How To Install and Use Composer on Debian 8

Posted February 15, 2017 83.1k PHP DEBIAN

2

By: Achilleas Pipinellis

Not using **Debian 8**? Choose a different version:

Introduction

Composer is a popular dependency management tool for PHP, created mainly to facilitate installation and updates for project dependencies. It will check which other packages a specific project depends on and install them for you, using the appropriate versions according to the project requirements.

This tutorial will show how to install and get started with Composer on a Debian 8 server.

Prerequisites

For this tutorial, you will need:

- One Debian 8 server with a non-root sudo user, as shown in Initial Server Setup with Debian 8

Step 1 — Installing the Dependencies

Before we download and install Composer, we need to make sure our server has all necessary dependencies installed.

First, update the package manager cache.

```
$ sudo apt-get update
```

Now, let's install the dependencies. We'll need `curl` in order to download Composer and `php5-cli`, a PHP package, for installing and running it. Composer uses `git`, a version control system, to download project dependencies. You can install all three of these packages at once with this command:

```
$ sudo apt-get install curl php5-cli git
```

Now that the essential dependencies are installed, let's proceed and install Composer itself.

Step 2 — Downloading and Installing Composer

We will follow the instructions as written in Composer's official documentation with a small modification to install Composer globally under `/usr/local/bin`. This will allow every user on the server to use Composer.

Download the installer to the `/tmp` directory.

```
$ php -r "copy('https://getcomposer.org/installer', '/tmp/composer-setup.php');"
```

Visit Composer's pubkeys and signatures page and copy the SHA-384 string at the top . Then, run the following command by replacing `sha_384_string` with the string you copied.

```
$ php -r "if (hash_file('SHA384', '/tmp/composer-setup.php') === 'sha_384_string') { echo 'Installer
```

This command checks the hash of the file you downloaded with the correct hash from Composer's website. If it matches, it'll print **Installer verified**. If it doesn't match, it'll print **Installer corrupt**, in which case you should double check that you copied the SHA-384 string correctly.

Next, we will install Composer. To install it globally under `/usr/local/bin`, we'll use the `--install-dir` flag; `--filename` tells the installer the name of Composer's executable file. Here's how to do this in one command:

```
$ sudo php /tmp/composer-setup.php --install-dir=/usr/local/bin --filename=composer
```

You'll see a message like the following:

```
Output
All settings correct for using Composer
Downloading...

Composer (version 1.3.2) successfully installed to: /usr/local/bin/composer
Use it: php /usr/local/bin/composer
```

You can verify that Composer is correctly installed by checking its version.

```
$ composer --version
```

You should see the version that was installed. At that time of this writing, the version is:

Composer version

```
Composer version 1.3.2 2017-01-27 18:23:41
```

Finally, you can safely remove the installer script as you no longer need it.

```
$ rm /tmp/composer-setup.php
```

Composer is now set up and running, waiting to be used by your project. In the next section, you'll generate the `composer.json` file, which includes the PHP libraries that your project depends on.

Step 3 — Generating the composer.json File

In order to use Composer for a project, you'll need a `composer.json` file. The `composer.json` file tells Composer which dependencies it needs to download for your project, and which versions of each package are allowed to be installed. This is important to keep your project consistent and avoid installing unstable versions that could potentially cause backwards compatibility issues.

You don't need to create this file manually; it's easy to run into syntax errors if you do. Composer auto-generates the `composer.json` file when you add a dependency to your project using the `require` command. Additional dependencies can also be added in the same way, without the need to manually edit this file.

The process of using Composer to install a package as dependency in a project usually involves the following steps:

- Identify what kind of library the application needs
- Research a suitable open source library on Packagist.org, the official repository for Composer
- Choose the package you want to depend on
- Run `composer require` to include the dependency in the `composer.json` file and install the package

We'll see how this works in practice with a simple demo application.

The goal of this application is to transform a given sentence into a *slug*, which is a URL-friendly string. This is used to convert page titles to URL paths (like the final portion of the URL for this tutorial).

Let's start by creating a directory for the project. We'll call it **slugify**:

```
$ cd ~
$ mkdir slugify
$ cd slugify
```

Next, let's search for the library that we need to use.

Searching for Packages on Packagist

Next, we'll search Packagist for a package to help generate slugs. If you search for "slug", you'll see a list of packages. On the right side of each package in the list, you'll see two numbers: the number on top is how many times the package was installed and the number on the bottom shows how many times a package was starred on GitHub.

Generally speaking, packages with more installations and stars tend to be more stable because many people are using them. It's also important to check the package description to make sure the package is really what you're looking for.

