

Dockerfile y Docker Compose

En el mundo de los dockers existen formas de **automatizar** o generar acciones repetidas teniéndolas **preprogramadas**. Eso qué significa, que podemos trabajar un fichero base, por ejemplo, un docker con IIS que nos servirá para múltiples proyectos. Os mostramos dos opciones Dockerfile y Docker Compose.

Crear un Dockerfile

**Dockerfile** no es un comando. Como su propio nombre indica **es un archivo**. Este archivo no tiene extensión y podemos crearlo con cualquier editor de texto. Dentro de este archivo lo que vamos a encontrar son las instrucciones para crear una imagen.

**¿Porque utilizar un Dockerfile?**

 Ahora veamos que hace Dockerfile en la situación anterior. Un Dockerfile es un **archivo de texto plano que contiene una serie de instrucciones necesarias para crear una imagen** que, posteriormente, se convertirá en **una sola aplicación** utilizada para un determinado propósito. Similar a lo explicado anteriormente, y **la base del funcionamiento de Docker es mediante Dockerfiles**.

**Como hacer un Dockerfile**

Se pueden generar imágenes a partir de **Dockerfiles**, unos ficheros que crearemos manualmente y que contendrán una secuencia de instrucciones que por medio de una imagen base nos permite crear de otras.

Para crear un **Dockerfile** nos hace falta un editor de texto y saber de antemano la sintaxis de los comandos para luego poder construirla.

Crearemos un fichero llamado **Dockerfile** (sin extensión alguna):

Los pasos principales para crear una imagen a partir de un fichero Dockerfile son:

* Crear un nuevo directorio que contenga el fichero, con el guion y otros ficheros que fuesen necesarios para crear la imagen.
* Crear el contenido.
* Construir la imagen mediante el comando docker build.

La sintaxis para el comando es:

docker build [opciones] RUTA | URL | -

Hay **diferentes maneras de construir nuestro dockerfile**, a continuación, se muestra un dockerfile mediante el cual construiremos una imagen con base **sistema operativo Ubuntu 14:04**, y sus sub-tareas serán:

* Instalación y actualización de paquetes del sistema operativo
* Establecer variable de entorno llamada DEBIAN\_FRONTEND
* Instalación de git

*# Descarga la imagen de Ubuntu 14.04*

FROM ubuntu:14.04

MAINTAINER juan juan@gmail.com

*# Actualiza la imagen base de Ubuntu 14.04*

RUN apt-get update

*# Definir ambiente de entorno*

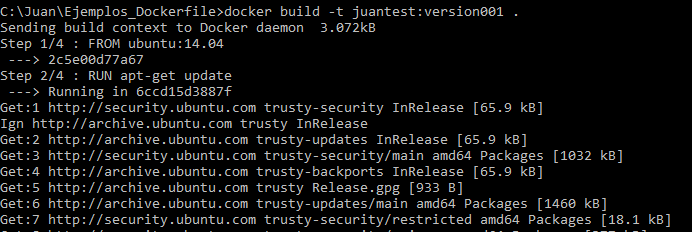
ENV DEBIAN\_FRONTEND noninteractive

*# Instalar Git*

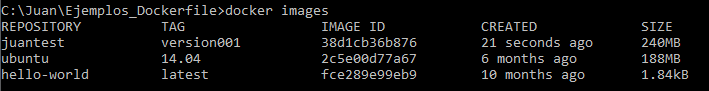
RUN apt-get -qqy install git

Después lo guardamos. Ahora vamos a construir la imagen. Nos situamos en la carpeta que contiene el Dockerfile. En el docker build vamos a indicar el **nombre** juantest y la **etiqueta** version001. Luego con el **último punto** vamos a indicar que el **dockerfile** está en el mismo directorio en el que estamos.

docker build -t juantest:version001 .



Y ya tendríamos nuestra imagen personalizada con el **Dockerfile** lista para crear un nuevo contenedor en **Docker**. Recuerda, no es recomendado borrar el **dockerfile** después de usarlo. Puedes crear un nuevo directorio y tenerlos ordenados por si los necesitamos de nuevo o simplemente para revisar.



Luego podemos actualizar esa imagen simplemente modificando la etiqueta. El último paso ya es hacer un docker run para crear un contenedor.

**Vamos a ver un poco el código del archivo Dockerfile**

**MAINTAINER:** (**opcional**) Podemos añadir nuestro nombre y correo. Los datos de quien ha creado el Dockerfile y la imagen.

Ejemplo:

MAINTAINER juan juan@gmail.com

**FROM**: (**Obligatorio**) indica la imagen base sobre la que se construirá la aplicación dentro del contenedor.

**Sintaxis:**

FROM <imagen>

FROM <imagen>:<tag>

Ejemplo: FROM ubuntu:14.04

Por ejemplo la imagen puede ser un sistema operativo como Ubuntu, Centos, etc. O una imagen ya existente en la cual con base a esta queramos construir nuestra propia imagen.

