ALTIA TECH DAY

Taller de Docker: Desarrollando entre contenedores

altia

ALTIA TECH DAY

Taller de Docker: Desarrollando entre contenedores

Altia **Senior Developer**
Disfrutando del desarrollo web desde 1998.

@rolando_caldas

https://altia.es

https://rolandocaldas.com

ALTIA TECH DAY

altia

Instalar Docker

Instalar Docker

- **Ubuntu**: https://docs.docker.com/install/linux/docker-ce/ubuntu/
- **Windows**:
  https://store.docker.com/editions/community/docker-ce-desktop-windows
- **Mac**:
  https://store.docker.com/editions/community/docker-ce-desktop-mac
- **Post-Install**:
  - **Ubuntu**: Lanzar docker sin ser root + Activar acceso remoto (IDE)
  - **Windows**: Activar acceso remoto + Permitir conexiones no seguras
  - **Mac**: Activar acceso remoto

Ejecutar docker sin ser root

- Es necesario crear el grupo docker y asociarlo al usuario con el que se va a utilizar docker:

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
```

- Logout/Login
- GO GO GO

ALTIA TECH DAY

altia

Docker no es un entorno virtual

Docker no es un entorno virtual

- Docker utiliza el Namespace y el Cgroups de Linux.

Docker no es un entorno virtual

- Docker utiliza el Namespace y el Cgroups de Linux.
- Como en programación, un namespace es un espacio aislado.

Docker no es un entorno virtual

- Docker utiliza el Namespace y el Cgroups de Linux.
- Como en programación, un namespace es un espacio aislado.
- Cgroups: Es una funcionalidad de linux para limitar los recursos que puede usar una aplicación.

Docker no es un entorno virtual

- Docker utiliza el Namespace y el Cgroups de Linux.
- Como en programación, un namespace es un espacio aislado.
- Cgroups: Es una funcionalidad de linux para limitar los recursos que puede usar una aplicación.
- Cuando lanzas un contenedor de Docker, éste crea un namespace en linux y crea una capa de aislamiento para el contenedor.

Docker no es un entorno virtual

- Docker utiliza el Namespace y el Cgroups de Linux.
- Como en programación, un namespace es un espacio aislado.
- Cgroups: Es una funcionalidad de linux para limitar los recursos que puede usar una aplicación.
- Cuando lanzas un contenedor de Docker, éste crea un namespace en linux y crea una capa de aislamiento para el contenedor.
- Todo lo que ocurra en el contenedor está limitado a su namespace.

Docker no es un entorno virtual

- Docker utiliza el Namespace y el Cgroups de Linux.
- Como en programación, un namespace es un espacio aislado.
- Cgroups: Es una funcionalidad de linux para limitar los recursos que puede usar una aplicación.
- Cuando lanzas un contenedor de Docker, éste crea un namespace en linux y crea una capa de aislamiento para el contenedor.
- Todo lo que ocurra en el contenedor está limitado a su namespace.
- Docker utiliza los Cgroups para limitar los recursos que pueden consumir los contenedores.

Docker no es un entorno virtual

En resumen: Lo que se ejecuta en el contenedor ...

Docker no es un entorno virtual

En resumen: Lo que se ejecuta en el contenedor ...

… se ejecuta en nuestro sistema, no en una máquina virtual.

ALTIA TECH DAY

Workshop timeline

Workshop timeline

https://github.com/rolando-caldas/altia-tech-day-docker/blob/master/README.md

ALTIA TECH DAY

altia

Comandos básicos

Comandos básicos

`$ docker pull`	Descarga una imagen de la red
`$ docker build`	Crea una imagen
`$ docker run`	Lanza un contenedor
`$ docker stop`	Para un contenedor
`$ docker rm`	Elimina un contenedor
`$ docker rmi`	Elimina un imagen
`$ docker ps`	Contenedores activos
`$ docker ps -a`	Contenedores activos e inactivos
`$ docker images`	Imágenes descargadas

ALTIA TECH DAY

altia

Corriendo nuestro primer contenedor

Corriendo nuestro primer contenedor

```
$ cd $HOME && mkdir dockerWorkshop && cd dockerWorkshop
$ docker run hello-world
```

- Docker intenta ejecutar la imagen hello-world

Corriendo nuestro primer contenedor

```
$ cd $HOME && mkdir dockerWorkshop && cd dockerWorkshop
$ docker run hello-world
```

- Docker intenta ejecutar la imagen hello-world
- Como no existe, porque no la hemos descargado (sería haciendo el pull) va a buscarla a la red

Corriendo nuestro primer contenedor

