PHP - Composer Dependency Management Research [June 2017]

This document contains a brief overview of PHP - Composer dependency management state, issues, practices etc. As these concept are wide and complex, not all details are within this text. For more information, please review the official online documentation. Also, please see the corresponding POC

1. Introduction

Composer is not a package manager in the same sense as Yum or Apt are. Yes, it deals with "packages" or libraries, but it manages them on a per-project basis, installing them in a directory (e.g. vendor) inside your project. By default it does not install anything globally. Thus, it is a dependency manager. It does however support a "global" project for convenience via the global command.
This idea is not new and Composer is strongly inspired by node's npm and ruby's bundler.
Composer uses this information to search for the right set of files in package "repositories" that you register using the repositories key, or in Packagist, the default package repository.

2. Installing Dependencies

To install the defined dependencies for your project, just run the install command.
php composer.phar install
When you run this command, one of two things may happen:

1. Installing Without composer.lock

   If you have never run the command before and there is also no composer.lock file present, Composer simply resolves all dependencies listed in your composer.json file and downloads the latest version of their files into the vendor directory in your project. (The vendor directory is the conventional location for all third-party code in a project).

2. Installing With composer.lock

   If there is already a composer.lock file as well as a composer.json file when you run composer install, it means either you ran the install command before, or someone else on the project ran the install command and committed the composer.lock file to the project (which is good).

   Either way, running install when a composer.lock file is present resolves and installs all dependencies that you listed in composer.json, but Composer uses the exact versions listed in composer.lock to ensure that the package versions are consistent for everyone working on your project. As a result you will have all dependencies requested by your composer.json file, but they may not all be at the very latest available versions (some of the dependencies listed in the composer.lock file may have released newer versions since the file was created). This is by design, it ensures that your project does not break because of unexpected changes in dependencies.

3. Package Names

The package name consists of a vendor name and the project's name. Often these will be identical - the vendor name just exists to prevent naming clashes. For example, it would allow two different people to create a library named json. One might be named igorw/json while the other might be seldaek/json.

4. Dependency Resolution

Normally, Composer deals with tags (as opposed to branches). When you write a version constraint, it may reference a specific tag (e.g., 1.1) or it may reference a valid range of tags (e.g., >=1.1 <2.0, or ~4.0). To resolve these constraints, Composer first asks the VCS to list all available tags, then creates an internal list of available versions based on these tags.

When Composer has a complete list of available versions from your VCS, it then finds the highest version that matches all version constraints in your project (it's possible that other packages require more specific versions of the library than you do, so the version it chooses may not always be the highest available version) and it downloads a zip archive of that tag to unpack in the correct location in your vendor directory. [see versions for more info]

5. Best practices

1. If you are using git for your project, you probably want to add vendor in your .gitignore. You really don't want to add all of that third-party code to your versioned repository.

2. Commit Your composer.lock File to Version Control: Committing this file to VC is important because it will cause anyone who sets up the project to use the exact same versions of the dependencies that you are using.

6. Packagist

Packagist is the main Composer repository. A Composer repository is basically a package source: a place where you can get packages from. Packagist aims to be the central repository that everybody uses. This means that you can automatically require any package that is available there, without further specifying where Composer should look for the package.
If you go to the Packagist website (packagist.org), you can browse and search for packages.
Any open source project using Composer is recommended to publish their packages on Packagist. A library does not need to be on Packagist to be used by Composer, but it enables discovery and adoption by other developers more quickly.