**RUN**: nos permite **ejecutar comandos** **en el contenedor**, por ejemplo, **instalar paquetes o librerías** (apt-get, yum install, etc.). Nos crea un contenedor intermedio. Además, tenemos **dos formas de colocarlo**:

**Ejemplo**: RUN apt-get update

**Sintaxis:**

* **Opción 1**🡪 RUN <comando>

Esta instrucción ejecuta comandos Shell en el contenedor.

* **Opción 2**🡪 [“ejecutable”,” parametro1”,” parametro2”]

Esta otra instrucción bastante útil, que permite ejecutar comandos en imágenes **que NO tengan** /bin/sh.

**ENV** 🡪 establece variables de entorno para nuestro contenedor, en este caso la variable de entorno es

DEBIAN\_FRONTEND noninteractive, el cual nos permite instalar un montón de archivos *.deb* sin tener que interactuar con ellos.

**Sintaxis:**

ENV <key><valor>

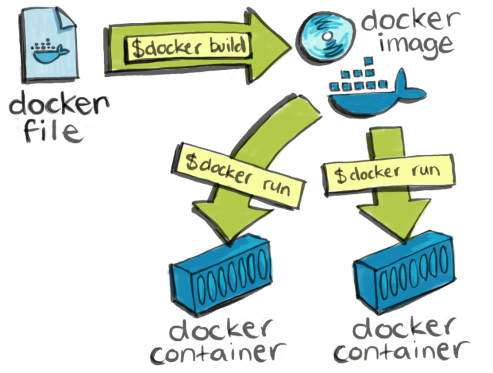
Podemos llamarlas después dentro del contendor

Ejemplo:

**ENV** nombre luis

Luego dentro del contenedor:

echo $nombre



Además de las instrucciones anteriormente explicadas, también existen otras como:

**ADD** -> esta instrucción **copia archivos a un destino específico** dentro del contenedor, normalmente nos sirve para dejar ubicados ciertos archivos que queremos mover entre directorios.

**Sintaxis:**

ADD <fuente> <destino>

**Ejemplo:**

ADD ./script.sh /var/tmp/script.sh

**MAINTAINER**: Este nos permite indicar el nombre del **autor del dockerfile**.

**Sintaxis:**

MAINTANER <nombre> <” correo”>

**Ejemplo:**

MAINTAINER Pedro Gómez “pedro@email.com”

**CMD** -> El comando que se especifique a continuación, se **ejecutara** cuando haga docker run si no se especificar otro comando.

Ejemplo:

**CMD** apt-get update

A **diferencia** del comando **RUN**, los comandos que se pasen por medio de este método se ejecutan **una vez que el contenedor se ha inicializado**, mientras que **RUN** se utiliza para **crear la imagen de un contenedor.**

**ENTRYPOINT** 🡪 la instrucción entrypoint define el comando que se va a ejecutar durante la puesta en marcha de un contenedor, es decir, **el proceso primario** que este último portara.

Los ejemplos tipo de su uso, son cuando se quiere levantar un **servidor web**, una **base de datos**, etc ….

**VOLUME** 🡪 esta instrucción crea un volumen **como punto de montaje** dentro del contenedor y es visible desde el host anfitrión marcado con otro nombre.

Sintaxis:

VOLUME /var/tmp

**USER** 🡪 **determina el nombre de usuario a utilizar** cuando se ejecuta un contenedor, y adicionalmente cuando se ejecutan comandos como RUN, CMD, ENTRYPOINT o WORKDIR.

**Ejemplo:**

**USER** juan

**WORKDIR**: Especifica el directorio desde el que se van a ejecutar las órdenes de:

* **RUN**
* **CMD**
* **ENTRYPOINT**
* **COPY**
* **ADD**

WORKDIR ruta/de/Proyecto

Sustituye al comando cd. Y si no existe el directorio, lo crea.

Ejemplo:

**WORKDIR** /midirectorio

**EXPOSE 🡪** permite exponer un **puerto de red al exterior,** algo parecido a lo que hace la palabra VOLUME, con las secciones del sistema de archivos. De la misma manera que no es obligatorio conectar el volumen a un directorio de la máquina host, tampoco lo es poner en correspondencia el puerto expuesto con un puerto de la máquina host, pero este comando permite hacerlo, utilizando la **opción –p** del comando **docker run.**

También, si solo usas **EXPOSE** y al hacer run no especificas ni -p ni -P lo que va a pasar es que esos puertos **solo sean accesibles** **desde otros contenedores**, no desde el exterior.

Se puede especificar también si es TCP o UDP. Por defecto es TCP.