```
$ cd $HOME && mkdir dockerWorkshop && cd dockerWorkshop
$ docker run hello-world
```

- Docker intenta ejecutar la imagen hello-world
- Como no existe, porque no la hemos descargado (sería haciendo el pull) va a buscarla a la red
- La busca, por defecto, en el docker hub. Como la encuentra, la descarga (hace el pull por nosotros) y tras ello, hace el run.

Corriendo nuestro primer contenedor

```
$ cd $HOME && mkdir dockerWorkshop && cd dockerWorkshop
$ docker run hello-world
```

- Docker intenta ejecutar la imagen hello-world
- Como no existe, porque no la hemos descargado (sería haciendo el pull) va a buscarla a la red
- La busca, por defecto, en el docker hub. Como la encuentra, la descarga (hace el pull por nosotros) y tras ello, hace el run.
- Si lanzamos `docker ps` no sale nada ¿por que? Porque el contenedor se lanzó, se ejecutó y, al no tener ningún proceso o servicio que lo mantenga abierto, se cerró.

Corriendo nuestro primer contenedor

```
$ cd $HOME && mkdir dockerWorkshop && cd dockerWorkshop
$ docker run hello-world
```

- Docker intenta ejecutar la imagen hello-world
- Como no existe, porque no la hemos descargado (sería haciendo el pull) va a buscarla a la red
- La busca, por defecto, en el docker hub. Como la encuentra, la descarga (hace el pull por nosotros) y tras ello, hace el run.
- Si lanzamos `docker ps` no sale nada ¿por que? Porque el contenedor se lanzó, se ejecutó y, al no tener ningún proceso o servicio que lo mantenga abierto, se cerró.
- Si lanzamos docker ps -a veremos nuestro contenedor.

Corriendo nuestro primer contenedor

```
$ cd $HOME && mkdir dockerWorkshop && cd dockerWorkshop
$ docker run hello-world
```

- Docker intenta ejecutar la imagen hello-world
- Como no existe, porque no la hemos descargado (sería haciendo el pull) va a buscarla a la red
- La busca, por defecto, en el docker hub. Como la encuentra, la descarga (hace el pull por nosotros) y tras ello, hace el run.
- Si lanzamos `docker ps` no sale nada ¿por que? Porque el contenedor se lanzó, se ejecutó y, al no tener ningún proceso o servicio que lo mantenga abierto, se cerró.
- Si lanzamos docker ps -a veremos nuestro contenedor.
- Si lanzamos docker image veremos la imagen descargada y su huella.

Corriendo nuestro primer contenedor

```
$ docker run ubuntu
```

ALTIA TECH DAY

Corriendo nuestro primer contenedor

```
$ docker run ubuntu
$ docker ps
```

ALTIA TECH DAY

Corriendo nuestro primer contenedor

```
$ docker run ubuntu
$ docker ps
$ docker ps -a
```

ALTIA TECH DAY

Corriendo nuestro primer contenedor

```
$ docker run ubuntu
$ docker ps
$ docker ps -a
$ docker images
```

Corriendo nuestro primer contenedor

```
$ docker run ubuntu
$ docker ps
$ docker ps -a
$ docker images
$ docker run -it ubuntu
```

- -i : Mantiene abierta la entrada estándar (STDIN) para poder interactuar desde fuera.
- -t: Abre una terminal (TTY)
- ¿por qué una consola bash? => Porque es el CMD de la imagen ubuntu

Corriendo nuestro primer contenedor

```
$ docker run ubuntu
$ docker ps
$ docker ps -a
$ docker images
$ docker run -it ubuntu
```

- -i : Mantiene abierta la entrada estándar (STDIN) para poder interactuar desde fuera.
- -t: Abre una terminal (TTY)
- ¿por qué una consola bash? => Porque es el CMD de la imagen ubuntu

```
$ docker run -it node
```

- Consola de node porque el CMD de la imagen es NODE, no BASH

Corriendo nuestro primer contenedor

```
$ docker run ubuntu
$ docker ps
$ docker ps -a
$ docker images
$ docker run -it ubuntu
```

- -i : Mantiene abierta la entrada estándar (STDIN) para poder interactuar desde fuera.
- -t: Abre una terminal (TTY)
- ¿por qué una consola bash? => Porque es el CMD de la imagen ubuntu

```
$ docker run -it node
```

- Consola de node porque el CMD de la imagen es NODE, no BASH

