
UNIVERSIDAD DE GRANADA

MASTER PROFESIONAL EN INGENIERÍA INFORMÁTICA

PRÁCTICA 2

Docker

Autor:

Manuel Jesús García Manday
(nickter@correo.ugr.es)

Master en Ingeniería Informática

18 de abril de 2017

Índice

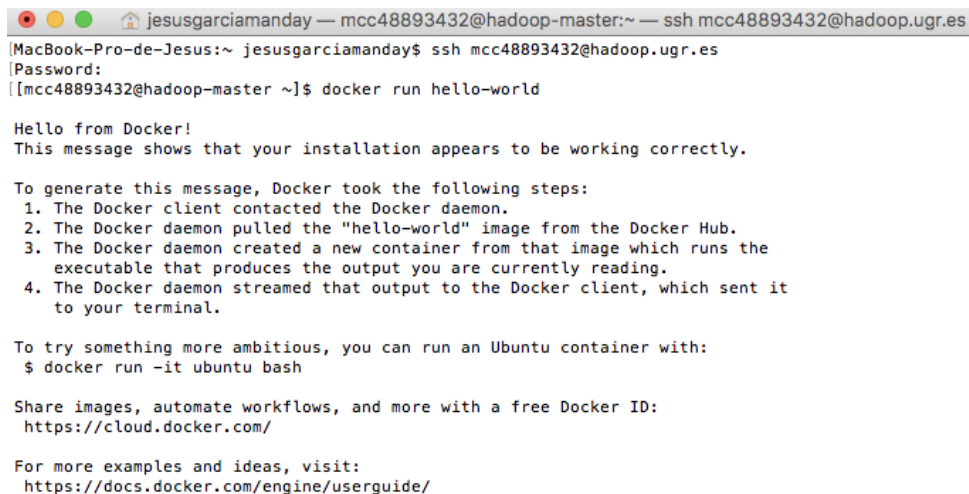
1. Objetivo.	3
2. Descripción general del despliegue de un contenedor en docker.	3
3. Configuración del servicio Web en docker.	6
4. Configuración del servicio de SGBD en docker.	6
5. Descripción de la aplicación web. Objetivo, funcionalidad, arquitectura software, base de datos, tablas. Puedes utilizar la misma que en la Práctica 1 (IaaS).	7
6. Análisis de la funcionalidad de los contenedores replicados.	12
7. Configuración de OwnCloud en docker.	12

1. Objetivo.

El objetivo de esta práctica es familiarizarse con el uso de una plataforma PaaS y desarrollar habilidades de despliegue de contenedores y configurar aplicaciones sencillas en los mismos.

2. Descripción general del despliegue de un contenedor en docker.

Lo primero que se debe realizar es una conexión remota hacia el servidor `hadoop.ugr.es` y comprobar que el acceso a docker es correcto.



```

jesusgarciamanday — mcc48893432@hadoop-master:~ — ssh mcc48893432@hadoop.ugr.es
[MacBook-Pro-de-Jesus:~ jesusgarciamanday$ ssh mcc48893432@hadoop.ugr.es
[Password:
[mcc48893432@hadoop-master ~]$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

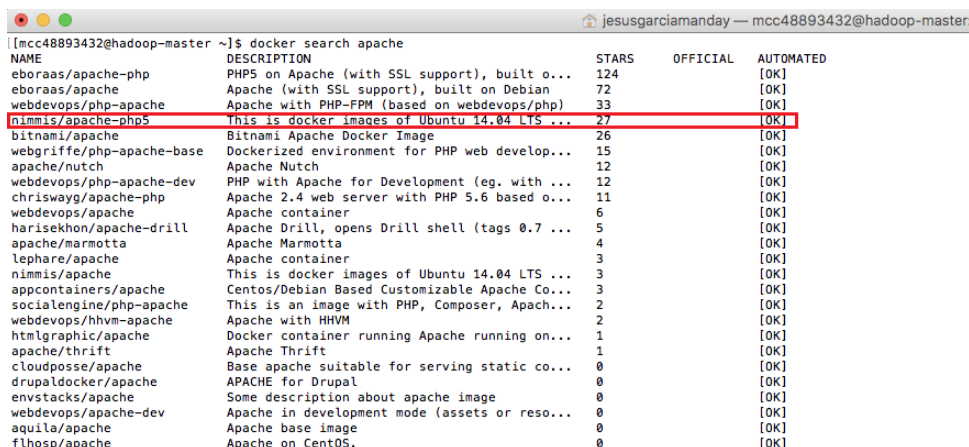
Share images, automate workflows, and more with a free Docker ID:
https://cloud.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/

```

Figura 1: Conexión ssh y prueba del sistema docker.

Una vez que hemos comprobado el funcionamiento del sistema docker, lo siguiente será buscar si ya existe alguna imagen en el sistema que cumpla los requisitos que buscamos, es decir, que tenga los servicios de **APACHE**, **SSL** y **PHP5** instalados como se solicita para el primer contenedor. Para ello vamos a utilizar el comando `docker search` y a continuación el nombre de los servicios que queremos que traiga la imagen incorporados como se muestra en la siguiente figura.



```

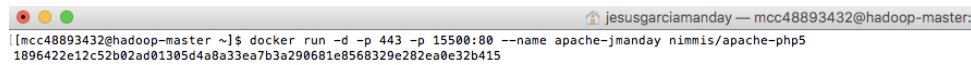
jesusgarciamanday — mcc48893432@hadoop-master:~
[mcc48893432@hadoop-master ~]$ docker search apache

```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
eboraas/apache-php	PHP5 on Apache (with SSL support), built o...	124		[OK]
eboraas/apache	Apache (with SSL support), built on Debian	72		[OK]
webdevops/php-apache	Apache with PHP-FPM (based on webdevops/php)	33		[OK]
nimmis/apache-php5	This is docker images of Ubuntu 14.04 LTS ...	27		[OK]
bitnami/apache	Bitnami Apache Docker Image	26		[OK]
webgriffe/php-apache-base	Dockerized environment for PHP web develop...	15		[OK]
apache/nutch	Apache Nutch	12		[OK]
webdevops/php-apache-dev	PHP with Apache for Development (eg. with ...	12		[OK]
chriswayg/apache-php	Apache 2.4 web server with PHP 5.6 based o...	11		[OK]
webdevops/apache	Apache container	6		[OK]
harisekhon/apache-drill	Apache Drill, opens Drill shell (tags 0.7 ...	5		[OK]
apache/marmotta	Apache Marmotta	4		[OK]
lephare/apache	Apache container	3		[OK]
nimmis/apache	This is docker images of Ubuntu 14.04 LTS ...	3		[OK]
appcontainers/apache	Centos/Debian Based Customizable Apache Co...	3		[OK]
socialengine/php-apache	This is an image with PHP, Composer, Apach...	2		[OK]
webdevops/hhvm-apache	Apache with HHVM	2		[OK]
htmlgraphic/apache	Docker container running Apache running on...	1		[OK]
apache/thrift	Apache Thrift	1		[OK]
cloudposse/apache	Base apache suitable for serving static co...	0		[OK]
drupaldocker/apache	APACHE for Drupal	0		[OK]
envstacks/apache	Some description about apache image	0		[OK]
webdevops/apache-dev	Apache in development mode (assets or reso...	0		[OK]
aquila/apache	Apache base image	0		[OK]
flhosp/apache	Apache on CentOS.	0		[OK]

Figura 2: Buscando imagenes en docker.

