## Composer et la gestion de dépendances

Composer **n'est pas juste un gestionnaire de packages** dans le même sens que Yum ou Apt (en linux) sont. Oui, **il traite de «packages» ou des bibliothèques dans un projet** (pas globalement). Composer les installe dans un répertoire (par exemple : « libsExternes » ou « vendor ») dans votre projet.

**Bibliothèque** : morceau de code réutilisable dans d’autres programmes.

**Package** : c’est souvent une bibliothèque qui est prête à être installée en utilisant un gestionnaire de packages/dépendances.

**C'est donc un gestionnaire de dépendances, car il nous permet d'installer le code dont notre application dépend d'une manière facile**

Cette idée n'est pas nouvelle et Composer est fortement inspiré par le **npm** de **node.js** et par le gestionnaire de Ruby's.

Note : Il supporte cependant un projet "global" pour plus de commodité via la commande **global** mais on n’étudiera pas cette commande dans le cadre de notre cours.

## Fondements de base de composer

**L’utilité de base est de faciliter la gestion des packages dans un projet.**

**Vous avez un projet qui dépend d'un certain nombre de packages** de tiers, et certaines de ces packages dépendent d'autres packages. En principe vous devriez télécharger/copier tous les packages dans votre projet et **gérer vous-mêmes tant leurs versions comme leur emplacement**. Ceci peut vite devenir un cauchemar pour de différentes de raisons : quelle est la version d'un package qui est en train d'être utilisée par chaque dev? Où est-ce qu'on trouve le package??

**Composer** vous permet de **spécifier** les packages dont votre application dépende **dans un seul fichier** **composer**.**json**. Dans **ce fichier vous indiquez les packages qui peuvent et doivent être installés pour votre projet, ainsi que leur version.** Une fois vous avez créé le fichier, la commande suivante

composer install

téléchargera les bibliothèques et les installera dans un dossier dans votre projet. Vous avez un control total sur les packages que vous utilisez.

Nous pouvons aussi **utiliser la ligne de commande pour spécifier (installer, mettre à jour, effacer)** les packages qu'on utilisera dans notre projet. Ces actions auront un effet direct sur le fichier **composer.json** (ex: composer install unRepo/unPackage rajoutera une ligne **require** dans composer.json)

Nous allons voir ces concepts avec un exemple pratique.

## Installation de Composer

**Téléchargez** Composer pour Windows/Linux/Mac sur le site

[**https://getcomposer.org/**](https://getcomposer.org/)

## Utilisation de Composer : exemple pratique

1. **Créez un dossier** (ex : ComposerProjet1) pour votre projet PHP

Nous avons un projet vide, mais on va télécharger avec **composer** – et pas manuellement ! - une dépendance : un package qui fournit la fonctionnalité pour générer de fichier de log en disque, BD etc... Il s'agit de **monolog** (<https://packagist.org/packages/monolog/monolog>).

1. Créez un fichier **composer.json** où vous allez spécifier une dépendance :

{

    "require": {

        "monolog/monolog": "1.0.\*"

    }

}

**require** : indique à composer quelles sont les packages à utiliser. Le fichier composer.json sera modifié et les dépendances (ici seulement un package) seront téléchargées

Le premier **monolog** est le fournisseur de la libraire à télécharger et le deuxième est le package en soi.

**1.0.\*** : indique la version à télécharger. Cela signifie toute version dans la branche de développement 1.0. C'est l'équivalent de dire des versions qui correspondent à >= 1.0 <1.1 (entre 1.0 et 1.1, sans inclure la derniere)

1. **Installation des dépendances** du projet : **composer install**

Dans la console, allez dans le dossier de votre projet et tapez **composer install**. Cette commande téléchargera le package dans le dossier **vendor** de votre projet.