```
$ docker ps
```

Ya tenemos cosas!!!

Recordatorio: Docker no es una VM

Con nuestro contenedor de ubuntu corriendo, vamos a su shell y escribimos:

```
root@f709b1535b47:/# top
```

Vamos a un terminal de nuestro linux (no del contenedor):

```
$ ps afux
```

```
root      3322 /usr/bin/dockerd -H fd://
root      3485  \_ docker-containerd --config /var/run/docker/containerd/containerd.toml
root     14708      \_ docker-containerd-shim -namespace moby -workdir /var/lib/docker/containerd/daemon/io.containerd.runtime.v1.linux/moby/f709b1535b477
root     14726          \_ bash
root     16877              \_ top
root      3323 /bin/sh /snap/nextcloud/8971/bin/renew-certs
```

ALTIA TECH DAY

altia

Dockerizando PHP

Dockerizando PHP

En el hub de docker existen múltiples imágenes oficiales, PHP es una de ellas:

- Entramos en https://hub.docker.com/
- Buscamos PHP: https://hub.docker.com/ /php/
- Tenemos múltiples imágenes de PHP disponibles, tanto en base a la versión como al SO base, PHP + Apache, PHP-CLI, PHP-FPM, ZTS…
- Vamos a utilizar la versión 7.2-fpm

```
$ docker run --name=php-fpm php:7.2-fpm
```

- con --name=php-fpm forzamos que nuestro contenedor se llame php-fpm… esto será muy útil cuando necesitemos interactuar con el contenedor.

PHP Dockerizado running!

Podemos ejecutar comandos en nuestros contenedores desde fuera con:

```
docker exec [container] [command]
```

PHP Dockerizado running!

Podemos ejecutar comandos en nuestros contenedores desde fuera con:

```
docker exec [container] [command]
```

Para ver qué versión de PHP está en nuestro container:

```
$ docker exec php-fpm php -v
```

Docker PHP + Apache2

Aunque pueda ser útil normalmente de poco o nada sirve tener un contenedor con PHP… necesitamos al menos un servidor web para acceder por HTTP.

De forma oficial, tenemos la imagen con Apache2:

```
$ docker run -p 8080:80 -it --name=php-apache php:7.2-apache
```

Docker PHP + Apache2

Aunque pueda ser útil normalmente de poco o nada sirve tener un contenedor con PHP… necesitamos al menos un servidor web para acceder por HTTP.

De forma oficial, tenemos la imagen con Apache2:

```
$ docker run -p 8080:80 -it --name=php-apache php:7.2-apache
```

Docker PHP + Apache2

Docker ejecuta los contenedores de forma aislada, por lo que el Apache del contenedor no estará accesible desde el exterior. Necesitamos solicitar a Docker que puentee un puerto de nuestra máquina con el contenedor:

- -p [puerto-local]:[puerto-contenedor]

Si accedemos con nuestro navegador a http://localhost:8080 veremos un error de Apache, porque aunque el servidor web funciona y la conexión entre puertos también, el contenedor no tiene contenido a mostrar.

Docker PHP + Apache2

```
$ docker exec -it php-apache bash
root@be8415f43808:/var/www/html# apt update && apt install vim
root@be8415f43808:/var/www/html# vim index.php
```

- escribimos el código para el phpinfo: <?php phpinfo();
- salimos con :wq

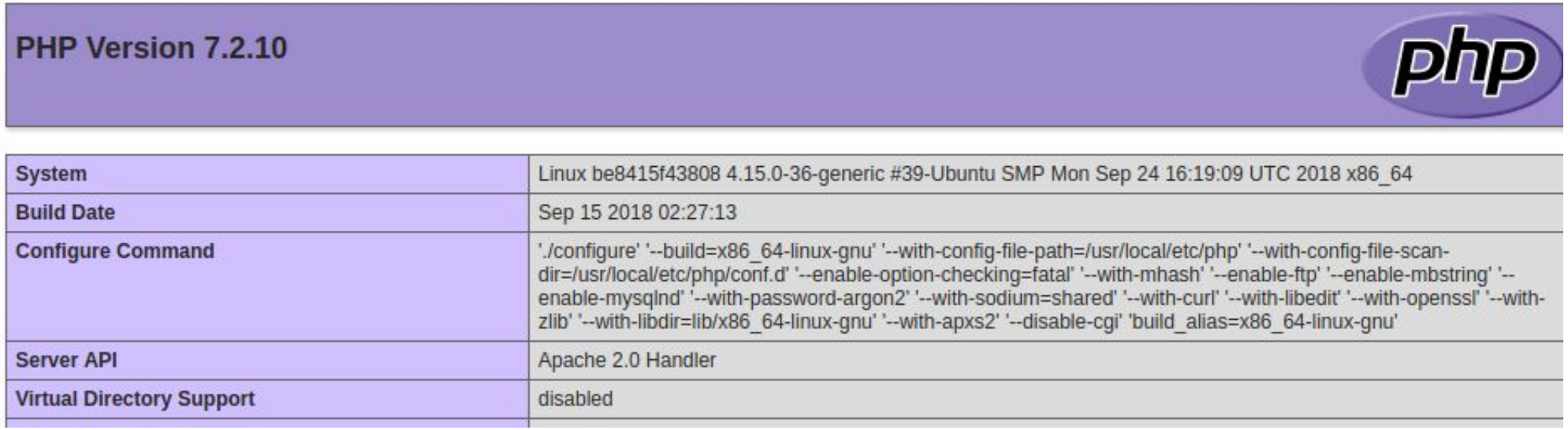
Ahora si accedemos a http://localhost:8080 …

Docker PHP + Apache2

