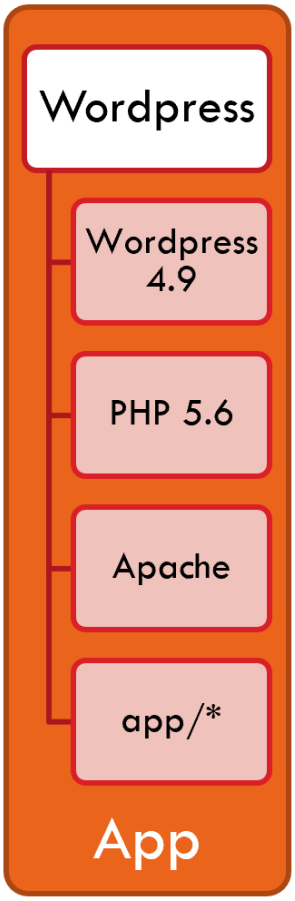
Solves Dependency Conflicts

In this lesson, you will learn how containers solve dependency conflicts between similar applications on the same server.

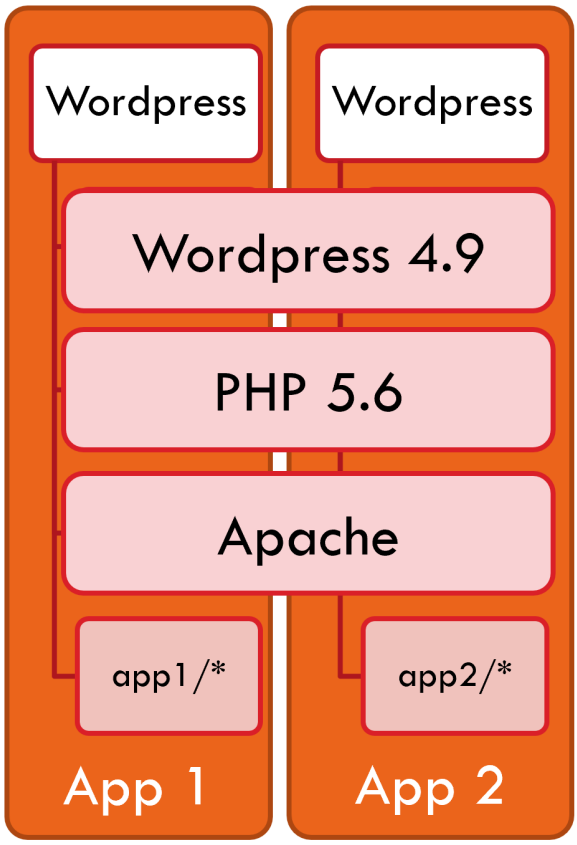
A typical web application looks something like the following:



A Web Application Model

The application is made of files served by an HTTP server (Apache here, but it could be Kestrel, IIS, NGINX, ...), a runtime (PHP 5.6 here) and a development framework (Wordpress 4.9 here).

Without containers, the dependencies and files are all placed together on a server. Since managing these dependencies is time-consuming, similar apps are typically grouped on the same server, sharing their dependencies:

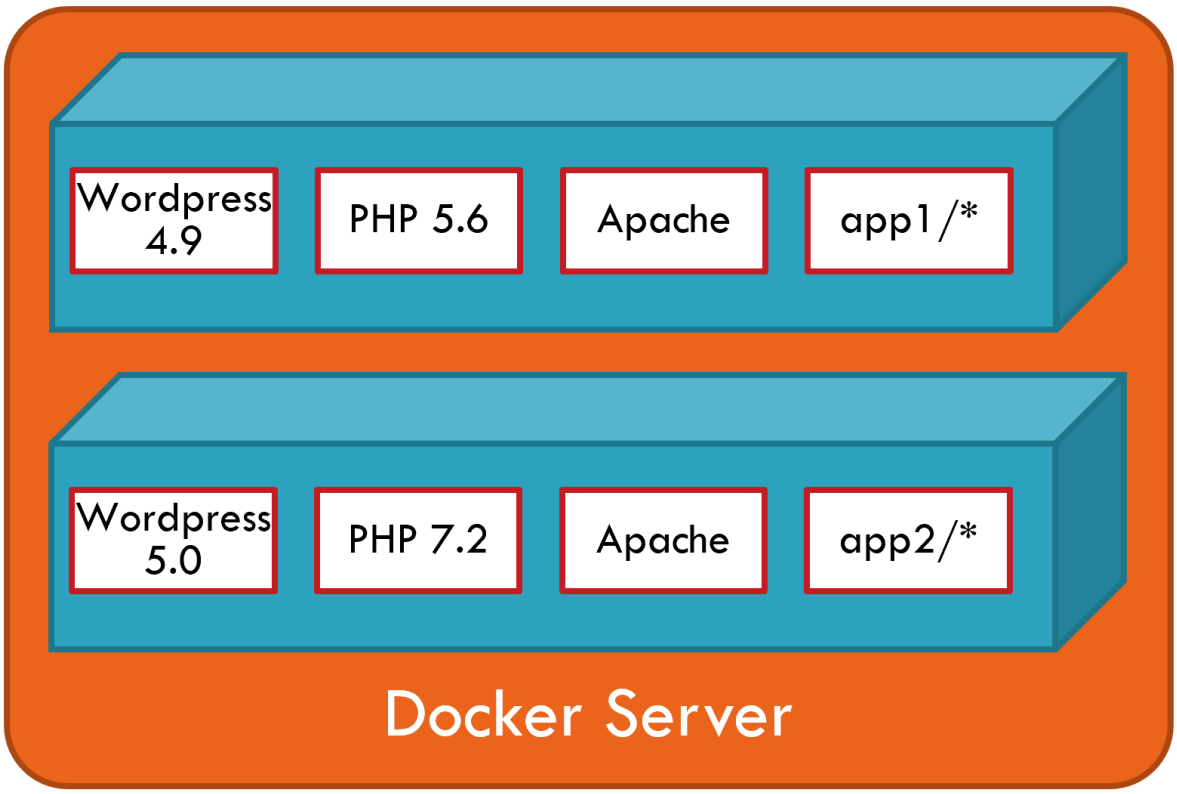


A model of two web applications sharing the same dependencies

Now suppose you want to upgrade the PHP runtime from version 5.6 to 7.2. However, the version change induces breaking changes in the applications that therefore need to be updated. You need to update both *App 1* and *App 2* when proceeding with the upgrade. On a server that may host many apps of this type, this is going to be a daunting task, and you'll need to delay the upgrade until all apps are ready.

Another similar problem is when you want to host *App 3* on the same server, but *App 3* uses the Node.JS runtime together with a package that, when installed, changes a dependency used by the PHP runtime. Conflicts between runtimes happen often, so you've probably faced that problem already.

Containers solve this problem since each app will run inside its own container with its own dependencies. Your typical server would look like:



A model of two web applications on a Docker Server

Each container encapsulates its own dependencies. Which means you can migrate the PHP runtime from version 5.6 to 7.2 in a container without it affecting others. Any other container that would use, for instance, Node.JS would not interfere with any of the Wordpress containers.

In the next lesson, we will continue our discussion on the advantages of containers.