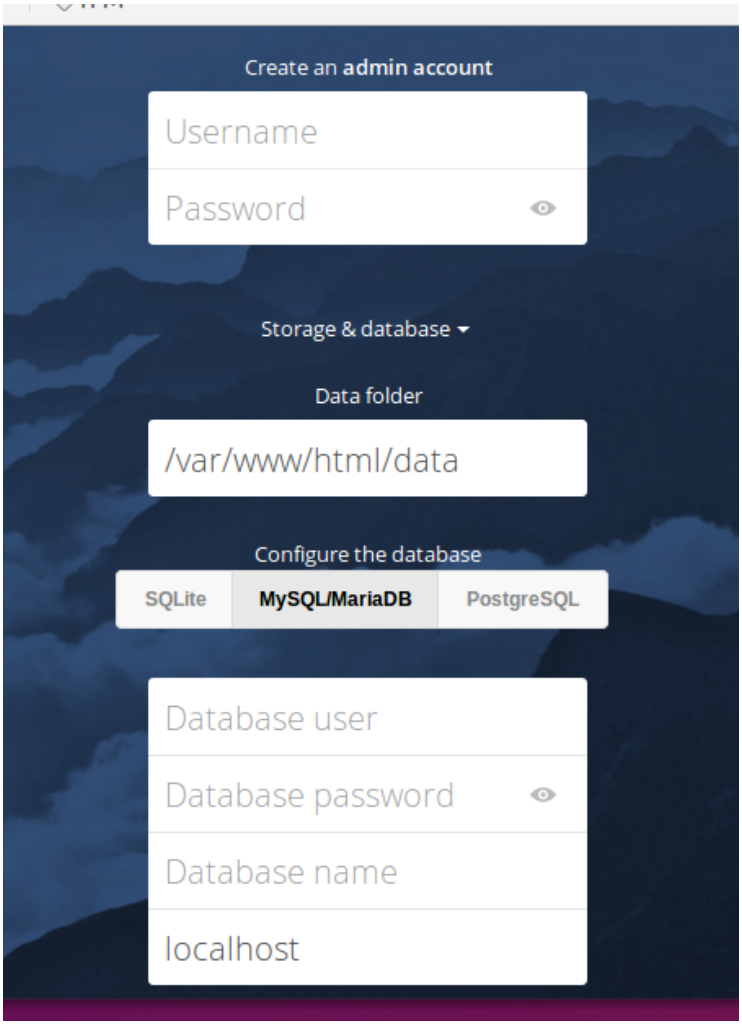
```
docker run -d --name mm.mariadb -e ALLOW_EMPTY_PASSWORD=yes \
 --volume ~/mariadb-persistence:/bitnami/mariadb bitnami/mariadb
```

CC2

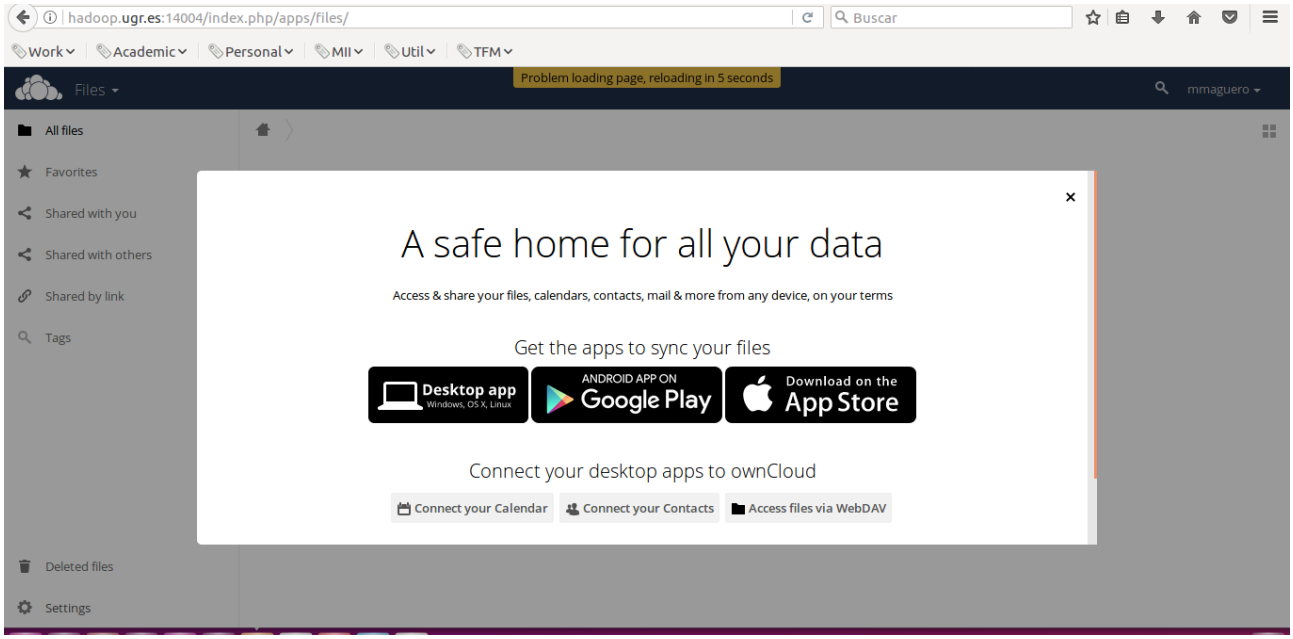
Ahora creamos el contenedor para la aplicación Web de Owncloud y lo enlazamos al contenedor CC1 para poder configurarlo luego, utilizamos el puerto 80 vinculado al 14004 del anfitrion, el mismo para la IP Pública (http://anfitrion:14004).

```
docker run -d -it --name mm.owncloud -p 14004:80 \
 --volume ~/owncloud-data:/var/www/html/data \
 --volume ~/owncloud-config:/var/www/html/config \
 -e OWNCLOUD_HOST=hadoop.ugr.es --link mm.mariadb:owncloud-db \
 owncloud
```