Como se puede apreciar en la anterior fotografía, la imagen **nimmis/apache-php5** cumple con los requisitos necesarios para crear el primer contenedor, por lo que lo siguiente que haremos será crearlo con el comando **docker run**. Cabe recordar que el puerto SSL por defecto es el 443, y que dicho servicio se instala con **APACHE**, por lo que para redirigirlo al puerto asignado en el host anfitrión hay que indicarlo mediante el flag **p** como se muestra en la imagen de a continuación.



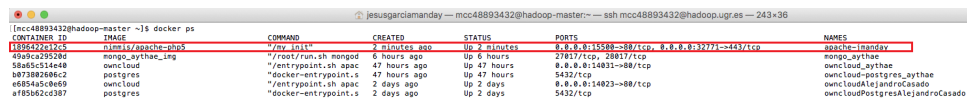
```

[mcc48893432@hadoop-master ~]$ docker run -d -p 443 -p 15500:80 --name apache-jmanday nimmis/apache-php5
1896422e12c52b02ad01305d4a8a33ea7b3a290681e8568329e282ea0e32b415

```

Figura 3: Creando el primer contenedor.

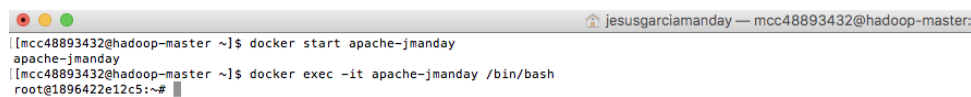
Una vez creado el contenedor pasamos a comprobar que se ha realizado correctamente ejecutando el comando **docker ps**, el cual nos lista los contenedores creados sin errores en el sistema con atributos como el nombre, el redireccionamiento de puertos entre el host anfitrión y el contenedor, etc. En la siguiente imagen se refleja.



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1896422e12c5	nimmis/apache-php5	"/bin/bash"	2 minutes ago	Up 2 minutes	0.0.0.0:15500->80/tcp, 0.0.0.0:15500->443/tcp	apache-jmanday
49a9ca255280	mongo_aythae_img	"/root/run.sh mongod"	6 hours ago	Up 6 hours	27017/tcp, 28017/tcp	mongo_aythae
58ad6c334e40	owncloud	"/entrypoint.sh apac"	47 hours ago	Up 47 hours	0.0.0.0:14023->80/tcp	owncloud_aythae
bb73882806c2	postgres	"docker-entrypoint.s"	47 hours ago	Up 47 hours	5432/tcp	owncloud-postgres_aythae
c6894d50e0d9	owncloud	"/entrypoint.sh apac"	2 days ago	Up 2 days	0.0.0.0:14023->80/tcp	owncloudAlejandroCasado
a185b62cc387	postgres	"docker-entrypoint.s"	2 days ago	Up 2 days	5432/tcp	owncloudPostgresAlejandroCasado

Figura 4: Comprobando que se ha creado el primer contenedor.

Viendo que el contenedor se ha creado correctamente, ahora vamos a lanzarlo y acceder a su terminal para poder comprobar que tiene los servicios de **apache** y **php** instalados. Para realizar este paso vamos a emplear el comando **docker run** a través del cual se despliega un contenedor, pudiéndole indicar un comando o acción a realizar por este.

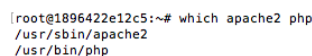


```

[mcc48893432@hadoop-master ~]$ docker start apache-jmanday
apache-jmanday
[mcc48893432@hadoop-master ~]$ docker exec -it apache-jmanday /bin/bash
root@1896422e12c5:~#

```

Figura 5: Lanzando el primer contenedor.



```

[root@1896422e12c5:~# which apache2 php
/usr/sbin/apache2
/usr/bin/php

```

Figura 6: Comprobando que apache y php están instalados.

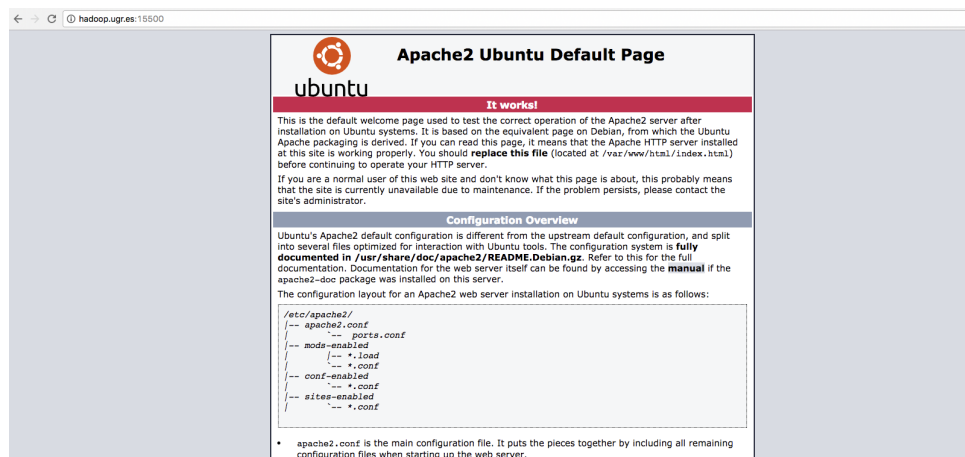


Figura 7: Comprobando apache.

Para crear el segundo contenedor se van a seguir los mismos pasos que para el primero, por lo que lo primero que se realizará será buscar si existe alguna imagen en **Docker** con **MySQL** y si es así crearlo, lanzarlo y comprobar que el servicio de mysql existe en el contenedor.