```
$ docker exec -it php-apache bash
root@be8415f43808:/var/www/html# apt update && apt install vim
root@be8415f43808:/var/www/html# vim index.php
```

- escribimos el código para el phpinfo: <?php phpinfo();
- salimos con :wq

Ahora si accedemos a http://localhost:8080 …



PHP Version 7.2.10

System	Linux be8415f43808 4.15.0-36-generic #39-Ubuntu SMP Mon Sep 24 16:19:09 UTC 2018 x86_64
Build Date	Sep 15 2018 02:27:13
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--with-libdir=lib/x86_64-linux-gnu' '--with-apxs2' '--disable-cgi' 'build_alias=x86_64-linux-gnu'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled

Levantar contenedores existentes

Cuando paramos un contenedor ya sea con docker stop o con “CTRL+C” el contenedor no se elimina sino que se para. Si hacemos un run:

```
$ docker run -it --name=php-fpm php:7.2-fpm
```

Docker creará el contenedor y lo levantará. Como le hemos especificado el nombre del contenedor y éste ya existe tenemos un error.

En estos casos lo que necesitamos es levantar el contenedor, no crear uno nuevo:

```
$ docker start php-fpm
```

ALTIA TECH DAY

altia

Dockerfile: Extendiendo el contenedor de PHP

Añadiendo extensiones PHP y utilidades

- Las imágenes de PHP, como todas las imágenes oficiales en Docker Hub, van con lo mínimo.

Añadiendo extensiones PHP y utilidades

- Las imágenes de PHP, como todas las imágenes oficiales en Docker Hub, van con lo mínimo.
- Programas habituales como vi, vim, nano, ping, etc… NO están en nuestro contenedor.

Añadiendo extensiones PHP y utilidades

- Las imágenes de PHP, como todas las imágenes oficiales en Docker Hub, van con lo mínimo.
- Programas habituales como vi, vim, nano, ping, etc… NO están en nuestro contenedor.
- Podríamos entrar al contenedor por bash y tirar de apt-get etc etc, pero NO debemos hacer eso.

Añadiendo extensiones PHP y utilidades

- Las imágenes de PHP, como todas las imágenes oficiales en Docker Hub, van con lo mínimo.
- Programas habituales como vi, vim, nano, ping, etc… NO están en nuestro contenedor.
- Podríamos entrar al contenedor por bash y tirar de apt-get etc etc, pero NO debemos hacer eso.
- La idea de un contenedor es que cuando se ejecuta con el run, realice todas las acciones necesarias y sólo las justas minimizando el espacio que necesita.

Añadiendo extensiones PHP y utilidades

- Las imágenes de PHP, como todas las imágenes oficiales en Docker Hub, van con lo mínimo.
- Programas habituales como vi, vim, nano, ping, etc… NO están en nuestro contenedor.
- Podríamos entrar al contenedor por bash y tirar de apt-get etc etc, pero NO debemos hacer eso.
- La idea de un contenedor es que cuando se ejecuta con el run, realice todas las acciones necesarias y sólo las justas minimizando el espacio que necesita.
- Para realizar todas estas acciones tenemos los Dockerfile.

Añadiendo extensiones PHP y utilidades

Para un entorno base, válido para desarrollo y producción deberíamos contar con:

- **php-intl**: Funciones de internacionalización.
- **php-gd**: Graphics draw. Funciones de tratamiento de imágenes
- **php-mbstring**: Cadenas de caracteres multibyte
- **composer**: Gestor de paquetes… el APT de PHP.

Para un entorno de desarrollo:

- **xDebug**: Debugger y Profiler para PHP
- **phpUnit**: Test unitarios en PHP + Cobertura de código (xDebug requerido)
- **BlackFire**: Análisis de rendimiento (recomendable sin xDebug)

Añadiendo extensiones PHP y utilidades

Gracias a utilizar imágenes base de Docker oficiales, tendremos extras interesantes. En el caso de PHP tenemos:

- **docker-php-ext-configure**: Permite configurar fácilmente extensiones de PHP
- **docker-php-ext-install**: Permite instalar fácilmente extensiones de PHP

Personalizar la imagen de PHP: Dockerfile