Ahora accedemos desde el navegador y configuramos el servicio de Owncloud con los valores siguientes: yourAdmin@yourPass para crear un usuario y contraseña para ingresar a la aplicación, las credenciales de MariaDB root@empty, asignamos un nombre a la BD, y en el host escribimos el nombre del contenedor CC1 mm.mariadb.



Una vez configurado, ya podemos ingresar en el servicio desde nuestros contenedores.



Bibliografía consultada

General

[a] http://codehero.co/como-construir-imagenes-usando-dockerfiles/

[b] http://blog.cloud66.com/how-to-get-code-into-a-docker-container/

[c] http://www.josedomingo.org/pledin/2016/02/ejemplos-de-ficheros-dockerfile-creando-imagenes-docker/

[d] https://github.com/mmaguero/CC17/blob/master/starting_docker.md

[e] http://www.linuxforums.org/forum/debian-linux/136720-solved-apache2-mod_proxy-mod_rewrite.html

[f] http://nginx.org/en/docs/http/load_balancing.html

CA

[1] https://hub.docker.com/r/bitnami/apache/

[2] https://picodotdev.github.io/blog-bitix/2016/06/como-redirigir-peticiones-de-http-a-https-en-nginx-apache-tomcat-jetty-y-wildfly/

[3] https://github.com/picodotdev/blog-ejemplos/tree/master/RedirigirHTTPaHTTPS

[4] http://www.tecmint.com/sync-two-apache-websites-using-rsync/

CB

[8] https://hub.docker.com/r/bitnami/mysql/

[9] http://stackoverflow.com/questions/41945292/executing-mysql-script-docker

CC

[10] https://hub.docker.com/r/bitnami/owncloud/
[11] http://blog.securem.eu/serverside/2015/08/25/setting-up-owncloud-server-in-a-docker-container/

Anexos

Disponibles en https://github.com/mmaguero

Adjuntos

- /appweb
  - appWeb.sh
    - Scripts para crear contenedores principales, replicas y balanceadores.

    - /web
        - incluye las vistas y layouts, configuraciones de aplicación como conexión a BD, usuarios, etc., archivo perl que ejecuta la aplicación.
        - Dockerfile creado para el contenedor
    - /sgbd
        - incluye la creación de la base de datos, tablas e inserciones.
- /owncloud
    - owncloud.sh
        - Script para crear los contenedores de owncloud

Dockerfile

FROM bitnami/apache:latest

MAINTAINER Marvin Agüero "marvin-aguero@hotmail.com"

RUN apt-get -y update && apt-get -y upgrade

*RUN apt-get install -y apt-utils && apt-get install -y perl *

build-essential libdancer-perl libdbd-mysql-perl cpanminus vim

RUN cpanm Dancer::Plugin::SimpleCRUD && cpanm Dancer::Plugin::Auth::Extensible

RUN apt-get autoclean && apt-get clean && apt-get autoremove

EXPOSE 14000 80 443