Más ejemplos

## Creación una imagen con el servidor web Apache2

En este caso vamos a crear un directorio nuevo que va a ser el contexto donde podemos guardar los ficheros que se van a enviar al docker engine, en este caso el fichero index.html que vamos a copiar a nuestro servidor web:

$ mkdir apache

$ cd apache

Creamos un fichero que se llame **index.html** con este contenido

<h1>Prueba de funcionamiento contenedor docker</h1>

En ese directorio vamos a crear un fichero Dockerfile, con el siguiente contenido:

FROM ubuntu:16.04

MAINTAINER Juan "juan@gmail.com"

RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm -rf /var/lib/apt/lists/\*

EXPOSE 80

ADD ["index.html","/var/www/html/"]

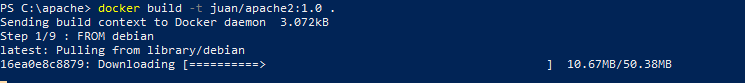
# Launch Apache

ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]

En este caso utilizamos la imagen base de debian, instalamos el servidor web apache2, para reducir el tamaño, borramos la caché de paquetes apt y la lista de paquetes descargada, , exponemos el puerto http TCP/80, copiamos el fichero index.html al DocumentRoot y finalmente indicamos el comando que se va a ejecutar al crear el contenedor, y además, al usar el comando **ENTRYPOINT**, **NO permitimos ejecutar ningún otro comando durante la creación**.

Vamos a generar la imagen:

docker build -t juan/apache2:1.0 .



Generamos la nueva imagen con el comando docker build con la **opción -t** indicamos **el nombre de la nueva imagen** (para indicar el nombre de la imagen es recomendable usar nuestro nombre de usuario en el registro **docker hub**, para posteriormente poder guardarlas en el registro), mandamos todos los ficheros del contexto (indicado con el punto). Podemos comprobar que tenemos generado la nueva imagen:



A continuación podemos crear un **nuevo contenedor(servidor\_web**) a partir de la nueva imagen:

docker run -p 85:80 --name **servidor\_web** juan/apache2:1.0

Comprobamos que el contenedor está creado:



## Ejemplo de contenedor para aplicaciones web en PHP

Vamos a construir un contenedor que incluya de forma estática una aplicación (p.e. la última versión de la aplicación). El proceso a seguir es:

1. Creación de la aplicación.
2. Creación del Dockerfile para generación de la imagen.
3. Generación de la imagen.

A partir de una carpeta nueva crearemos lo siguiente:

* Archivo Dockerfile
* Carpeta html con los scritps de nuestra aplicación
* Archivo html/index.php con el código de nuestra aplicación

El **Dockerfile**

FROM tutum/apache-php

MAINTAINER Juan "juan@gmail.com"

EXPOSE 80

ADD html /var/www/html

Archivo html/index.php de ejemplo

**<?php**

**echo** "Hola desde Docker";

**?>**

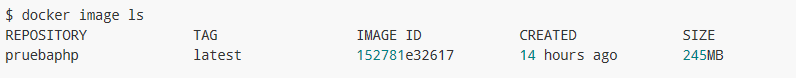
#### Construcción de la imagen.

docker build -t pruebaphp .

Con **-t** definimos una etiqueta o nombre de la imagen. Al construir la imagen pasa a nuestro registro local.

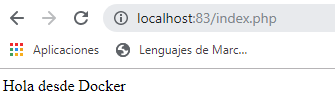
#### Listado de imágenes locales

docker image ls



#### Creación de un contenedor a partir de la imagen

docker run -d -p 83:80 152781e32617



Un posible inconveniente que podemos encontrar en este ejemplo es que la aplicación va incluida en la propia imagen, por lo que para actualizar la aplicación deberemos crear una nueva imagen, y después crear un nuevo contenedor a partir de ella desechando el contenedor anterior.

|  |
| --- |
| A la hora de distribuir y actualizar aplicaciones podemos incluir la aplicación en la imagen.  Con un ciclo de CI/CD tendríamos la aplicación actualizada al actualizar su repositorio. |

Ejemplo de contenedor con volumen externo

En este ejemplo la aplicación la tendremos aparte en un volumen externo accesible por el contenedor. De esta forma, si nuestra aplicación está vinculada a un repositorio, la actualización de la aplicación se realiza descargando la última versión del repositorio, manteniendo intacto el contenedor.

