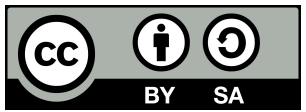
CURSO .PHP

GESTIÓN DE DEPENDENCIAS



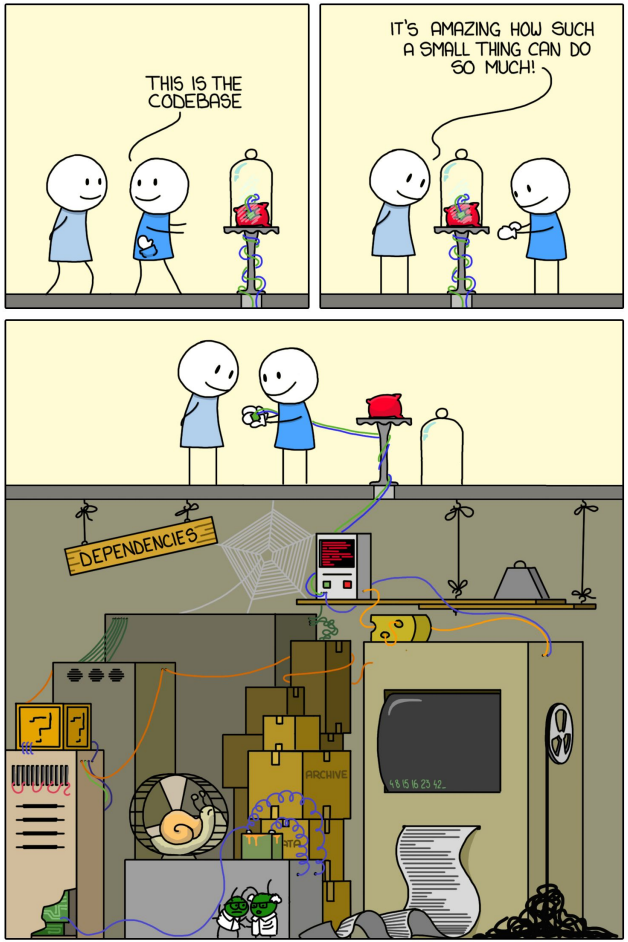
Autor: Jon Vadillo

www.jonvadillo.com

Contenidos

- Fundamentos básicos
- Composer
- Instalación de composer
- Definir dependencias
- Instalar dependencias
- Actualizar dependencias

IMPLEMENTATION



Las librerías, frameworks y componentes que utiliza una aplicación se conocen como dependencias.

Composer

- El gestor de dependencias más utilizado es ~~PEAR~~ Composer
- Composer permite indicar todas las librerías que va a utilizar la aplicación en un fichero y se encarga de la instalación/actualización.
- Las librerías no se instalan a nivel global, la gestión de dependencias se hace por proyecto.

Instalación

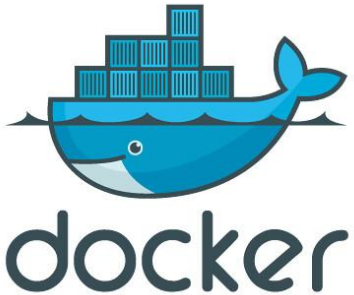
- Windows
  - Descargar el instalador oficial.
  - Ejecutar y seguir las instucciones.
- Linux / Unix / macOS
  - Ejecutar los 4 comandos de la documentación oficial para descargar, comprobar y ejecutar el instalador.