El Dockerfile es un fichero utilizado por Docker para crear imágenes, a partir de “nada” o de otras imágenes existentes:

Personalizar la imagen de PHP: Dockerfile

El Dockerfile es un fichero utilizado por Docker para crear imágenes, a partir de “nada” o de otras imágenes existentes:

- Se basa en una ejecución secuencial de comandos.

Personalizar la imagen de PHP: Dockerfile

El Dockerfile es un fichero utilizado por Docker para crear imágenes, a partir de “nada” o de otras imágenes existentes:

- Se basa en una ejecución secuencial de comandos.
- Cada comando ejecutado sería como realizar un commit en un repositorio.

Personalizar la imagen de PHP: Dockerfile

El Dockerfile es un fichero utilizado por Docker para crear imágenes, a partir de “nada” o de otras imágenes existentes:

- Se basa en una ejecución secuencial de comandos.
- Cada comando ejecutado sería como realizar un commit en un repositorio.
- Cada “commit” se guarda en caché, para no ejecutarlo nuevamente al reconstruir la imagen.

Personalizar la imagen de PHP: Dockerfile

El Dockerfile es un fichero utilizado por Docker para crear imágenes, a partir de “nada” o de otras imágenes existentes:

- Se basa en una ejecución secuencial de comandos.
- Cada comando ejecutado sería como realizar un commit en un repositorio.
- Cada “commit” se guarda en caché, para no ejecutarlo nuevamente al reconstruir la imagen.
- Siempre usará la caché, hasta que exista un cambio en un comando, en este caso, usa la caché hasta ese comando y, a partir de ahí, no.

Personalizar la imagen de PHP: Dockerfile

Creamos un directorio php-base:

```
$ mkdir docker && cd docker
$ mkdir php-base && cd php-base
```

Creamos un fichero Dockerfile:

```
$ touch Dockerfile
```

Lo abrimos con nuestro editor favorito:

```
$ vim Dockerfile
```

Personalizar la imagen de PHP: Dockerfile

```
FROM php:7.2-fpm

WORKDIR "/application"
RUN apt-get update \
    && apt-get install -y libicu-dev libpng-dev libjpeg-dev libpq-dev  \
    && apt-get clean; rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
/usr/share/doc/* \
    && docker-php-ext-configure gd --with-png-dir=/usr --with-jpeg-dir=/usr \
    && docker-php-ext-install intl gd mbstring
RUN apt-get update && apt-get install -y mysql-client \
    && apt-get clean; rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
/usr/share/doc/* \
    && docker-php-ext-install mysqli pdo pdo_mysql
RUN curl -sS https://getcomposer.org/installer | php --
--install-dir=/usr/local/bin --filename=composer
```

Personalizar la imagen de PHP: Dockerfile

Definimos cuál será la imagen base de nuestro contenedor:

```
FROM php:7.2-fpm
```

Establecemos el directorio de trabajo:

```
WORKDIR "/application"
```

Una vez levantado el contenedor, si nos conectamos por bash accederemos al directorio /application en lugar de a /

Personalizar la imagen de PHP: Dockerfile

```
RUN apt-get update \
    && apt-get install -y libicu-dev libpng-dev libjpeg-dev libpq-dev \
    && apt-get clean; rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/ \
/usr/share/doc/* \
    && docker-php-ext-configure gd --with-png-dir=/usr --with-jpeg-dir=/usr \
    && docker-php-ext-install intl gd mbstring
```

- **apt-get update**: Para actualizar las fuentes de APT
- **apt-install -y**: Para instalar sin necesidad de confirmar por consola la acción
- **apt-get clean && rm**: Se elimina la caché y posibles ficheros temporales de la instalación. Recordemos la importancia de mantener nuestra imagen lo más liviana posible.
- **docker-php-ext-configure**: Agrega los parámetros necesarios a la configuración de PHP
- **docker-php-ext-install**: Se instalan las extensiones básicas de PHP

Personalizar la imagen de PHP: Dockerfile