La forma de usar volúmenes con Dockerfile consiste en:

1. Añadir en el Dockerfile la lista de carpetas que se montarán con volúmenes externos
2. Al crear el contenedor indicar el punto de montaje en el host remoto en forma de ruta absoluta

El Dockerfile

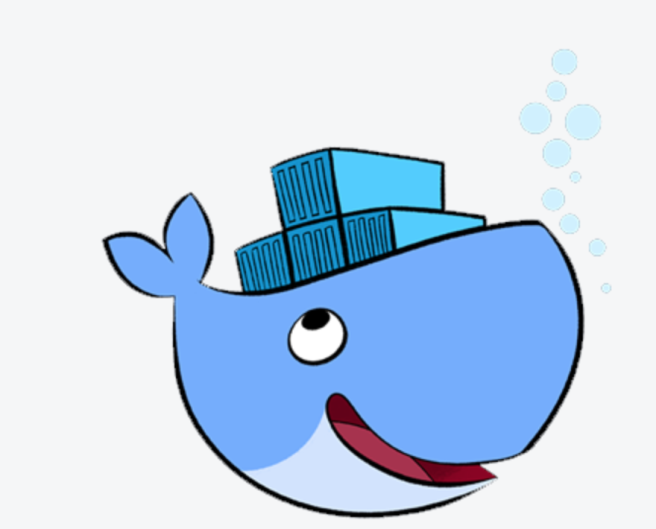
FROM juan/apache2:1.0

VOLUME /var/www/html

EXPOSE 80

$ docker run -d -p 83:80 -v=/Users/manolo/Documents/Desarrollo/SeminarioDocker/phpsimple/html:/var/www/html prueba.php

|  |
| --- |
| También podemos hacer uso de la evaluación de órdenes con apóstrofes para obtener  el path actual y añadirle sólo la carpeta html.  $ docker run -d -p 83:80 -v=’pwd’/html:/var/www/html prueba.php |



**Comparativa Dockerfile vs Docker Compose**

Teniendo claro que es y cómo se construye *dockerfile*, vamos a hablar un poco acerca de docker-compose.

**Docker Compose** es una herramienta para definir y ejecutar aplicaciones Docker con **varios contenedores**.

***Docker compose***, es una herramienta que nos permite facilitar el uso de docker, obteniendo como resultado un **conjunto de scripts que nos facilitan la construcción de nuestros servicios**. La mayor ventaja de este es que podemos crear **diferentes contenedores y por cada contenedor diferentes servicios**. ¿Cómo es esto? Vamos a ver.

*Docker compose* nos provee un esquema para indicarle al *engine* realizar **determinadas tareas escritas en lenguaje *yaml****,* similar a *Ansible,* para quienes lo hemos usado en el caso de automatizar la construcción de servicios. Para colocarlo en términos más claros, el *dockerfile*define como crear la imagen de una aplicación, en cambio el *docker compose* nos permite**vincular y configurar estos contenedores en conjunto** para construir varios servicios.

**Flujo de trabajo básico con Docker Compose**

1. Crear el archivo **docker-compose.yml**con los servicios de la aplicación (p.e. php y mysql)
2. Construir y lanzar el entorno en modo *dettached* con **docker-compose up -d**
3. Echar abajo el entorno con **docker-compose down**

**Comandos básicos para Docker Compose**

$ **docker-compose** up -d

**docker-compose** inicia los contenedores en el orden que hemos indicado en el archivo de definición, Indicando la opción **-d** los contenedores se iniciarán en **segundo plano**

$ **docker-compose** pull

Descarga las imágenes pero no inicia los contenedores

$ **docker-compose** rm [-fs]

Borra los contedores parados. Con -fs los detiene y fuerza su borrado

$ **docker-compose** ps

podremos ver el estado de los contenedores y de cuales está compuesta la aplicación

$ **docker-compose** stop

podremos pararlos

$ **docker-compose** restart

$ **docker-compose** logs

veremos las trazas emitidas por los servicios que nos serán de utilizar si iniciamos los contenedores en segundo plano.

$ **docker-compose** –help

podemos ver la lista completa de comandos que podemos usar.

Nuestro primer Docker Compose

Gestionar docker a pelo es un poco tedioso: tienes que levantar los contenedores en orden, meter los parámetros de configuración por línea de comandos, etc. Por suerte, existe una herramienta llamada docker-compose que nos permite crear un fichero de configuración con todos los contenedores que queremos levantar, establecer las dependencias entre ellos y configurarlos individualmente. Veamos un ejemplo sencillo.

Como puedes imaginar, en el repositorio de contenedores de Docker existe un contenedor (varios, de hecho) de WordPress. Dicho contenedor tiene varios inconvenientes:

* Por un lado, con él sólo no podremos ejecutar WordPress, ya que no incluye la base de datos. Así que si queremos usar WordPress debemos tener corriendo una base de datos en nuestro propio ordenador, en otra máquina o, ¿cómo no?, en otro contenedor.
* Por otro lado, el contenedor de WordPress dispone de varios parámetros de configuración que debemos especificar para que funcione (por ejemplo, dónde está la base de datos y qué credenciales usar para conectarse a ella).