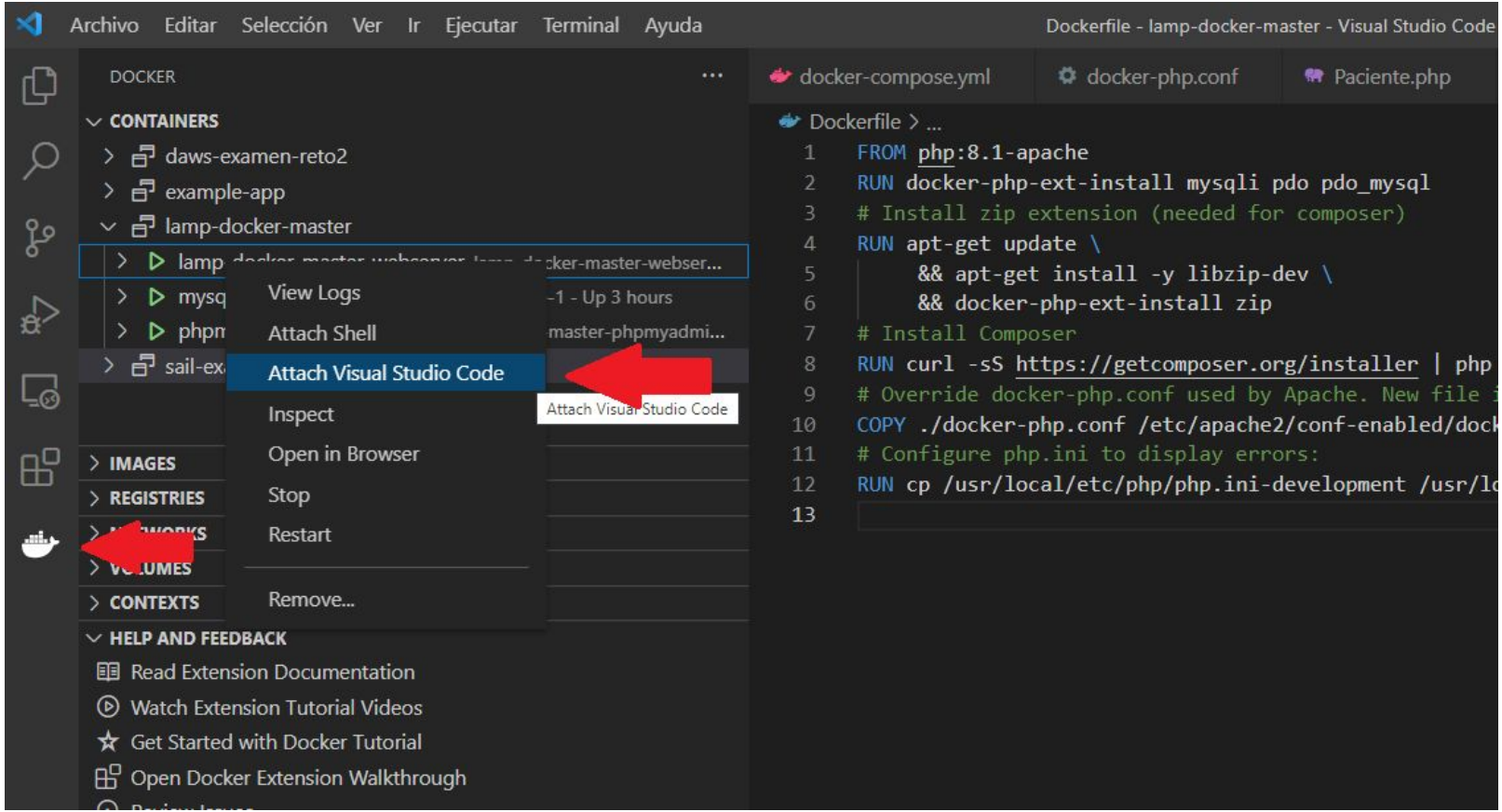
Instalación

- Docker
  - Descarga el repositorio con el entorno de Docker preparado (servidor web + base de datos).
  - Ejecuta el siguiente comando para abrir un terminal dentro del contenedor:

    ```
    docker exec -it <mycontainer> bash
    ```

  - Puedes comprobar que composer está instalado ejecutando: `composer –version`




Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda
Dockerfile - lamp-docker-master - Visual Studio Code
DOCKER
CONTAINERS
daws-examen-reto2
example-app
lamp-docker-master
View Logs
Attach Shell
Attach Visual Studio Code
Inspect
Open in Browser
Stop
Restart
Remove...
Attach Visual Studio Code
IMAGES
REGISTRIES
CONTEXTS
HELP AND FEEDBACK
Read Extension Documentation
Watch Extension Tutorial Videos
Get Started with Docker Tutorial
Open Docker Extension Walkthrough
docker-compose.yml
docker-php.conf
Paciente.php
Dockerfile > ...
1 FROM php:8.1-apache
2 RUN docker-php-ext-install mysqli pdo pdo_mysql
3 # Install zip extension (needed for composer)
4 RUN apt-get update \
5 && apt-get install -y libzip-dev \
6 && docker-php-ext-install zip
7 # Install Composer
8 RUN curl -sS https://getcomposer.org/installer | php
9 # Override docker-php.conf used by Apache. New file i
10 COPY ./docker-php.conf /etc/apache2/conf-enabled/dock
11 # Configure php.ini to display errors:
12 RUN cp /usr/local/etc/php/php.ini-development /usr/lo
13

