Proyectos con path public, public_html, app en Docker con Apache+PHP-FPM

Introducción

Existen desarrollos en software web, que siguen una pauta moderna y más segura, consistente en exponer solamente al público el fichero index.php dejando en el `doc_root` de **Apache** o **Nginx** el directorio que lo contiene, y no todos el desarrollo. Ejemplos clásicos son, Laravel, Symfony, Slim, y otros muchos. aunque si no eres amigo de los marcos de trabajo (framework), bueno es que también uses esta formula de trabajo y no meter todo en el directorio del `doc_root` del servidor Web.

En mi último trabajo he tenido que lidiar con una aplicación obsoleta, en PHP 5.6 desarrollada con el marco Zend Framework, que estaba en un cPanel pero con las estructura correcta por debajo del `public_html`. (Lastima que encima puso un blog con Wordpress, y se le olvido actualizarlo)

Esta guía te mostrara el camino de la creación de los contenedores necesarios para Apache, PHP-Fpm y MySQL para ejecutar tu desarrollo, usando **Docker compose**

Docker Apache, PHP FPM setup with docker compose y MySQL

Cuando uno llega aquí, lo único que queda es dockerizar nuestro area de trabajo, ya que desarrollar en el servidor está... **prohibido**

El proceso esta indicado para un escenario concreto, pero con los ajustes necesarios, puedes elegir la versión de PHP, la de MySQL/MariaDb/Percona que necesites (siempre trabaja con el mismo escenario que en producción).

Requisitos

Necesitamos tener instalado en nuestro equipo Docker.

Generaremos la siguiente estructura en nuestro proyecto

```
proyecto
├── Dockerfile
├── docker-compose.yml
├── apache/
│ │ ├── apache.vhost.conf
│ │ └── Dockerfile
├── public/
│    └── index.php
├── vendor/
├── ...
```

Algo así quedaría

Image not found or type unknown

- **Dockerfile** global que contiene la configuración necesaria para crear y preparar el escenario para un PHP con PHP-FPM y sus extensiones necesarias en las imágenes necesarias.
- **docker-compose.yml** que crear los contenedores necesarios y los unirá.
- **apache/apache.vhost.conf** que contendrá la información necesaria para que Apache sea el servidor web de nuestro proyecto.
- **apache/Dockerfile**
- **public** que será el directorio root de nuestro proyecto, expuesto a internet por el par Apache/PHP-FPM

Crear la imagen PHP FPM

Te damos las instrucciones necesarias a docker para construir la imagen. El ejemplo es el básico que me permite usar y trabajar en esta dockerización, con el proyecto Zend en un PHP 5.6. Si necesitas más extensiones, o algún cambio necesitaras un trabajo extra para añadirlo.

Consulta siempre en Hub Docker las posibilidad que tienes, y si quieres un consejo, usa siempre imágenes oficiales. Te ahorraras muchos disgustos.

> Los comentarios indican modificación sobre la imagen oficial. Si usas una no oficial, o versiones oficiales basadas en otro so, como alpine, etc. los paths indicados pueden sufrir variaciones

```
FROM php:5.6-fpm

RUN docker-php-ext-install mysqli pdo pdo_mysql

## Esta linea la puedes comentar si tu proyecto tiene un estructura plana, como puede ser un Wordpress.
Donde todo está en el mismo nivel del doc_root
RUN echo 'php_admin_value[doc_root] = /var/www/html/public' >> /usr/local/etc/php-fpm.d/www.conf
```

- Docker hará un pull **php:5.6-fpm**
- Si quieres o necesitas instalar una extensión adicional con Docker usa el commando **docker-php-ext-install** dentro de doc usando **RUN**

Crear Apache

Necesitamos crear la imagen con el servidor web Apache, atendiendo a que usaremos PHP-FPM como manejador de PHP. Así que usaremos un fichero de configuración del host virtual, y usaremos **apache/apache.vhost.conf** para este propósito.

```
# La configuracion esta adaptada al escenario indicado public/
# Si no es así deberá modificarse al típico /var/www/html/ allí donde este usándose public/
# Establezca el nombre del servidor en localhost
ServerName localhost

# Configure un VirtualHost para manejar solicitudes en el puerto 80
<VirtualHost *:80>
    # Solicitudes de proxy PHP para portar el contenedor PHP-FPM 9000
    ProxyPassMatch ^/(.*\.php(/.*)?)$ fcgi://php-fpm:9000/var/www/html/public/$1

    # Set the DocumentRoot for the virtual host
    DocumentRoot /var/www/html/public

    # Directory configuration for the DocumentRoot
    <Directory /var/www/html/public>
        DirectoryIndex index.php
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    # Definir los destinos CustomLog y ErrorLog
    CustomLog /proc/self/fd/1 common
```

```
    ErrorLog /proc/self/fd/2
</VirtualHost>
```