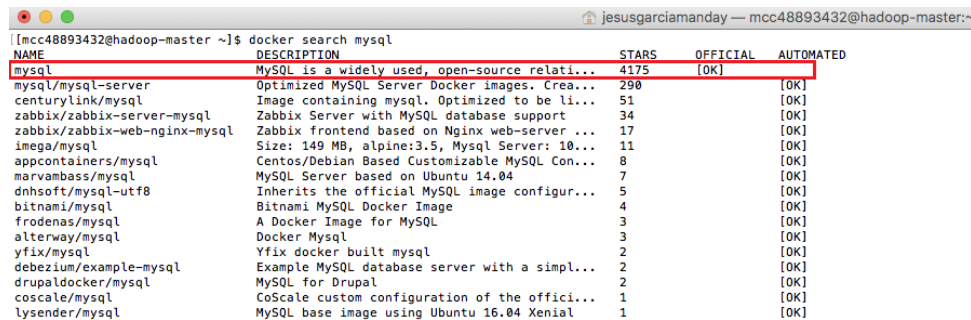


Figura 8: Buscando una imagen para crear el segundo contenedor.

Con las imagenes de **MySQL** listadas, seleccionamos la primera para crear el segundo contenedor.

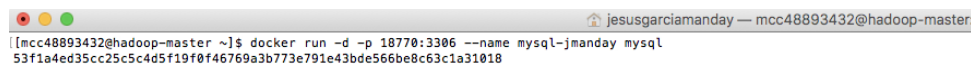


Figura 9: Creando el segundo contenedor.

Comprobamos que se ha creado correctamente, y una vez verificado lo lanzamos y buscamos el servicio **MySQL**.

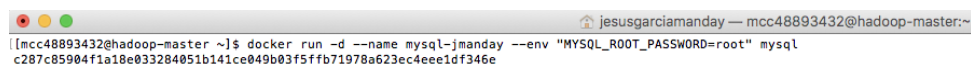


Figura 10: Lanzando el segundo contenedor.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
c287c85904f1	mysql	"docker-entrypoint.sh mysq..."	About a minute ago	Up About a minute	3306/tcp	mysql-jmanday
1896422412c5	nimmis/apache-php5	"my_init"	59 minutes ago	Up 59 minutes	0.0.0.0:15508->88/tcp, 0.0.0.0:32771->443/tcp	apache-jmanday
439da295286	manga/aythae_img	"rootfs.sh mongod"	7 hours ago	Up 7 hours	27037/tcp, 28027/tcp	manga_aythae
58a65c514e40	owncloud	"entrypoint.sh apac..."	2 days ago	Up 2 days	0.0.0.0:14831->88/tcp	owncloud_aythae
b073882606c2	postgres	"docker-entrypoint.sh..."	2 days ago	Up 2 days	5432/tcp	owncloud-postgres_aythae
e68545c5e6d9	owncloud	"entrypoint.sh apac..."	2 days ago	Up 2 days	0.0.0.0:14823->88/tcp	owncloudajandrocasado
af85862cd387	postgres	"docker-entrypoint.sh..."	2 days ago	Up 2 days	5432/tcp	owncloudpostgresajandrocasado
ef7589702086	nginx_aythae_img	"nginx -g 'daemon off..."	2 days ago	Up 2 days	443/tcp, 0.0.0.0:14805->88/tcp	nginx_aythae
78c213863c9f	python_aythae_img	"supervisord -n..."	2 days ago	Up 2 days	8888/tcp	python_aythae
8a77769f2183	eborasi/apache-php	"usr/sbin/apache2ct..."	2 days ago	Up 2 days	443/tcp, 0.0.0.0:14805->88/tcp	apacheCesar
28a15f4f137	jdeathie/centos-ssh-mysql	"usr/bin/supervisor..."	2 days ago	Up 2 days	22/tcp, 0.0.0.0:14808->3306/tcp	mysqlCesar

Figura 11: Comprobando que el segundo contenedor se ha creado.

```
[mcc48893432@hadoop-master ~]$ docker exec -it mysql-jmanday /bin/bash
root@c287c85904f1:/# which mysql
/usr/bin/mysql
```

Figura 12: Comprobando que MySQL está instalado en el contenedor.

3. Configuración del servicio Web en docker.

Se ha tomado la configuración por defecto para el servicio Web instalado en el primer contenedor y para comprobar que funciona correctamente le lanzamos una petición a su página de inicio como se muestra en la siguiente figura.

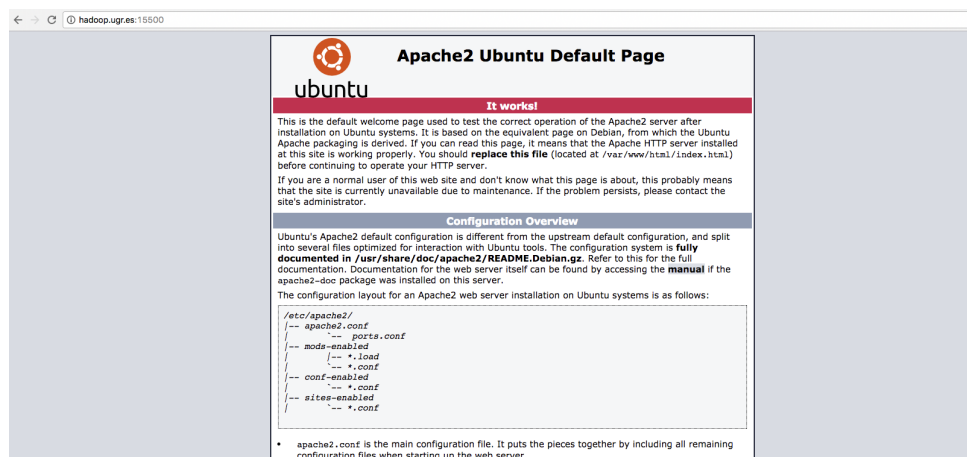


Figura 13: Configuración de Apache en docker.