```
RUN apt-get update && apt-get install -y mysql-client \
    && apt-get clean; rm -rf /var/lib/apt/lists/* /tmp/*
/var/tmp/* /usr/share/doc/* \
    && docker-php-ext-install mysqli pdo pdo_mysql
```

- **apt-get update**: Para actualizar las fuentes de APT
- **apt-install -y**: Para instalar sin necesidad de confirmar por consola la acción
- **apt-get clean && rm**: Se elimina la caché y posibles ficheros temporales de la instalación. Recordemos la importancia de mantener nuestra imagen lo más liviana posible.
- **docker-php-ext-install**: Se instalan las extensiones de PHP para trabajar con MySQL

Personalizar la imagen de PHP: Dockerfile

```
RUN curl -sS https://getcomposer.org/installer | php --
--install-dir=/usr/local/bin --filename=composer
```

Podríamos utilizar wget, pero tendríamos que instalarlo por APT y desinstalarlo posteriormente, para mantener la idea de “minimal installation”

Con este comando, lo que hacemos es instalar composer en /usr/local/bin bajo el nombre composer.

Personalizar la imagen de PHP: Dockerfile

- Guardamos el fichero Dockerfile y volvemos a la consola, al directorio que contiene el Dockerfile.

Personalizar la imagen de PHP: Dockerfile

- Guardamos el fichero Dockerfile y volvemos a la consola, al directorio que contiene el Dockerfile.
- Hacemos el build y el run de nuestra imagen de docker customizada:

```
$ docker build -t "php-fpm-base" .
```

Personalizar la imagen de PHP: Dockerfile

- Guardamos el fichero Dockerfile y volvemos a la consola, al directorio que contiene el Dockerfile.
- Hacemos el build y el run de nuestra imagen de docker customizada:

```
$ docker build -t "php-fpm-base" .
```

  - -t : El nombre que le damos a nuestra imagen
  - . : El directorio dónde está el Dockerfile a utilizar a la hora de construir la imagen.

Personalizar la imagen de PHP: Dockerfile

- Guardamos el fichero Dockerfile y volvemos a la consola, al directorio que contiene el Dockerfile.
- Hacemos el build y el run de nuestra imagen de docker customizada:

```
$ docker build -t "php-fpm-base" .
```

  - -t : El nombre que le damos a nuestra imagen
  - . : El directorio dónde está el Dockerfile a utilizar a la hora de construir la imagen.
- Ya tenemos la imagen creada. Ahora si volvemos a lanzar el comando, veremos que no tarda nada en crear la imagen, ésto es porque está utilizando la caché.

Personalizar la imagen de PHP: Dockerfile

Con la imagen construida, la ejecutamos:

```
$ docker run --name="php-fpm-base" php-fpm-base
```

Ejecutamos un comando de PHP en el contenedor para ver el phpinfo:

```
$ docker exec php-fpm-base php -i
```

Y hacemos lo mismo para ver que tenemos composer disponible:

```
$ docker exec php-fpm-base composer -V
```

Personalizar la imagen de PHP: Dockerfile

Creamos un directorio php-fpm-dev:

```
$ cd docker && mkdir php-fpm-dev && cd php-fpm-dev
```

Creamos un fichero Dockerfile:

```
$ touch Dockerfile
```

Lo abrimos con nuestro editor favorito:

```
$ vim Dockerfile
```

Personalizar la imagen de PHP: Dockerfile

Definimos cuál será la imagen base de nuestro contenedor, que será la que acabamos de crear:

```
FROM php-fpm-base
```

Instalamos xdebug + las dependencias para phpUnit

```
RUN pecl install xdebug && docker-php-ext-enable xdebug
```

```
RUN apt-get update \
    && apt-get -y install unzip zlib1g-dev \
    && apt-get clean; rm -rf /var/lib/apt/lists/* /tmp/*
/var/tmp/* /usr/share/doc/* \
    && docker-php-ext-install zip
```

Personalizar la imagen de PHP: Dockerfile

- Guardamos el fichero Dockerfile y volvemos a la consola, al directorio que contiene el Dockerfile.
- Hacemos el build y el run de nuestra imagen de docker customizada:

```
$ docker build -t "php-fpm-dev" .
$ docker run --name="php-fpm-dev" php-fpm-dev
```

ALTIA TECH DAY

altia

Llevando nuestras imágenes a GitHub

Llevando nuestras imágenes a GitHub

- Ya tenemos nuestras imágenes de PHP creadas en local.
- Aunque podemos trabajar en local… no deja de ser algo limitado.
- Lo interesante de Docker es poder llevarnos nuestras imágenes a cualquier lado.

Llevando nuestras imágenes a GitHub

So...

Llevando nuestras imágenes a GitHub

- Como primer paso, crearemos un repositorio en GitHub.
- En nuestro repositorio guardaremos nuestra configuración.
- Creamos el repositorio “docker-php” desde la interfaz de Github
- Vamos a transformar nuestro directorio de trabajo en el repositorio de docker-php

```
$ cd ..
```

Llevando nuestras imágenes a GitHub

```
$ git init
$ git config --global user.email "rolando.caldas@gmail.com"
$ git config --global user.name "Rolando Caldas"
$ git add .
$ git commit -m "My php docker containers configuration for development"
$ git remote add origin https://github.com/rolando-caldas/cebem-docker-php.git
$ git push -u origin master
```

Llevando nuestras imágenes a GitHub

- Ya tenemos nuestras imágenes de PHP en GitHub
- A partir de ahora podremos hacer un git clone en cualquier equipo y hacer el docker build en él.
- También podremos hacer commits, push y pull… para mantener actualizados nuestros Dockerfiles

Llevando nuestras imágenes a GitHub

...pero...

Llevando nuestras imágenes a GitHub

- ¿por qué tener que estar haciendo constantemente los build?

Llevando nuestras imágenes a GitHub

- ¿por qué tener que estar haciendo constantemente los build?
- ¿no sería mejor tener ya las imágenes construidas en alguna parte?

Llevando nuestras imágenes a GitHub

- ¿por qué tener que estar haciendo constantemente los build?
- ¿no sería mejor tener ya las imágenes construidas en alguna parte?
- Let me introduce the docker hub

ALTIA TECH DAY

altia

Docker Hub
https://hub.docker.com

Construyendo nuestras imágenes en Docker Hub

- Vinculamos Docker Hub con GitHub: Vamos a “Settings” => “Linked Accounts & Services” y clicamos en “Link Github”.

Construyendo nuestras imágenes en Docker Hub

- Vinculamos Docker Hub con GitHub: Vamos a “Settings” => “Linked Accounts & Services” y clicamos en “Link Github”.
- Aunque estamos usando GitHub, podremos hacer lo mismo con Bitbucket.

Construyendo nuestras imágenes en Docker Hub

- Vinculamos Docker Hub con GitHub: Vamos a “Settings” => “Linked Accounts & Services” y clicamos en “Link Github”.
- Aunque estamos usando GitHub, podremos hacer lo mismo con Bitbucket.
- Cualquier repositorio de Docker Hub que creemos público podremos tenerlo sin pagar nada. Para tener builds privados, tendremos que actualizarnos a la versión de pago.

Construyendo nuestras imágenes en Docker Hub

- Vinculamos Docker Hub con GitHub: Vamos a “Settings” => “Linked Accounts & Services” y clicamos en “Link Github”.
- Aunque estamos usando GitHub, podremos hacer lo mismo con Bitbucket.
- Cualquier repositorio de Docker Hub que creemos público podremos tenerlo sin pagar nada. Para tener builds privados, tendremos que actualizarnos a la versión de pago.
- Ya con la cuenta vinculada, podemos crear nuestro repositorio:

Construyendo nuestras imágenes en Docker Hub

- Vinculamos Docker Hub con GitHub: Vamos a “Settings” => “Linked Accounts & Services” y clicamos en “Link Github”.
- Aunque estamos usando GitHub, podremos hacer lo mismo con Bitbucket.
- Cualquier repositorio de Docker Hub que creemos público podremos tenerlo sin pagar nada. Para tener builds privados, tendremos que actualizarnos a la versión de pago.
- Ya con la cuenta vinculada, podemos crear nuestro repositorio:
- “Create” => “Create Automated Build” => “Create Auto-Build Github”

Construyendo nuestras imágenes en Docker Hub

- Vinculamos Docker Hub con GitHub: Vamos a “Settings” => “Linked Accounts & Services” y clicamos en “Link Github”.
- Aunque estamos usando GitHub, podremos hacer lo mismo con Bitbucket.
- Cualquier repositorio de Docker Hub que creemos público podremos tenerlo sin pagar nada. Para tener builds privados, tendremos que actualizarnos a la versión de pago.
- Ya con la cuenta vinculada, podemos crear nuestro repositorio:
- “Create” => “Create Automated Build” => “Create Auto-Build Github”
- Filtramos por nuestro repo de github “cebem-docker-php”

Construyendo nuestras imágenes en Docker Hub

- Escribimos la descripción
- Agregamos nuestras diferentes versiones de las imágenes a construir:
  - Branch => master => /php-base => latest
  - Branch => master => /php-base => base
  - Branch => master => /php-dev => dev
- Creamos nuestro repository.
- Vamos a “Build Settings” y clicamos en “Trigger” para el Docker Tag base.
- Esperamos (mucho) a que se construya la imagen… mientras tanto actualizaremos nuestros Dockerfiles.

Actualizado los Dockerfiles de nuestro repo

- Hasta ahora, los Dockerfiles de debug, dev y dev-mysql utilizaban imágenes de Docker creadas de forma local.
- Al tener ya nuestra php-base en el Docker Hub es el momento de hacer que la imagen base sea la del Docker Hub.
- Para los Dockerfiles de php-fpm-dev cambiamos el FROM por:
  `FROM rolandocaldas/cebem-php:php-base`
- Haremos un commit con todos los cambios.
- Ejecutamos el push.
- Puede que algún build falle, en cuyo caso se lanzará manualmente

Actualizado los Dockerfiles de nuestro repo

- Al tener nuestras imágenes de Docker subidas en el Docker Hub, podemos lanzar la que necesitemos como si fuese una de las imágenes oficiales:

```
$ docker run rolandocaldas/cebem-php:php-fpm-dev
```

- Además, el contenedor más completo tiene un peso de 192 MB.

Ahora bien, tenemos un contenedor con PHP-FPM, pero no deja de ser algo “cojo”...
… no tenemos servidor web …
… y tampoco tenemos un servidor MySQL ...

Un contenedor por servicio

Una idea latente en Docker es utilizar un contenedor por servicio. Esto significa que para una aplicación web PHP necesitaríamos:

- Un contenedor con PHP-FPM: Hecho!
- Un contenedor con un servidor web... por ejemplo Nginx. Pendiente :(
- Otro contenedor con MySQL. Pendiente :(

Un contenedor por servicio

Problema: Los contenedores por defecto no estarán en la misma red, por lo que entre ellos ni se ven, ni se escuchan… Nginx no podrá enviar la petición a PHP-FPM y éste no podrá conectarse al MySQL.

Solución: Crear en docker un network y correr todos los contenedores asociandolos al network creado.

Un contenedor por servicio

La cosa empieza a complicarse… afortunadamente tenemos…

Un contenedor por servicio

docker-compose

ALTIA TECH DAY

altia

docker-compose

docker-compose

docker-compose es una capa sobre docker que nos permite montar entornos complejos a partir de un simple yaml.

Todo lo que hacemos a través del docker-compose se puede hacer directamente con el comando de docker, aunque con un nivel de complejidad muy superior a utilizar docker-compose.

Para ejecutar el entorno que configuramos vía YAML sólo hay que lanzar:

```
$ docker-compose up -d
```

- -d: Lanza todos los contenedores en segundo plano, sin bloquear nuestro terminal.

Con docker-compose stop detendremos todos los contenedores asociados.

docker-compose

Para ejecutar el entorno que configuramos vía YAML sólo hay que lanzar:

```
$ docker-compose up -d
```

- -d: Lanza todos los contenedores en segundo plano, sin bloquear nuestro terminal.

Con docker-compose stop detendremos todos los contenedores asociados.

ALTIA TECH DAY

altia

Persistencia en Docker: Volúmenes

Persistencia en Docker:Volúmenes

Lo malo de los contenedores de Docker es que la información se almacena en el interior de los contenedores… si eliminamos un contenedor se pierden todos los cambios realizados.

Ésto nos genera muchas limitaciones.

Persistencia en Docker:Volúmenes

Afortunadamente, docker permite el uso de volúmenes.

Un volúmen en Docker es como un enlace simbólico, vinculamos un directorio o un fichero de nuestro equipo, con un directorio o fichero dentro de contenedor.

De esta forma, todo o que se almacene en un directorio del contenedor vinculado con local por un volúmen…. o todo fichero que se modifique que esté vinculado de igual manera no se perderá al eliminar el contenedor, porque es información persistida en nuestro equipo local.

ALTIA TECH DAY

altia

Caso real

Caso Real

```
git clone
https://github.com/rolando-caldas/workshop-gitlab-backend.git

git clone
https://github.com/rolando-caldas/workshop-gitlab-frontend.git
```

ALTIA TECH DAY

altia

altia.es