- La aplicación se ejecutara en modo localhost y responderá a las peticiones que se hagan la puerto 80.
- Apache reenviará las peticiones php al container PHP-FPM, cuyo nombre es 'php-fpm'. Este nombre será usado en el `docker-compose.yml` del proyecto.
- `/var/www/html/public/` será el directorio por defecto de tu sitio virtual de Apache. Por tanto sería el index o fichero que este allí.
- Si deseas depurar tu aplicación, cree destinos CustomLog y ErrorLog para administrar los registros de la aplicación y los de Docker.

Ahora construiremos nuestro **Dockerfile** para construir la imagen de nuestro servidor web Apache

```
# httpd base image
FROM httpd:2.4

# Copy the Apache virtual host configuration file to the container
COPY ./apache/apache.vhost.conf /usr/local/apache2/conf/extra/apache.vhost.conf

# Activar módulos de Apache para garantizar una funcionalidad adecuada.
RUN sed -i \
    # Descomente la configuración de mod_rewrite para habilitar el control de reescritura contenido.
    -e '/#LoadModule rewrite_module/s/^#//g' \
    # Descomente la configuración de mod_rewrite para habilitar el control de caducidad del contenido.
    -e '/#LoadModule expires_module/s/^#//g' \
    # Descomente la configuración de mod_deflate para habilitar la compreión
    -e '/#LoadModule deflate_module/s/^#//g' \
    # Descomente la configuración del poryx para habilitar su uso.
    -e '/#LoadModule proxy_module/s/^#//g' \
    # Descomente la configuración de mod_proxy_fcgi para habilitar el módulo proxy FastCGI
    -e '/#LoadModule proxy_fcgi_module/s/^#//g' \
    /usr/local/apache2/conf/httpd.conf

# Incluir el archivo de configuración del host virtual en la configuración principal de Apache.
RUN echo "Include /usr/local/apache2/conf/extra/apache.vhost.conf" >> /usr/local/apache2/conf/httpd.conf
```

Crear el container

Bien es hora de armarlo todo.

> En mi caso, no uso PHPMyAdmin así que no instalaremos nada adicional. Nunca lo he usado porque nunca me ha parecido correcto tenerlo instalado. En su lugar uso un programa de consola, y me conecto via tunnel SSH. Pero os dejo comentada la opción. Gracias TablePlus

```
version: '3.9'
services:
  php-fpm:
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - .:/var/www/html
    depends_on:
      - mysql-db

  apache-httpd:
    build:
      context: .
      dockerfile: ./apache/Dockerfile
    volumes:
      - .:/var/www/html
    ports:
      - "8080:80"
    depends_on:
      - php-fpm
      - mysql-db
  mysql-db:
    image: mariadb:10.1
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: base_de_datos
      MYSQL_USER: user_datos
      MYSQL_PASSWORD: contraseña
    ports:
      - "3336:3306"
    volumes:
      - mysql-data:/var/lib/mysql
# Mac not working
```

```
#  phpmyadmin:
#    image: phpmyadmin/phpmyadmin:latest
#    links:
#      - mysql-db
#    ports:
#      - "8081:80"
#    environment:
#      PMA_HOST: mysql-db
#      MYSQL_ROOT_PASSWORD: password
volumes:
  mysql-data:
```

- `.:/var/www/html` esto indica que crearemos un **volumen** que montara en la raíz de nuestro proyecto `.` en `/var/www/html/` en el container.
- El puerto expuesto para acceder será el 8080
- Crearemos un container para **MySQL** que tendrá persistencia de datos, en un volumen **mysql-data** que permitirá tener persistencia de datos.
- Mi consejo es que uses los mismo datos para el usuario, base de datos, y contraseña relativos a tu proyecto, que los que usas en producción, por si realizas sincronizaciones. El root no lo toques.
- El puerto expuesto es el 3336 ya que tengo en local, una instalación **Herd** que es la que uso habitualmente y no me apetece apagarla.

Testing

Para evitar perdidas de tiempo, es bueno hacer dos comprobaciones.