What we need is a simple string-to-slug converter. As an example here, we'll use the package `cocur/slugify`. It seems to be a good match because it has a reasonable amount of installations and stars.

You will notice that the packages on Packagist have a vendor name and a package name. Each package has a unique identifier (a namespace) in the same format GitHub uses for its repositories: `vendor/package`. The library we want to install uses the namespace `cocur/slugify`. The namespace is what we need in order to require the package in our project.

Now that we identified the library we want, let's add it to the `composer.json` file.

Requiring a Package

We can run `composer require` to include the library as a dependency and also generate the `composer.json` file for the project:

```
$ composer require cocur/slugify
```

```
Output
Using version ^2.3 for cocur/slugify
./composer.json has been created
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Installing cocur/slugify (v2.3) Downloading: 100%
Writing lock file
Generating autoload files
```

As you can see from the output, Composer automatically decided which version of the package should be used. If you check your project's directory now, it will contain two new files: `composer.json` and `composer.lock`, and a `vendor/` directory:

```
$ ls -l
```

```
Output
total 12
-rw-r--r-- 1 sammy sammy   59 Feb  1 13:43 composer.json
-rw-r--r-- 1 sammy sammy 2896 Feb  1 13:43 composer.lock
drwxr-xr-x 4 sammy sammy 4096 Feb  1 13:43 vendor
```

The `composer.lock` file is used to store information about which versions of each package are installed, and make sure the same versions are used if someone else clones your project and installs its dependencies. The `vendor/` directory is where the project dependencies are located. The `vendor/` folder should not be committed into version control; you only need to include the `composer.json` and `composer.lock` files.

> **Note:** When installing a project that already contains a `composer.json` file, you need to run `composer install` in order to download the project's dependencies.

You may notice that the `composer.lock` file includes specific information about the version of the PHP libraries our project depends on. Composer uses a special syntax to restrict libraries to specific versions. Let's see how that works.

Understanding Version Constraints

If you check the contents of your `composer.json` file, you'll see something like this:

```
$ cat composer.json
```

Output

```
{
    "require": {
        "cocur/slugify": "^2.3"
    }
}
```

There is a caret, `^`, before the version number. Composer supports several different constraints and formats for defining the required package version, in order to provide flexibility while also keeping your project stable. The caret operator used by the auto-generated `composer.json` file is the recommended operator for maximum interoperability, following semantic versioning. In this case, it defines **1.3** as the minimum compatible version, and allows updates to any future version below **2.0**. You can read more information about the versions rationale in Composer's versioning documentation.

So far, we've seen how to add and restrict the PHP libraries our project needs with Composer using the `composer.json` file. The next step is to actually use these libraries inside our application. For that purpose, Composer provides the `autoload.php` file that facilitates the process of loading external libraries.

Step 4 — Including the Autoload Script

Composer provides an autoload script that you can include in your project to get autoloading for free. This makes it much easier to work with your dependencies and define your own namespaces. The only thing you need to do is include the `vendor/autoload.php` file in your PHP scripts before any class instantiation.

Let's come back to the `slugify` example application. Using `nano` or your favorite text editor, create a `test.php` script where we'll use the `cocur/slugify` library.

```
$ nano test.php
```

test.php

```
<?php
require __DIR__ . '/vendor/autoload.php';

use Cocur\Slugify\Slugify;

$slugify = new Slugify();

echo $slugify->slugify('Hello World, this is a long sentence and I need to make a slug from it!');
```

You can run the script in the command line with:

```
$ php test.php
```

This should produce the output:

```
Output
hello-world-this-is-a-long-sentence-and-i-need-to-make-a-slug-from-it
```

You've fully set up Composer, but read on to learn how to maintain the installation.

Step 5 — Updating Composer and Project Dependencies (Optional)

To update Composer itself, you can use the built-in command `self-update` it provides. Because Composer is installed globally, you need to run the command with root privileges.

```
$ sudo -H composer self-update
```

You can update one or more specific libraries by listing them specifically with `composer update`.

```
$ composer update namespace/package
```

If you want to update all of your project dependencies, run the `update` command.

```
$ composer update
```

This will check for newer versions of the libraries you required in your project. If a newer version is found and it's compatible with the version constraint defined in the `composer.json` file, it will replace the previous version installed. The `composer.lock` file will be updated to reflect these changes.

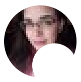
Conclusion

Composer is a powerful tool every PHP developer should have in their utility belt. Beyond providing an easy and reliable way for managing project dependencies, it also establishes a new de facto standard for sharing and discovering PHP packages created by the community.