Paso 1: definir las dependencias

- Composer gestiona las dependencias del proyecto mediante el fichero composer.json. Créalo manualmente en la raíz de tu proyecto o ejecuta el comando **`composer init`** para crearlo.
- El comando **`composer require`** añade una dependencia al fichero (o lo crea si no existe).

```
composer require twig/twig:^2.0
```

- También es posible modificar el fichero desde un editor de texto (luego será necesario ejecutar el comando **`composer install`**).

composer.json

```
{
  "require": {
       "facebook/php-sdk": "3.2.*",
       "twig/twig": "1.*",
  }
}
```

Paso 2: instalar las dependencias

- Una vez creado el fichero composer.json, ejecutar el comando:

```
composer install
```

- Este comando descarga los ficheros de las librerías indicadas en la carpeta `/vendor`
- Para cargar y utilizar las dependecias, sólo hay que añadir la siguiente línea al fichero .php principal de la aplicación:

```
require 'vendor/autoload.php';
```

Actualizar las dependencias

- Para actualizar las dependencias ejecutar el comando:

```
composer update
```

- Este comando es útil cuando se declaran las versiones de las dependencias de forma flexible: 1.8.* o ~1.8
- Composer crea el fichero **`composer.lock`** de forma automática en el que indica las versiones instaladas. Es útil a la hora de compartir el proyecto, para que otros desarrolladores instalen la aplicación con las mismas versiones.

Hands on!

Ejercicio 1

- Crea una sencilla aplicación que muestre por pantalla 100.000 números aleatorios y al finalizar muestre el tiempo que ha tardado en mostrarlos. Para ello utiliza la librería “phpunit/php-timer”. Búscala en el repositorio de **Packagist**, donde tendrás toda la información necesaria para instalarla utilizando Composer.

  Nota: tendrás que buscar la función de PHP que permite generar números aleatorios.

Hands on!

Ejercicio 2

- Una acción muy habitual al desarrollar aplicaciones es crear logs de lo que está sucediendo en nuestra aplicación (ya sean de aviso, error o informativos). Desarrolla una aplicación que cumpla lo siguiente:
  - Tendrá como dependencia la librería **Monolog**. Esta librería irá escribiendo en un fichero llamado “app.log” los logs descritos en este enunciado.
  - Debe contener una clase llamada `Paciente` con las propiedades nombre, apellidos y edad que deberán enviarse como parámetros en el constructor. También tendrá un logger como propiedad que inicializará en el constructor. También incluirá los métodos enfermar() y curar(), los cuales únicamente generarán un log que diga: He enfermado / Me he curado.
  - Crea un archivo index.php que muestre dos enlaces: enfermar y curar. Cada vez que un usuario haga click en uno de los enlaces, creará un objeto de la clase paciente y llamará al método enfermar o curar en función de lo clickado por el usuario.

Autoload de clases locales

- Composer puede encargarse de incluir también las clases creadas por nosotros en el proyecto.
- Pasos:
  - **Paso 1.** Incluir todas las clases que queramos cargar automáticamente en un directorio. Por ejemplo ‘src’:

```
mi-proyecto/
  src/
    Db.php
    Persona.php
    Empelado.php
```

Autoload de clases locales

- **Paso 2**. Asignar un Namespace a todas las las clases del directorio:

```php
namespace Egibide;

class Persona {
    private $nombre;
    private $profesion;

    ...
```

Autoload de clases locales

- **Paso 3**. Relacionar el Namespace creado con el directorio ‘src’ en el fichero composer.json:

```
{
  "autoload": {
    "psr-4": {
      "Egibide\\":"src/"
    }
  }
}
```

  - Es necesario utilizar la clave “psr-4” y finalizar el nombre del namespace con “\\”