- Que el container esta preparado para trabajar con PHP en el directorio `public`
- Que la conexión con Mysql funciona.

info.php

Crea un fichero `public/info.php`

```
<?php phpinfo();
```

index.php

Si ya tienes uno renombrarlo, para más tarde y crea un nuevo en `public/index.php`

> También pues añadir al código al principio del index.php y salir con un `exit();`

```
<?php
$host = 'mysql-db';
$user = 'root';
$pass = 'password';
$db = 'mysql';



$conn = new mysqli($host, $user, $pass, $db);

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

echo "Connected to MySQL successfully";

$conn->close();
?>
```

Construir los contenedores

Bueno, ya esta todo.

```
docker-compose up --build -d
docker-compose up --build -d
[+] Building 2.7s (16/16)
FINISHED
              docker:desktop-linux
 => [php-fpm internal] load build definition from
Dockerfile
     0.0s
 => => transferring dockerfile:
204B
                      0.0s
 => [php-fpm internal] load metadata for docker.io/library/php:5.6-
fpm                                                                                                   1.6s
 => [php-fpm internal] load
.dockerignore
                              0.0s
 => => transferring context:
```

2B
0.0s
=> [php-fpm 1/3] FROM docker.io/library/php:5.6-
fpm@sha256:4f070f1b7b93cc5ab364839b79a5b26f38d5f89461f7bc0cd4bab4a3ad7d67d7
0.0s
=> CACHED [php-fpm 2/3] RUN docker-php-ext-install mysqli pdo
pdo_mysql
0.0s
=> CACHED [php-fpm 3/3] RUN echo 'php_admin_value[doc_root] = /var/www/html/public' >>
/usr/local/etc/php-fpm.d/www.conf 0.0s
=> [php-fpm] exporting to
image
0.0s
=> => exporting
layers
0.0s
=> => writing image
sha256:552b673bf2989a93258340ee86ae7c31477c946f95aa73a63af49cede70235a6
0.0s
=> => naming to docker.io/library/staybarcelona-php-
fpm
0.0s
=> [apache-httpd internal] load build definition from
Dockerfile
0.0s
=> => transferring dockerfile:
1.17kB
0.0s
=> [apache-httpd internal] load metadata for
docker.io/library/httpd:2.4
0.9s
=> [apache-httpd internal] load
.dockerignore
0.0s
=> => transferring context:
2B
0.0s
=> [apache-httpd internal] load build
context

```
              0.0s
 => => transferring context:
72B
                     0.0s
 => [apache-httpd 1/4] FROM
docker.io/library/httpd:2.4@sha256:104f07de17ee186c8f37b9f561e04fbfe4cf080d78c6e5f3802fd08fd118c3da
                                                                  0.0s
 => CACHED [apache-httpd 2/4] COPY ./apache/apache.vhost.conf
/usr/local/apache2/conf/extra/apache.vhost.conf
              0.0s
 => CACHED [apache-httpd 3/4] RUN sed -i             -e '/#LoadModule rewrite_module/s/^#//g'         -e
'/#LoadModule expires_module/s/^#//g'     -e '/#LoadModule deflate_module/s/^#//g'     -e '/#LoadModule
proxy_module/s/^#  0.0s
 => CACHED [apache-httpd 4/4] RUN echo "Include /usr/local/apache2/conf/extra/apache.vhost.conf" >>
/usr/local/apache2/conf/httpd.conf                                                  0.0s
 => [apache-httpd] exporting to
image
                   0.0s
 => => exporting
layers
                            0.0s
 => => writing image
sha256:7428ace356414dd78d15c185ffa4137e7e3806018ce31eac61be778c773738f5
                                                          0.0s
 => => naming to docker.io/library/staybarcelona-apache-
httpd
0.0s
[+] Running 3/3
 ✔ Container staybarcelona-mysql-db-1
Started
        0.0s
 ✔ Container staybarcelona-php-fpm-1
Started
        0.6s
 ✔ Container staybarcelona-apache-httpd-1  Started
```

Comprobaciones

En el navegador probaremos la información del `phpinfo()` en la ruta `localhost:8080`

Info
Image not found or type unknown

Ahora la conexión **mySQL**

Info
Image not found or type unknown

Conclusión

Bien, esto es un esbozo de como podemos adecuarnos con Docker Composer, en un escenario como el descrito además de aprender ciertas metodologías con Docker.

Esto es muy básico y dependerá de la instalación y trabajos que tienes en tu máquina. Hay muchos caminos y formas de trabajar con Docker. Esta es una rápida para un escenario particular.

Aviso

Esta documentación y su contenido, no implica que funcione en tu caso o determinados casos. También implica que tienes conocimientos sobre lo que trata, y que en cualquier caso tienes copias de seguridad. El contenido el contenido se entrega, tal y como está, sin que ello implique ningún obligación ni responsabilidad por parte de Castris

Si necesitas soporte profesional puedes contratar con Castris soporte profesional.

Revision #1
Created 3 March 2024 09:49:33 by Abkrim
Updated 3 March 2024 09:50:37 by Abkrim