Empecemos por crear un directorio vacío donde meter el fichero de configuración. Por ejemplo, crea el directorio **C:/docker/test** y añade un fichero de texto plano llamado **docker-compose.yml** con el siguiente contenido:

version: '2'

services:

  mysql:

    image: mysql:5.7

    restart: always

    ports:

      - 8081:3306

    environment:

      MYSQL\_USER: wordpress

      MYSQL\_ROOT\_PASSWORD: wordpress

      MYSQL\_DATABASE: wordpress

      MYSQL\_PASSWORD: wordpress

  wordpress:

    depends\_on:

      - mysql

    image: wordpress

    ports:

      - 8080:80

    restart: always

    environment:

      WORDPRESS\_DB\_HOST: mysql:3306

      WORDPRESS\_DB\_USER: wordpress

      WORDPRESS\_DB\_PASSWORD: wordpress

No te asustes, que se trata de un fichero muy sencillo. Fíjate bien. En él definimos dos servicios: mysql y wordpress. En el primer servicio (mysql) indicamos la imagen que queremos descargarnos (un MySQL versión 5.7) y algunos parámetros de configuración. Los más relevantes son la redirección de puertos (del 8081 del anfitrión al 3306 del huésped) y parámetros de la base de datos en sí (usuario, contraseña, etc).

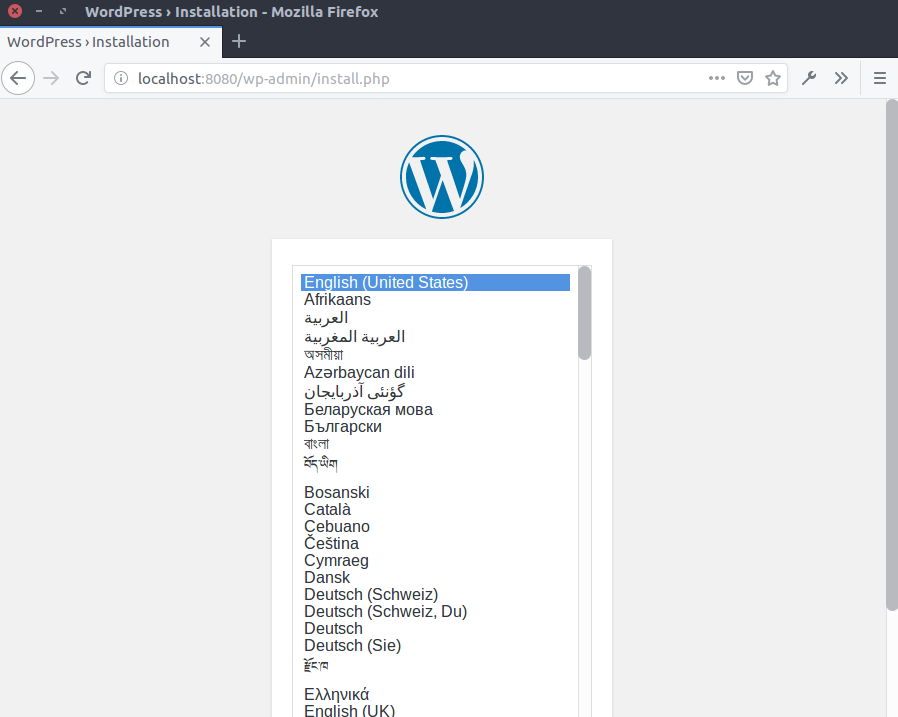
El segundo servicio es WordPress (wordpress) y sigue un patrón parecido. Aquí, por ejemplo, indicamos que queremos usar el puerto 8080 para acceder a nuestro WordPress. También especificamos que mysql es una dependencia de WordPress (este contenedor no puede funcionar sin el otro). Y también indicamos parámetros adicionales como, por ejemplo, dónde está la base de datos y cómo acceder a ella.

Una vez tenemos el fichero listo, ejecutamos:

**docker-compose up -d**

y Docker se descargará las imágenes de WordPress y MySQL (si no las tenemos ya) y pondrá en marcha nuestro WordPress.

Cuando esté listo **( ¡OJO !dale un momento, porque a veces tarda un poco en tenerlo todo en marcha**), podremos ir a nuestro navegador web y en la dirección http://localhost:8080 (fíjate que es el mismo puerto que hemos puesto en el fichero de configuración) encontraremos nuestro WordPress esperándonos para completar su instalación:



A partir de este punto, si quieres nuevos WordPress en paralelo, sencillamente tendrías que repetir el proceso. Es decir, creas un nuevo directorio en ~/docker/, añades el fichero docker-compose.yml, cambias los puertos para acceder a él (por ejemplo, 8082 para WordPress y 8083 para MySQL) para que no colisione con el contenedor que ya tenemos en marcha y lo pones a correr también.

Cuando acabes de trabajar, puedes ejecutar:

**docker-compose stop**

para detener la instancia. ¡Cuidado! Si en lugar de stop usaras down:

**docker-compose down**

también detendrías la instancia pero, además, se eliminaría, con lo que la próxima vez que hicieras el docker-compose up empezaría todo el proceso **de cero** (base de datos vacía, WordPress sin instalar, etc).