Autoload de clases locales

- **Paso 4**. Actualizar el autoloader de composer:

```
$ composer dumpautoload -o
```

- **Paso 5.** Importar el namespace cuando queramos utilizar las clases:

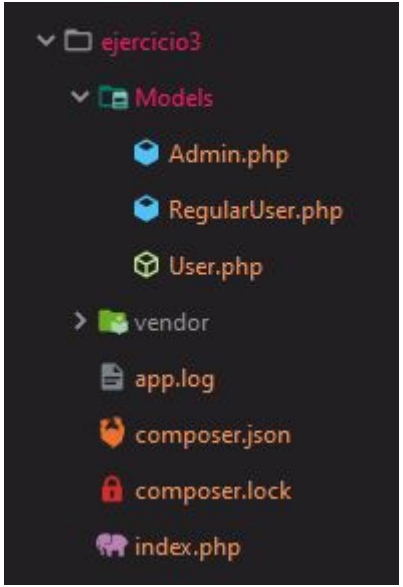
```php
require 'vendor/autoload.php';

use Egibide\Persona;

$persona1 = new Persona("Ane Larrain", 32);
```

Hands on!

Ejercicio 3

Crea una aplicación con las siguientes clases:

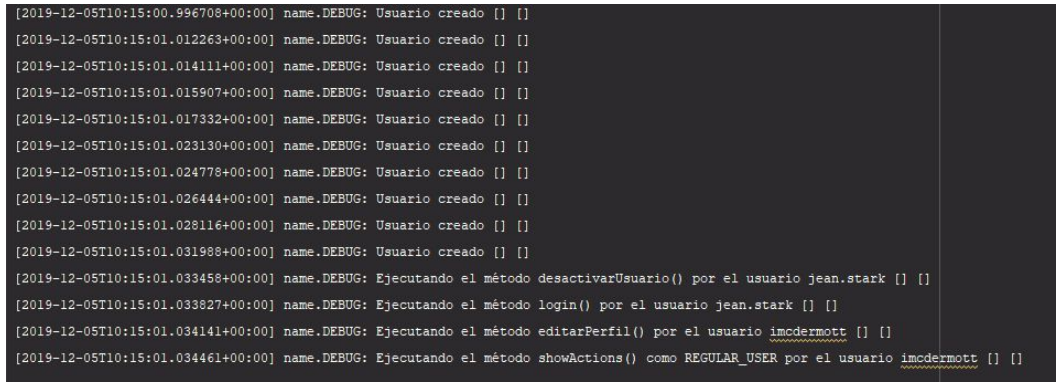
- User
  - Propiedades (protected): username, nombre, apellidos, email, password, ultimo_acceso (formato DateTime), log (instancia de Monolog)
  - Métodos: showActions (abstracto), login, logout, getters y setters.
- Admin (extiende de User)
  - Propiedades: admin_level (valores posibles: 1,2,3)
  - Métodos: desactivarUsuario()
- RegularUser (extiende de user)
  - Propiedades: twitter, instagram
  - Métodos: editarPerfil()



Hands on!

Ejercicio 3

- Crea un index.php que genere 5 usuarios de cada tipo mediante un bucle. Los datos de los usuarios se generarán utilizando la librería fzaninotto/faker.
- Los métodos (incluido el constructor) almacenarán un mensaje en un log (mediante Monolog) con el siguiente mensaje: “Ejecutando el método <NOMBRE:METODO> por el usuario <USERNAME>”.
- Las clases se almacenarán en la carpeta “Models” y se incluirán automáticamente con composer.



require vs require-dev

- Las dependencias incluidas en require-dev son aquellas que **no son necesarias en la versión de producción**.
- Normalmente **se utilizan únicamente durante el desarrollo**, como PHPUnit.
- Estas dependencias **no se instalan cuando se utilizan en otros proyectos** (por ejemplo, si nosotros incluimos el paquete twig/twig, no se instalarán sus dependencias de tipo dev.
- También podemos utilizar **el flag "`--no-dev`"** en los comandos install o uptdate.

Sources

- Composer official docs: https://www.getcomposer.org
- PHP Enthusiast: https://phpenthusiast.com/blog/how-to-autoload-with-composer