4. Configuración del servicio de SGBD en docker.

Para este servicio instalado en el segundo contenedor se ha procedido a emplear la configuración por defecto al igual que en el apartado anterior con el servicio Web, por lo que el usuario de **MySQL** que se utilizará para conectarse a dicho servicio será el que viene por defecto (`root`). En la imagen de a continuación se puede ver como la configuración tomada es óptima.

```
jesusgarciamanday — mcc48893432@hadoop-master:~ — ssh mcc48893432@hadoop.ugr.es
[mcc48893432@hadoop-master ~]$ docker exec -it mysql-jmanday /bin/bash
[root@c287c85904f1:/# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Figura 14: Configuración de MySQL en docker.

5. Descripción de la aplicación web. Objetivo, funcionalidad, arquitectura software, base de datos, tablas. Puedes utilizar la misma que en la Práctica 1 (IaaS).

La aplicación que se va a estar ejecutando en el primer contenedor con el servicio Web **Apache** es una aplicación de gestión de usuarios que muestra una lista de los mismos y que permite las operaciones de insertar y eliminar usuarios de la base de datos que se encuentra en el segundo contenedor con el servicio de **MySQL**.

La arquitectura de la aplicación esta basada en microservicios, en este caso son dos los principales software sobre los que se sustenta dicha aplicación (**Apache** y **MySQL**). Cada uno de los servicios mencionados residen en un contenedor diferente, comunicandose de forma remota entre ellos a través de protocolos de internet como http, icmp y tcp entre otros. En la siguiente imagen se puede apreciar la arquitectura en la que está basada la aplicación web.

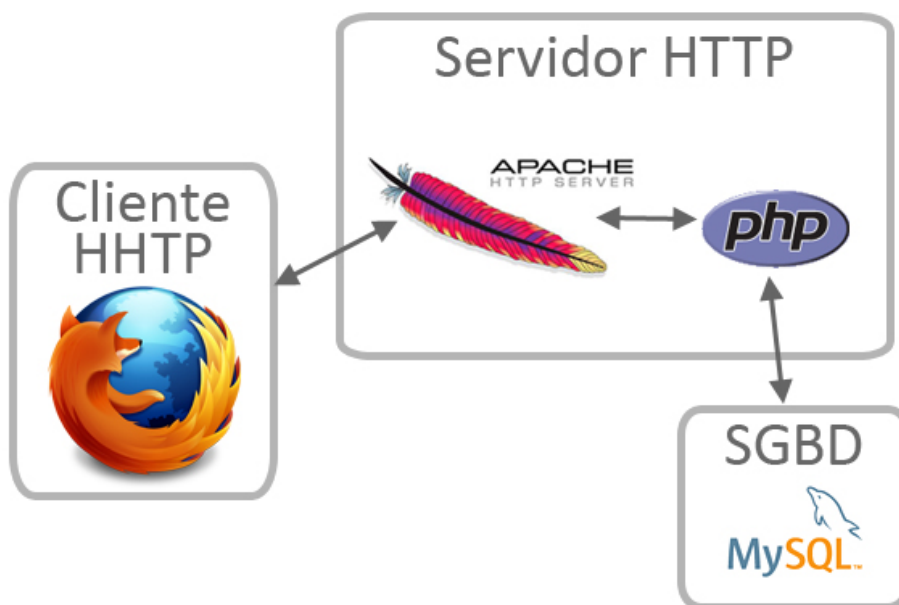
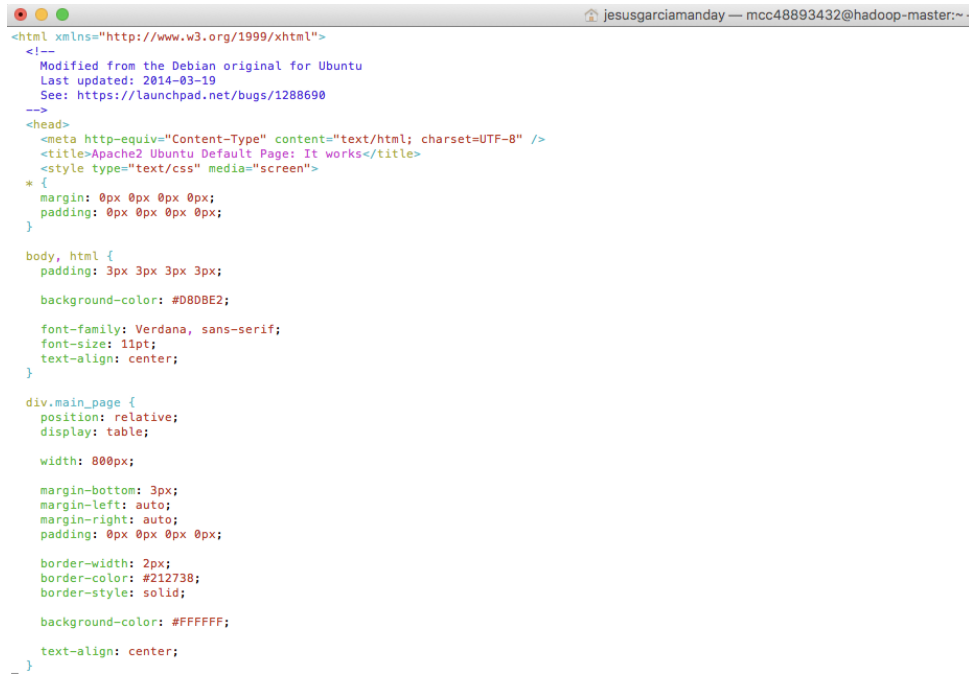


Figura 15: Arquitectura de la aplicación.

En el primer contenedor es donde se aloja el servidor Web de **Apache** y es por tanto donde se han definido los ficheros **php** que manejan las acciones realizadas por el usuario sobre la aplicación. Son tres los ficheros que se han implementado para el servidor los cuales residen en la dirección **/var/www/html** del mismo. El fichero por defecto que se ejecuta al acceder a la página inicial del servidor <http://hadoop.ugr.es:15500/> es el **index.php**, existen dos ficheros más que se encargan de cada una de las dos operaciones permitidas en la aplicación, la insercción de un nuevo usuario a través del fichero **insert_user.php** y eliminar a un usuario existente mediante el fichero **delete_user.php**. En las siguientes imagenes se muestra la estructura y el contenido de dichos ficheros.



```
<html xmlns="http://www.w3.org/1999/xhtml">
<!--
Modified from the Debian original for Ubuntu
Last updated: 2014-03-19
See: https://launchpad.net/bugs/1288690
-->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Apache2 Ubuntu Default Page: It works</title>
<style type="text/css" media="screen">
* {
margin: 0px 0px 0px 0px;
padding: 0px 0px 0px 0px;
}

body, html {
padding: 3px 3px 3px 3px;
background-color: #D8DBE2;

font-family: Verdana, sans-serif;
font-size: 11pt;
text-align: center;
}

div.main_page {
position: relative;
display: table;

width: 800px;

margin-bottom: 3px;
margin-left: auto;
margin-right: auto;
padding: 0px 0px 0px 0px;

border-width: 2px;
border-color: #212738;
border-style: solid;
background-color: #FFFFFF;

text-align: center;
}
_ }
```

Figura 16: Fichero index.php (I).


```

jesusgarciamanday — mcc48893432@hadoop-master:~ — s
<body>
<div class="main_page">
<div class="page_header floating_element">
<span class="floating_element" text-align="center">
Gestion de Usuarios
</span>
</div>
<br><br><br>
<div align="center">

<?php
$servername = "172.17.2.160";
$username = "root";
$password = "root";
$dbname = "Docker";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT ID, nombre, apellidos from Usuarios";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // output data of each row
    echo "<br><h2><em>Lista de Usuarios</em></h2><br>";
    echo "<table border='1' align='center'>";
    echo "<tr>";
    echo "<td><b>id</b></td>";
    echo "<td><b>Nombre</b></td>";
    echo "<td><b>Apellidos</b></td>";
    echo "</tr>";
    while($row = $result->fetch_assoc()) {
        echo "<tr>";
        echo "<td>" . $row["ID"] . "</td>";
        echo "<td>" . $row["nombre"] . "</td>";
        echo "<td>" . $row["apellidos"] . "</td>";
        echo "<td><form action='delete_user.php' method='POST'>";
        echo "<input class='form-btn' name='borrar' type='submit' value='Borrar' />";
        echo "<input type='hidden' name='id' value='" . $row["ID"] . "' />";
        echo "</form></td>";
        echo "</tr>";
    }
    echo "</table>";
} else {
    echo "0 results";
}

$conn->close();
?>

```

Figura 17: Fichero index.php (II).