Composer créera aussi un fichier **composer.lock**

**Les packages téléchargées** (ici uniquement **monolog**) se trouveront dans un dossier **vendor** dans votre projet.

Vous pouvez aussi utiliser la commande **composer remove** en console pour effacer une dépendance du projet :

**Exemple :**

composer remove monolog/monolog

composer remove twig/twig --update-with-dependencies

1. Testez la fonctionnalité du package dans votre code PHP. Créez un ficher exemple.php contenant ce code :

<?php

require "./vendor/autoload.php";

// importer les namespaces

use Monolog\Logger;

use Monolog\Handler\StreamHandler;

// créer un log (ici un fichier dans le disque)

$objectLogger = new Logger('monLogger');

$objectLogger->pushHandler(new StreamHandler('monLogConnexions.log', Logger::INFO));

// écrire de messages

$objectLogger->info("User xxxxx connecté");

$objectLogger->err('Mauvais password');

?>

## Le control de versions : le fichier composer.lock

**La première fois** qu’on lance **« composer install » , Composer lira le fichier composer.json et téléchargera la version indiquée de chaque package, en plus de créer le fichier composer.lock** qui spécifie les **packages installés et leur version**. Après avoir installé les dépendances, Composer écrit la liste des versions exactes actuellement installées dans un fichier composer.lock. Cela **verrouille le projet à ces versions spécifiques.** Voici pouquoi :

La commande d'installation **« composer install » vérifie toujours si un fichier de verrouillage (composer.lock) est présent. Si le fichier existe, il télécharge les versions qui correspondent au composer.lock** indépendamment de ce que indique composer.json au moment de lancer la commande (car on a pu le changer entre temps)

**Cela signifie que quiconque configure le projet téléchargera la même version des dépendances**. Votre serveur CI, les machines de production, les autres développeurs de votre équipe, **tout le monde fonctionne sur les mêmes dépendances, ce qui atténue le risque de bugs affectant seulement certaines parties des déploiements**. Même si vous développez seul, dans six mois lors de la réinstallation du projet, vous pouvez être sûr que les dépendances installées fonctionnent toujours même si vos dépendances ont publié de nombreuses nouvelles versions depuis.

**Si aucun fichier composer.lock** n'existe, Composer **lira les dépendances et les versions de composer.json et créera le fichier de verrouillage après l'exécution de composer update** ou **composer** **install**.

Note : si vous utilisez un système de contrôle de versions (GIT, par exemple) faites commit aussi du composer.lock de votre application (avec composer.json).

## Mise à jour de dépendances (composer update)

Si **l'une des dépendances est mise à jour**, vous **n'obtiendrez pas ces mises à jour automatiquement**. Pour mettre à jour la nouvelle version, utilisez la commande **composer update**

**Cela récupérera les versions indiquées dans composer.json** et mettra également à jour le fichier de verrouillage avec la nouvelle version.

**Remarque :** Composer affichera un **avertissement lors de l'exécution de composer install**

**si composer.lock et composer.json ne sont pas synchronisés** (ex : changez le composer.json et mettez la version 1.1)

Si vous souhaitez uniquement installer ou mettre à jour une dépendance, vous pouvez l’indiquer de façon explicite avec **composer update monolog/monolog** dans la console

## Des origines des packages: the Packagist

Le **Packagist** est le dépôt principal de Composer. Un dépôt (repository) de Composer est essentiellement **une source de packages**: un endroit où vous pouvez obtenir de packages. Le Packagist vise à être le référentiel central que tout le monde utilise. Cela signifie que vous pouvez automatiquement demander tout package qui est disponible là-bas.

**Si vous allez sur le site Web Packagist (packagist.org), vous pouvez parcourir et rechercher des packages**.

**Tout projet open source utilisant Composer est recommandé de publier leurs packages sur Packagist**. Un package n'a pas besoin d'être sur Packagist pour être utilisée par Composer, mais elle permet la découverte et l'adoption par d'autres développeurs plus rapidement.

## Chargement des Packages : autoload de base

Une fois les Packages se trouvent dans le dossier de notre projet, on doit les « inclure » dans nos scripts (pages).

Pour les Packages qui spécifient des informations de chargement automatique (**autoload**), Composer génère un fichier vendor/autoload.php. Vous pouvez simplement inclure ce fichier et vous obtiendrez l'autoloading automatiquement.

require \_\_DIR\_\_ . '/vendor/autoload.php';

**Cela rend très facile à utiliser le code de tiers**. Par exemple : Si votre projet dépend de Monolog, vous pouvez simplement commencer à utiliser des classes de celui-ci, et elles seront chargées automatiquement.