## Ejemplo: Aplicación web (PHP) con soporte de Base de datos (MySQL)

Basta con ejecutar los tres comandos siguientes en una máquina con Docker instalado para tener una aplicación web que muestra un catálogo de clientes almacenados en una base de datos MySQL.

$ git clone https://github.com/juanaves/Docker\_Compose\_PHP\_MySQL.git

$ cd docker\_customer\_catalog

$ docker-compose up -d

* Aplicación que muestra un listado de clientes almacenado en una base de datos MySQL.
* Podemos distribuirla con un repositorio que incluya una carpeta html con la aplicación PHP.
* Al iniciar el servicio MySQL se ejecutará un script de inicialización de la base de datos.
* Usaremos volúmenes externos para la base de datos y para la aplicación web para asegurar la persistencia de los cambios.

Comencemos clonando el repositorio de la aplicación:

$ git clone https://github.com/juanaves/Docker\_Compose\_PHP\_MySQL.git

En ese repositorio se encuentra:

* Un archivo docker-compose.yml que configura dos serivicios. Un servicio para almacenamiento de datos con MySQL y otro servicio para el front-end PHP de la aplicación.
* Una carpeta html con la aplicación. Esta carpeta será la que monte el servicio front-end, de forma que la aplicación no está almacenada en el contenedor.
* Un script SQL init.sql que inicializa la base de datos de nuestra aplicación. La base de datos se almacena en nuestro host, garantizando almacenamiento persistente.

**docker-compose.yml**

docker-compose.yml

version: '2'

services:

  mysql:

    container\_name: my\_mysql

    restart: always

    image: mysql:5.7

    environment:

      MYSQL\_ROOT\_PASSWORD: 'secret' # TODO: Change this

    ports:

      - "3306:3306"

    volumes:

      - ./data:/var/lib/mysql

      - ./init.sql:/docker-entrypoint-initdb.d/init.sql

  php:

    container\_name: my\_php

    restart: always

    image: tutum/apache-php

    ports:

      - "80:80"

    volumes:

      - ./html:/var/www/html

* “*volumes*”: Aqui hacemos que el directorio local ./hmtl se mapee directamente con el /var/www/html, lugar donde hemos creado la aplicación. De este modo, cualquier cambio en el directorio local en el host, se hará de inmediato en el contenedor.

Montar una carpeta data de nuestro host en la ruta en la que el servicio mysql almacena la base de datos

La imagen de MySQL ejecutará al inicio cualquier script que encuentre en `/docker-entrypoint-initdb.d/

Montar una carpeta html de nuestro host en la ruta en la que el servicio php almacena la aplicación

index.php [Descargar index.php](https://gist.githubusercontent.com/ualmtorres/0c9ba76eb22a35e328dbc322e6c100d1/raw/812f0db2da07037416db8967130eb16b16b5b88e/index.php)

<!DOCTYPE html>

<html>

<head>

  <meta charset="utf-8">

  <meta http-equiv="X-UA-Compatible" content="IE=edge">

  <title>Web PHP-MySQL con Docker</title>

  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css" integrity="sha384-WskhaSGFgHYWDcbwN70/dfYBj47jz9qbsMId/iRN3ewGhXQFZCSftd1LZCfmhktB" crossorigin="anonymous">

</head>

<body>

  <div class = "container">

    <div class="jumbotron">

      <h1 class="display-4">Docker app</h1>

      <p class="lead">Ejemplo de aplicacion PHP y MySQL con contenedores</p>

      <hr class="my-4">

      <p>Usa un contenedor para Apache/PHP y otro para MySQL con almacenamiento de aplicación y de datos en volúmenes externos</p>

    </div>

    <table class="table table-striped table-responsive">

      <thead>

        <tr>

          <th>Name</th>

          <th>Credit Rating</th>

          <th>Address</th>

          <th>City</th>

          <th>State</th>

          <th>Country</th>

          <th>Zip</th>

        </tr>

      </thead>

      <tbody>

        <?php

        $conexion = mysql\_connect("my\_mysql", "root", "secret");

        mysql\_select\_db("SG", $conexion);

        $cadenaSQL = "select \* from s\_customer";

        $resultado = mysql\_query($cadenaSQL);

        while ($fila = mysql\_fetch\_object($resultado)) {

         echo "<tr><td> " .$fila->name .

         "</td><td>" . $fila->credit\_rating .

         "</td><td>" . $fila->address .

         "</td><td>" . $fila->city .

         "</td><td>" . $fila->state .

         "</td><td>" . $fila->country .

         "</td><td>" . $fila->zip\_code .