```

jesusgarciamanday — mcc48893432@hadoop-master:~ — ssh mcc48893432@hadoop.ugr.es
<body>
<div class="main_page">
<div class="page_header floating_element">
<span class="floating_element" text-align="center">
Gestion de Usuarios
</span>
</div>
<br><br><br>
<div align="center">

<?php
$servername = "172.17.2.190";
$username = "root";
$password = "root";
$dbname = "Docker";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT ID, nombre, apellidos from Usuarios";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // output data of each row
    echo "<br><h2><em>Lista de Usuarios</em></h2><br>";
    echo "<table border='1' align='center'>";
    echo "<tr>";
    echo "<td><b>id</b></td>";
    echo "<td><b>Nombre</b></td>";
    echo "<td><b>Apellidos</b></td>";
    echo "</tr>";
    while($row = $result->fetch_assoc()) {
        echo "<tr>";
        echo "<td>" . $row["ID"] . "</td>";
        echo "<td>" . $row["nombre"] . "</td>";
        echo "<td>" . $row["apellidos"] . "</td>";
        echo "<td><form action='delete_user.php' method='POST'>";
        echo "<input class='form-btn' name='borrar' type='submit' value='Borrar' />";
        echo "<input type='hidden' name='id' value='" . $row["ID"] . "' />";
        echo "</form></td>";
        echo "</tr>";
    }
    echo "</table>";
} else {
    echo "0 results";
}

$conn->close();
?>

```

Figura 18: Fichero index.php (III).



```
<?php
$servername = "172.17.2.168";
$username = "root";
$password = "root";
$dbname = "Docker";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$subs_name = utf8_decode($_POST['nombre']);
$subs_apellidos = utf8_decode($_POST['apellidos']);
$subs_edad = utf8_decode($_POST['edad']);

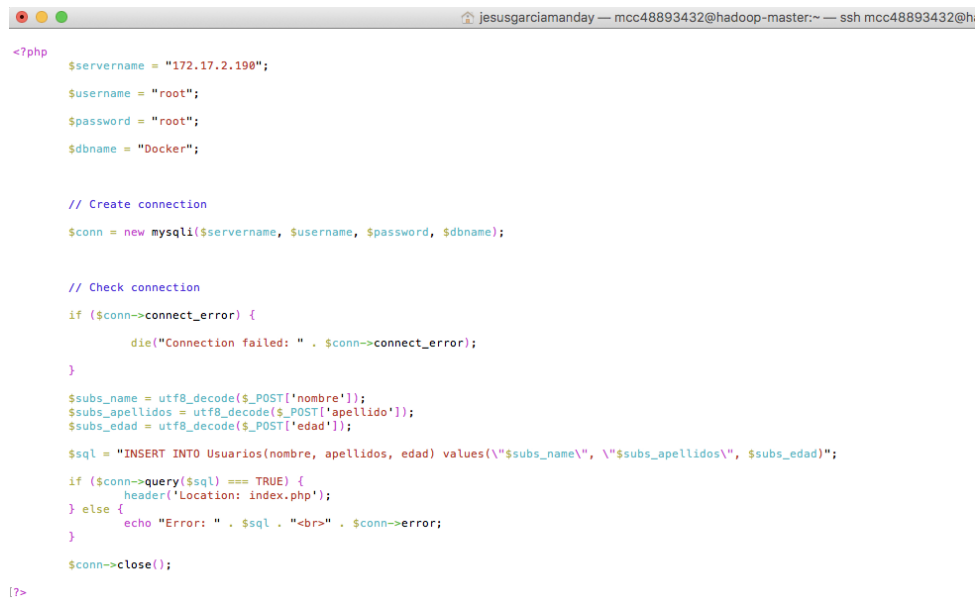
$sql = "INSERT INTO Usuarios(nombre, apellidos, edad) values('$subs_name', '$subs_apellidos', $subs_edad)";

if ($conn->query($sql) === TRUE) {
    header('Location: index.php');
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();

[?>
```

Figura 19: Fichero insert_user.php .



```
<?php
$servername = "172.17.2.190";
$username = "root";
$password = "root";
$dbname = "Docker";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$subs_name = utf8_decode($_POST['nombre']);
$subs_apellidos = utf8_decode($_POST['apellidos']);
$subs_edad = utf8_decode($_POST['edad']);

$sql = "DELETE FROM Usuarios WHERE nombre='$subs_name' AND apellidos='$subs_apellidos' AND edad=$subs_edad";

if ($conn->query($sql) === TRUE) {
    header('Location: index.php');
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();

[?>
```

Figura 20: Fichero delete_user.php .

En el segundo contenedor es donde reside el SGBD **MySQL** y por lo tanto es donde se ha creado la base de datos correspondiente sobre la que trabaja la aplicación. En dicha base de datos se ha creado la tabla **Usuarios** la cual va a representar el modelo de datos para almacenar la información referente a los usuarios que se añadan o eliminen desde la aplicación. En las imágenes de a continuación se muestra la conexión hacia dicho contenedor así como el uso de la base de datos, la creación de la tabla y la inserción de algunos registros.

```

<?php
    if(isset($_POST['id'])){
        $var=$_POST['id'];

        $servername = "172.17.2.190";
        $username = "root";
        $password = "root";
        $dbname = "Docker";

        // Create connection
        $conn = new mysqli($servername, $username, $password, $dbname);

        // Check connection
        if ($conn->connect_error) {
            die("Connection failed: " . $conn->connect_error);
        }

        $sql = "DELETE FROM Usuarios WHERE ID = $var";

        if ($conn->query($sql) === TRUE) {
            header('Location: index.php');
        } else {
            echo "Error: " . $sql . "<br>" . $conn->error;
        }

        $conn->close();
    }
}
[7>

```

Figura 21: Creando tabla Usuarios en base de datos.