In this tutorial we briefly described the basics of Composer: how to install it, how to create a project and find packages for it, and how to maintain it. To learn more, you can explore the official Composer documentation.

By: Achilleas Pipinellis

Upvote (2) Subscribe Share



Editor:
Hazel Virdó

Related Tutorials

How To Deploy a PHP Application with Kubernetes on Ubuntu 16.04

How To Set Up Laravel, Nginx, and MySQL with Docker Compose

How to Deploy a Symfony 4 Application to Production with LEMP on Ubuntu 18.04

How To Install and Use Composer on Debian 9

How To Install and Secure phpMyAdmin on Debian 9

2 Comments

Leave a comment...

Log In to Comment

jlinkelsfb *April 8, 2017*

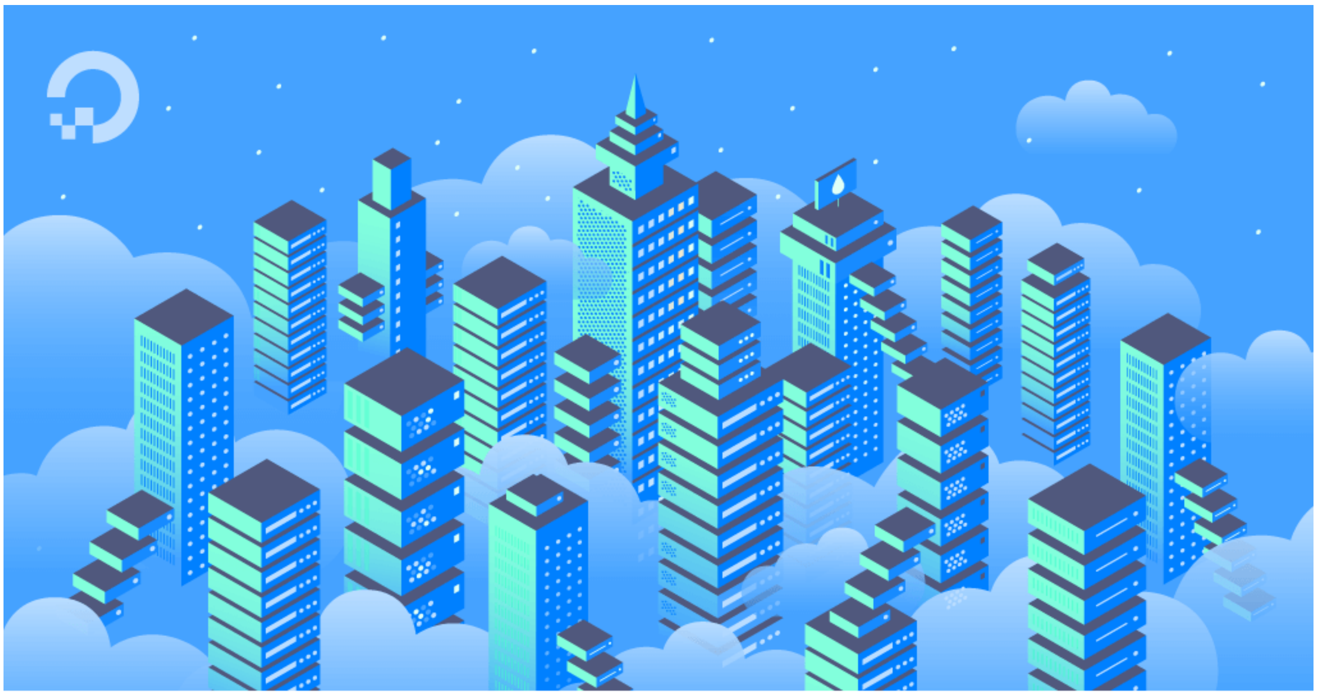
0 Thanks Achilleas and Hazel for writing this down. I am a die-hard Debian forever user. And this is the first time I went through an understandable tutorial for composer. There are plenty of step-by-step instruction which dumb you down to step1, step2... step-n. You never understand what is going on. And because of that you

don't know whether it fits the Debian policy. (Debian is great, but doing things the "Debian way" is really better for keeping you system stable).
So now finally I am not left out anymore in using Composer.

---

thedude *May 20, 2017*

0 You're very welcome :) This article is mostly based on https://www.digitalocean.com/community/tutorials/how-to-install-and-use-composer-on-ubuntu-14-04 to be frank. Glad you liked it!



How To Install and Use Composer on Ubuntu 14.04

by Erika Heidi

Composer is a popular dependency management tool for PHP, created to facilitate installation and update of project dependencies. It will check which other packages a specific project depends on and install them for you, using the appropriate versions according to the project requirements. This