         "</td></tr>";

       }

       ?>

     </tbody>

   </table>

 </div>

 <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

 <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js" integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49" crossorigin="anonymous"></script>

 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js" integrity="sha384-smHYKdLADwkXOn1EmN1qk/HfnUcbVRZyYmZ4qpPea6sjB/pTJ0euyQp0Mk8ck+5T" crossorigin="anonymous"></script>

</body>

</html>

init.sql [Descargar init.sql](https://gist.githubusercontent.com/ualmtorres/eb328b653fcc5964f976b22c320dc10f/raw/448b00c44d7102d66077a393dad555585862f923/init.sql)

CREATE SCHEMA IF NOT EXISTS SG;

USE SG;

DROP TABLE IF EXISTS s\_customer;

CREATE TABLE s\_customer

 (id                         VARCHAR(3)  NOT NULL,

  name                       VARCHAR(20) NOT NULL,

  phone                      VARCHAR(20) NOT NULL,

  address                    VARCHAR(20),

  city                       VARCHAR(20),

  state                      VARCHAR(15),

  country                    VARCHAR(20),

  zip\_code                   VARCHAR(15),

  credit\_rating              VARCHAR(9),

  sales\_rep\_id               VARCHAR(3),

  region\_id                  VARCHAR(3),

  comments                   VARCHAR(255),

  CONSTRAINT s\_customer\_id\_pk PRIMARY KEY (id),

  CONSTRAINT s\_customer\_credit\_rating\_ck

  CHECK (credit\_rating IN ('EXCELLENT', 'GOOD', 'POOR'))

 );

INSERT INTO s\_customer VALUES ('301', 'Sports,Inc', '540-123-4567','72 High St',

'Harrisonburg', 'VA','US', '22809','EXCELLENT', '12', '1', NULL);

INSERT INTO s\_customer VALUES ('302', 'Toms Sporting Goods', '540-987-6543','6741 Main St',

'Harrisonburg', 'VA','US', '22809','POOR', '14', '1', NULL);

INSERT INTO s\_customer VALUES ('303', 'Athletic Attire', '540-123-6789','54 Market St',

'Harrisonburg', 'VA','US', '22808','GOOD', '14', '1', NULL);

INSERT INTO s\_customer

VALUES ('304', 'Athletics For All', '540-987-1234','286 Main St', 'Harrisonburg', 'VA',

'US', '22808','EXCELLENT', '12', '1', NULL);

INSERT INTO s\_customer VALUES ('305', 'Shoes for Sports', '540-123-9876','538 High St',

'Harrisonburg', 'VA','US', '22809','EXCELLENT', '14', '1', NULL);

INSERT INTO s\_customer VALUES ('306', 'BJ Athletics', '540-987-9999','632 Water St',

'Harrisonburg', 'VA','US', '22810','POOR', '12', '1', NULL);

INSERT INTO s\_customer VALUES ('403', 'Athletics One', '717-234-6786','912 Columbia Rd',

'Lancaster', 'PA','US', '17601','GOOD', '14', '1', NULL);

INSERT INTO s\_customer VALUES ('404', 'Great Athletes', '717-987-2341','121 Litiz Pike',

'Lancaster', 'PA','US', '17602','EXCELLENT', '12', '1', NULL);

INSERT INTO s\_customer VALUES ('405', 'Athletics Two', '717-987-9875','435 High Rd',

'Lancaster', 'PA','US', '17602','EXCELLENT', '14', '1', NULL);

INSERT INTO s\_customer VALUES ('406', 'Athletes Attic', '717-234-9888','101 Greenfield Rd',

'Lancaster', 'PA','US', '17601','POOR', '12', '1', NULL);

INSERT INTO s\_customer VALUES ('201', 'One Sport', '55-112066222','82 Via Bahia', 'Sao Paolo',

NULL, 'Brazil', NULL,'EXCELLENT', '12', '2', NULL);

INSERT INTO s\_customer VALUES ('202', 'Deportivo Caracas', '58-28066222','31 Sabana Grande',

'Caracas', NULL, 'Venezuela', NULL,'EXCELLENT', '12', '2', NULL);

INSERT INTO s\_customer VALUES ('203', 'New Delhi Sports', '91-11903338','11368 Chanakya',

'New Delhi', NULL, 'India', NULL,'GOOD', '11', '4', NULL);

INSERT INTO s\_customer VALUES ('204', 'Ladysport', '1-206-104-0111','281 Queen Street',

'Seattle', 'Washington', 'US', NULL,'EXCELLENT', '11', '1', NULL);

INSERT INTO s\_customer VALUES ('205', 'Kim''s Sporting Goods', '852-3693888','15 Henessey Road',

'Hong Kong', NULL, NULL, NULL,'EXCELLENT', '11', '4', NULL);

INSERT INTO s\_customer VALUES ('206', 'Sportique', '33-93425722253','172 Rue de Place',

'Cannes', NULL, 'France', NULL,'EXCELLENT', '13', '5', NULL);

INSERT INTO s\_customer VALUES ('207', 'Tall Rock Sports', '234-16036222','10 Saint Antoine',

'Lagos', NULL, 'Nigeria', NULL,'GOOD', NULL, '3', NULL);

INSERT INTO s\_customer VALUES ('208', 'Muench Sports', '49-895274449','435 Gruenestrasse',