```

[mysql> INSERT INTO Usuarios(nombre, apellidos, edad) values ('Jesus', 'Garcia Manday', 33);
Query OK, 1 row affected (0.05 sec)

[mysql> INSERT INTO Usuarios(nombre, apellidos, edad) values ('Roberto', 'Perez Hinojosa', 23);
Query OK, 1 row affected (0.05 sec)

[mysql> INSERT INTO Usuarios(nombre, apellidos, edad) values ('Luis', 'Alcala Torres', 19);
Query OK, 1 row affected (0.06 sec)

[mysql> INSERT INTO Usuarios(nombre, apellidos, edad) values ('Antonio', 'Espejo Martinez', 28);
Query OK, 1 row affected (0.07 sec)

[mysql> INSERT INTO Usuarios(nombre, apellidos, edad) values ('Carlos', 'Martin Fernandez', 28);
Query OK, 1 row affected (0.06 sec)

```

Figura 22: Insertando registros en la base de datos.

El resultado final de la aplicación lo podemos ver ejecutando desde el navegador de otra máquina cualquiera la dirección <http://hadoop.ugr.es:15500/> como se aprecia en la imagen de abajo.

Gestion de Usuarios

Lista de Usuarios

id	Nombre	Apellidos	Borrar
1	Jesus	Garcia Manday	Borrar
2	Roberto	Perez Hinojosa	Borrar
3	Luis	Alcala Torres	Borrar
4	Antonio	Espejo Martinez	Borrar
5	Carlos	Martin Fernandez	Borrar
6	Paco	Sanchez Sanchez	Borrar
8	Joaquin	Sanchez Sanchez	Borrar

Nuevo Usuario

Nombre (requerido)

Apellido

Edad

Figura 23: Aplicación web.

6. Análisis de la funcionalidad de los contenedores replicados.

Existe una funcionalidad a partir de la versión **1.12** de **Docker** que permite crear un clúster formado por varios Docker Hosts llamada **Swarm Mode**.

En un clúster de Swarm existen dos tipos de nodo, **Manager** y **Worker**. Los nodos **Manager** son los encargados de gestionar el clúster. Entre todos los Manager se elige automáticamente un líder que será el encargado de mantener el estado del clúster.

Los **Manager** también son los encargados de distribuir las tareas o task (unidades básicas de trabajo) entre todos los nodos **Worker**, los cuales reciben estas tareas y las ejecutan. Los nodos Manager por defecto también actúan como nodos Worker aunque se su configuración se puede cambiar para que solo asuma tareas de Manager. En la siguiente imagen se puede ver el esquema de esta funcionalidad de **Docker**.

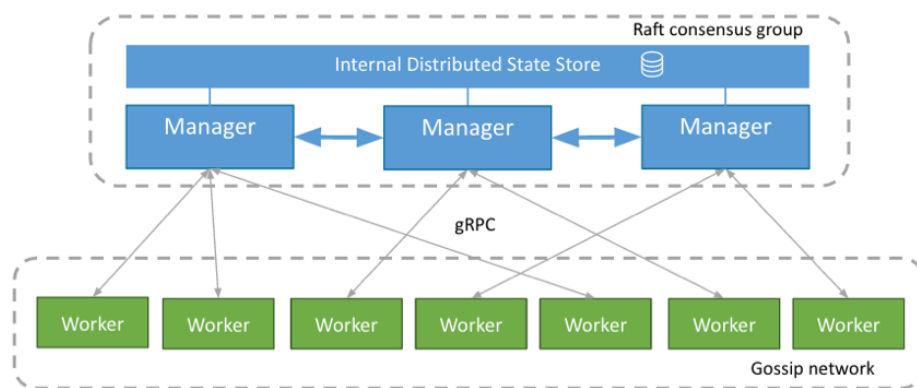


Figura 24: Estructura Swarm Mode.

Swarm incluye además nuevas funcionalidades como **escalado**, **balanceo de carga** y **rolling updates**. Estas utilidades serán empleadas con los dos contenedores creados anteriormente, ya que la idea es crear un clúster de contenedores donde exista dos nodos **Manager** (uno para el servicio Web y otro para la base de datos), y un conjunto de nodos **Worker** reunidos en dos grupos, uno que lo gestionará el nodo **Manager** del servicio Web, y otro que lo administrará el nodo **Manager** de la base de datos. En cada uno de los grupos se replicará el servicio del servicio de un contenedor principal, es decir, en los nodos **Worker** que sean gestionados por el nodo **Manager** del servicio Web se replicará todo lo referente a la aplicación y servicios instalados (ficheros php, Apache, etc), mientras que en los nodos **Worker** gestionados por el nodo **Manager** de la base de datos se replicará esta misma sobre todos los nodos, de manera que la información (tablas, registros, etc) quedará replicada en cada nodo.

Al hacer esto el proyecto gana en robustez y rapidez, ya que las funcionalidades de escalado y balanceo de carga van a permitir que los servicios sigan activos aunque algún nodo haya caído por cualquier problema, y que no exista sobrecarga a un sólo nodo y la carga de trabajo o tareas sea repartida entre los diferentes nodos del clúster.

7. Configuración de OwnCloud en docker.

En este punto se va a proceder a configurar el servicio **ownCloud** en un contenedor **Docker**. De este manera se consigue tener dicha aplicación de un modo más aislado y portable, evitando también el instalarla directamente sobre la máquina.

Lo primero que haremos será buscar si existe alguna imagen en **Docker** con dicho servicio, para ello volvemos a usar el comando **docker hub**.

```

[mcc48893432@hadoop-master ~]$ docker search owncloud
NAME                DESCRIPTION                STARS     OFFICIAL   AUTOMATED
ownCloud             ownCloud is a self-hosted file sync and sh... 679      [OK]
l3iggs/owncloud      Bleeding edge ownCloud container with the ... 124
jchaney/owncloud     ownCloud 7 on Nginx        51
pschmitt/owncloud    Reasonably configurable Docker image for t... 22
dperson/owncloud     Spoke container for Owncloud, an open sour... 11
radial/owncloud      Bitnami Docker Image for Owncloud           4
bitnami/owncloud     ownCloud is a self-hosted file sync and sh... 3
freenas/owncloud     Deploy the latest ownCloud on Debian 8 wit... 2
deserbit/owncloud    ownCloud/MySQL on Alpine (Image size: 98 MB) 1
splattael/owncloud   Owncloud                    1
actualys/owncloud    Dockerized OwnCloud based on alpine.         1
martingabelmann/owncloud ownCloud running on Nginx with data persis... 1
dinkel/owncloud      The official owncloud builds but with LDAP... 1
mrskensington/owncloud-ldap Docker image for ownCloud community edition 1
d4v1d31/owncloud     Simple OwnCloud container based on alpine ... 1
ndslabs/owncloud     NDS Labs Service: ownCloud                   0
derjudge/owncloud    Arch Linux based ownCloud server            0
ownCloud Docker image.
whatwedo/owncloud    Based on the official owncloud image with ... 0
cloyne/owncloud      Docker images for owncloud                   0
jangmarker/owncloud  based on the official owncloud build, incl... 0
fusengine/owncloud   Owncloud nginx ph7                         0
gnarea/owncloud      Lightweight Docker image for ownCloud v9     0

```

Figura 25: Buscando imagen en Docker.

Como se aprecia en la anterior figura, existe una imagen en **Docker** con el servicio **ownCloud**. Es esta la imagen que vamos a utilizar para crear y lanzar un contenedor con dicho servicio. Antes de lanzar el contenedor con **OwnCloud** vamos a lanzar otro que tenga el servicio de **PostgreSQL** del que habiendo buscado en el hub de **Docker** se ha podido ver que existe una imagen oficial con dicho servicio.