'Munich', NULL, 'Germany', NULL,'GOOD', '13', '5', NULL);

INSERT INTO s\_customer VALUES ('209', 'Beisbol Si!', '809-352666','415 Playa Del Mar',

 'San Pedro de Macoris', NULL, 'Dominican Republic', NULL, 'EXCELLENT', '11', '6', NULL);

INSERT INTO s\_customer VALUES ('210', 'Futbol Sonora', '52-404555','5 Via Saguaro', 'Nogales',

NULL, 'Mexico', NULL,'EXCELLENT', '12', '2', NULL);

INSERT INTO s\_customer VALUES ('211', 'Helmut''s Sports', '42-2111222','45 Modrany', 'Prague',

NULL, 'Czechoslovakia', NULL,'EXCELLENT', '11', '5', NULL);

INSERT INTO s\_customer VALUES ('212', 'Hamada Sport', '20-31209222','47A Corniche',

'Alexandria', NULL, 'Egypt', NULL,'EXCELLENT', '13', '3', NULL);

INSERT INTO s\_customer VALUES ('213', 'Sports Emporium', '1-415-555-6281','4783 168th Street',

'San Francisco', 'CA', 'US', NULL,'EXCELLENT', '11', '1', NULL);

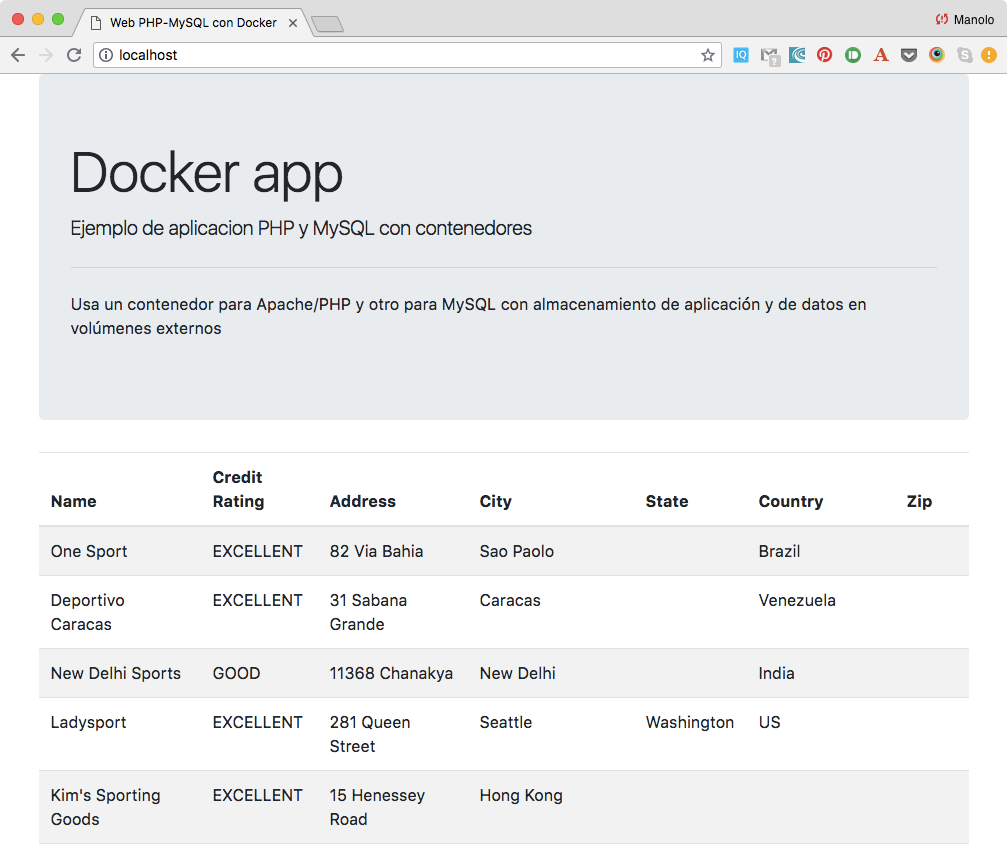
INSERT INTO s\_customer VALUES ('214', 'Sports Retail', '1-716-555-7777','115 Main Street',

'Buffalo', 'NY', 'US', NULL, 'POOR', '11', '1', NULL);

INSERT INTO s\_customer VALUES ('215', 'Sports Russia', '7-0953892444','7070 Yekatamina',

'Saint Petersburg', NULL, 'Russia', NULL,'POOR', '11', '5', NULL);

COMMIT;



*Figure 10. Aplicación web PHP que muestra listado de clientes almacenados en MySQL*

#### Archivo .dockerignore

Todavía no sé todos los trucos para optimizar el tamaño de las imágenes en Docker. Sin embargo, he utilizado .dockerignore para omitir algunas carpetas y aligerar la carga.

Principio del formulario