```

[mcc48893432@hadoop-master ~]$ docker run --name postgres-jmanday -e POSTGRES_PASSWORD=root -d postgres
19964ab3df50ef30aad68593011e9f91cec5afe6d7fb2f0419e35c65637b9e

```

Figura 26: Lanzando contendor con PostgreSQL en Docker.

Con el contenedor de **PostgreSQL** lanzado, lo siguiente es lanzar el contenedor de **ownCloud** al cual se le enlazará el host (contenedor) con la base de datos creada anteriormente.

```

[mcc48893432@hadoop-master ~]$ docker run --rm -it --link postgres-jmanday:owncloud --name owncloud-jmanday -p 15080:80 owncloud
AMR8958: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.180. Set the 'ServerName' directive globally to suppress this message
AMR8958: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.180. Set the 'ServerName' directive globally to suppress this message
[Mon Apr 17 17:11:33.224000 2017] [mpm_prefork:notice] [pid 1] AH00084: Command Line: 'apache2 -D FOREGROUND'
47.63.2.91 - [17/Apr/2017:17:12:02 -0800] "GET / HTTP/1.1" 200 1559 "-" Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36"
47.63.2.91 - [17/Apr/2017:17:12:02 -0800] "GET /core/css/styles.css?vba2226ed25d9570808d3be9f4333dc HTTP/1.1" 200 6273 "-" Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36"
47.63.2.91 - [17/Apr/2017:17:12:02 -0800] "GET /core/css/inputs.css?vba2226ed25d9570808d3be9f4333dc HTTP/1.1" 200 2236 "-" Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36"
47.63.2.91 - [17/Apr/2017:17:12:02 -0800] "GET /core/css/header.css?vba2226ed25d9570808d3be9f4333dc HTTP/1.1" 200 2329 "-" Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36"
47.63.2.91 - [17/Apr/2017:17:12:02 -0800] "GET /core/css/inputs.css?vba2226ed25d9570808d3be9f4333dc HTTP/1.1" 200 1748 "-" Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36"
47.63.2.91 - [17/Apr/2017:17:12:02 -0800] "GET /core/css/fonts.css?vba2226ed25d9570808d3be9f4333dc HTTP/1.1" 200 626 "-" Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36"
47.63.2.91 - [17/Apr/2017:17:12:02 -0800] "GET /core/css/apps.css?vba2226ed25d9570808d3be9f4333dc HTTP/1.1" 200 2768 "-" Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36"
47.63.2.91 - [17/Apr/2017:17:12:02 -0800] "GET /core/css/global.css?vba2226ed25d9570808d3be9f4333dc HTTP/1.1" 200 688 "-" Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36"
47.63.2.91 - [17/Apr/2017:17:12:02 -0800] "GET /core/css/inputs.css?vba2226ed25d9570808d3be9f4333dc HTTP/1.1" 200 688 "-" Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36"

```

Figura 27: Lanzando contendor con ownCloud en Docker.

```

[mcc48893432@hadoop-master ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED          STATUS          PORTS                               NAMES
02811468a8b        owncloud           "entrypoint.sh apac... 49 seconds ago  Up 8 seconds   0.0.0.0:15588->88/tcp               owncloud-jmanday
19964ab3df50        postgres           "docker-entrypoint... 25 minutes ago  Up 25 minutes   5432/tcp                           postgres-jmanday
333a19754f1a        mongo             "docker-entrypoint... About an hour ago  Up About an hour 27017/tcp, 0.0.0.0:14084->27017/tcp server1-gmnr
28cfe896ed66        owncloud           "entrypoint.sh apac... 2 hours ago      Up 2 hours      0.0.0.0:14086->88/tcp               owncloudserverCesar
c22ef4c5b85        joomla/joomla-cms-4.0.10-latest "php-fpm --fpm-user=www-data" 2 hours ago      Up 2 hours      22/tcp, 0.0.0.0:14087->3306/tcp      mysqlCesar-owncloud
f8085c376726        owncloud           "bin/bash -c /init/... 4 hours ago      Up 4 hours      0.0.0.0:14081->88/tcp               server1-gmnr
b31c4856a4c3        eboraa/apache-php  "usr/sbin/apache2ct... 5 hours ago      Up 5 hours      443/tcp, 0.0.0.0:14082->88/tcp       apache-pho-gmnr
83c8b7f5687        owncloud           "entrypoint.sh apac... 22 hours ago     Up 22 hours     0.0.0.0:14084->88/tcp               jablo_owncloud
c6784f043382        postgres           "docker-entrypoint... 22 hours ago     Up 22 hours     5432/tcp                           jablo_postgres
976a032327d2        owncloud           "entrypoint.sh apac... 28 hours ago     Up 28 hours     0.0.0.0:14083->88/tcp               owncloudAlejandroCasado
e44860277f9        webdevops/nginx    "opt/docker/bin/ent... 31 hours ago     Up 31 hours     443/tcp, 0.0.0.0:14028->88/tcp       servidorWebAlejandroCasado
57f0a933521a        mysql/mysql-server "entrypoint.sh mysq... 32 hours ago     Up 32 hours     3306/tcp, 0.0.0.0:14021->3306/tcp     BDAlejandroCasado
c287c85984f1        mysql              "docker-entrypoint... 46 hours ago     Up About an hour 3306/tcp                           mysql-jmanday
1896422612c5        nrmis/apache-php5  "httpd -DSSL -DSSL_... 47 hours ago     Up 47 hours     0.0.0.0:15588->88/tcp, 0.0.0.0:32771->443/tcp apache-jmanday
496dca265286        mongo-xythae_img  "mongo --xythae_img... 2 days ago       Up 2 days       27017/tcp, 208017/tcp               mongo-xythae
58a5c514e49        owncloud           "entrypoint.sh apac... 3 days ago       Up 3 days       0.0.0.0:14031->88/tcp               owncloud_xythae
1a73882686c2        postgres           "docker-entrypoint... 3 days ago       Up 3 days       5432/tcp                           owncloud-postgres_xythae
af85b62c0387        postgres           "docker-entrypoint... 3 days ago       Up 3 days       5432/tcp                           owncloudPostgresAlejandroCasado

```

Figura 28: Comprobando que ambos contenedores se han lanzado correctamente.

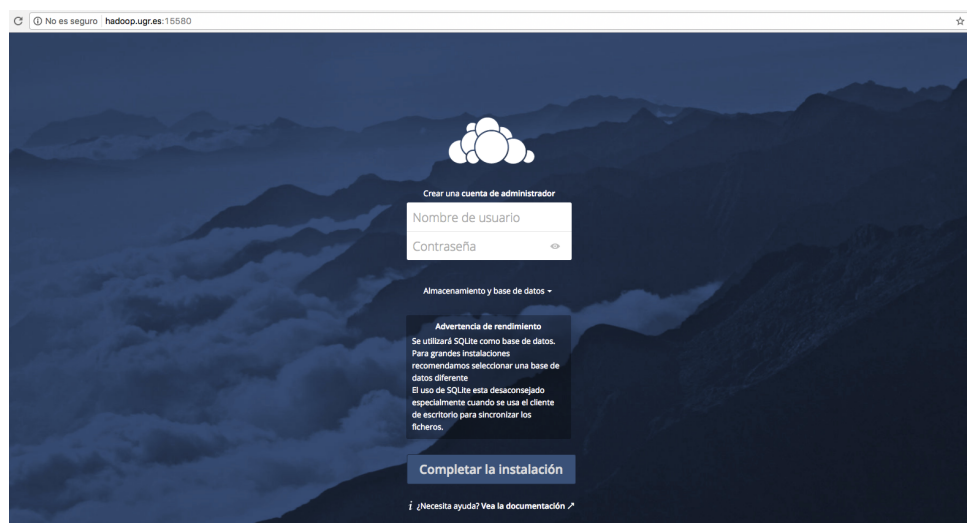


Figura 29: Comprobando que se esta ejecutando la aplicación.

Una vez realizados los pasos anteriores, se puede comprobar con las imagenes como la configuración del servicio **ownCloud** se ha realizado correctamente y su funcionamiento es el deseado.

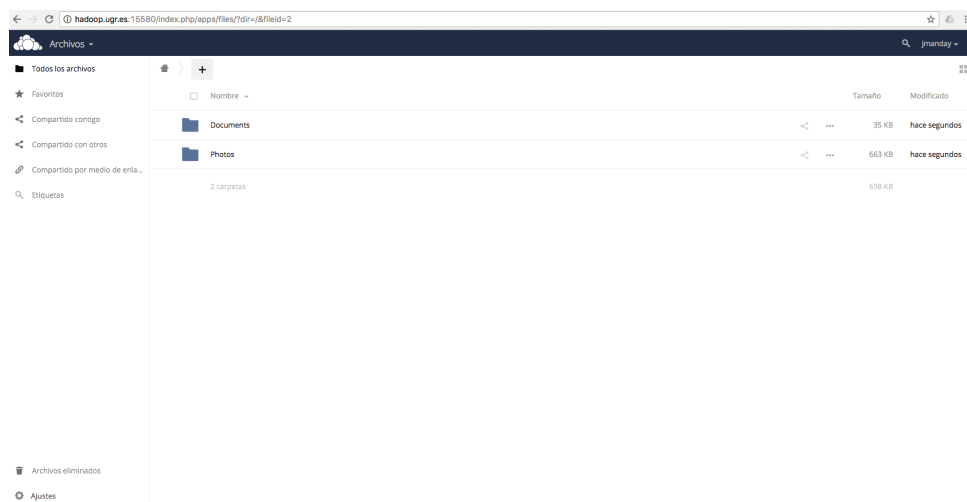


Figura 30: Probando ownCloud (I).

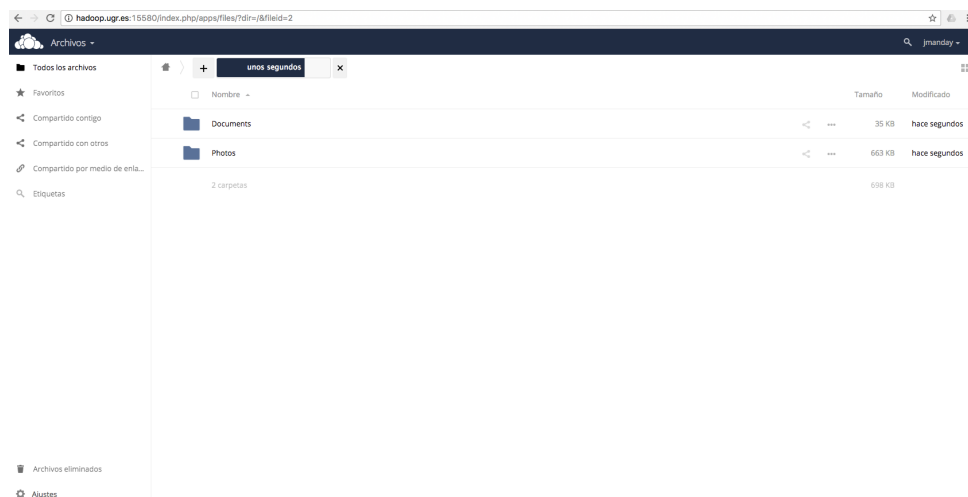


Figura 31: Probando ownCloud (II).

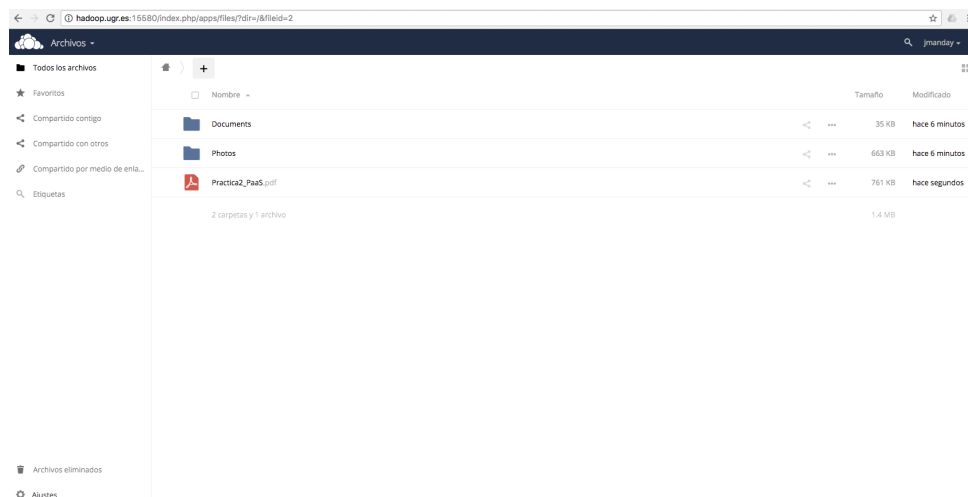


Figura 32: Probando ownCloud (III).