

Using Composer



eZ Publish Platform 5.3 and higher

This page only applies to eZ Publish Platform 5.3 and higher, for earlier versions of the Enterprise version of eZ Publish consult *Service Portal user guide* available for download at support.ez.no. This page is also generic about using Composer, for instructions specific to a release, see [release notes](#).

Keeping your system up-to-date is important, to make sure your it is running optimally and securely. The update mechanism in eZ Publish Platform is using the *de facto* standard PHP packaging system called **Composer**.

This makes it easy to adapt package installs and updates to your workflow, allowing you to test new/updated packages in a development environment, put the changes in your version control system (git, Subversion, Mercurial, etc.), pull in those changes on a staging environment and, when approved, put it in production.

- [Installing Composer](#)
- [Prerequisite to using composer](#)
 - [Setting up Authentication tokens for access to commercial updates](#)
 - [Optional: Save authentication information in auth.json to avoid repeatedly typing it](#)
- [Using composer](#)
 - [# Running composer update and version changes](#)
 - [# Installing versioned updates on other development machines and/or staging](#)
 - [# Installing eZ packages via composer](#)
 - [# Notes on using Composer with Legacy](#)

Composer

Composer is an opensource PHP packaging system to manage dependancies.

This makes it easy to adapt package installs and updates to your workflow, allowing you to test new/updated packages in a development environment, put the changes in your version control system (git, Subversion, Mercurial, etc.), pull in those changes on a staging environment and, when approved, put it in production.

Installing Composer

This step is only needed once per machine (per project by default, but installing globally on the machine is also possible. For alternatives see: <http://getcomposer.org/download/>).

Composer is a command line tool, so the main way to install it is via command-line, example:

composer download in current folder:

```
php -r "readfile('https://getcomposer.org/installer');" | php
```

NB: this should be executed in the root directory of your eZ Publish installation.


Prerequisite to using composer

Setting up Authentication tokens for access to commercial updates

Out of the box composer uses a packaging repository called packagist.org to find all open source packages and their updates. Additional commercial packages are available for the eZ Publish Platform at updates.ez.no/bul/ (*which is password protected, you will need to set up authentication tokens as described below to get access*).

To get access to these updates log in to your service portal on support.ez.no. If your project is configured for eZ Publish Platform 5.3 or higher, you will see the following on the *"Maintenance and Support agreement details"* screen:

Authentication tokens [\(What's this?\)](#)

	Token label
<div>Create token Remove tokens</div>	

1. Click "Create token"
2. Fill in a label describing the use of the token. This will allow you to revoke access later
 - Example, if you need to provide access to updates to a third party a good to example would be "53-upgrade-project-by-partner-x"
3. Copy the password, **you will not get access to this again!**

When running composer to get updates, you will be asked for a Username and Password. Use:

as Username: your Installation key found higher up on the *"Maintenance and Support agreement details"* page in the support portal

as Password: the token password you retrieved in step 3.

Optional: Save authentication information in auth.json to avoid repeatedly typing it

To avoid having to always fill your credentials, add an `auth.json` file at your project root, next `composer.json`, or in your `COMPOSER_HOME` directory for machine-wide use. The file looks like the following:

auth.json

```
{
  "config": {
    "basic-auth": {
      "updates.ez.no": {
        "username": "<installation-key>",
        "password": "<token password>"
      }
    }
  }
}
```

The fine print: bundled with eZ Publish Platform is a [Composer extension](#) which makes it possible to store authentication information in that file.

Using composer

Running composer update and version changes

Updating eZ Publish Platform via composer is nothing different then [updating other projects via composer](#), but for illustration here is how you update your project locally:

composer update

```
php -d memory_limit=-1 composer.phar update --no-dev --prefer-dist
```



Tip

Tip: This will load in all updated packages, from eZ as well as third party libraries both used by eZ and others you may have added. So when updating like this it is recommended to take note of what was updated so you have an idea of what you should test before putting the updates into production.

When this has completed, make sure to version (assuming you use a version control system) changes done to `composer.lock` file. This is the file that contains **all details of which versions are currently used** and makes sure the same version is used among all developers, staging and eventually production when current changes are approved for production (assuming you have a workflow for this).



Tip

Tip2: In large development teams make sure people don't blindly update and install third party components, this might easily lead to version conflicts on `composer.lock` file and can be tiring to fix-up if happening frequently. A workflow involving composer install and unit test execution on proposed changes can help avoid most of this, like available via Github/Bitbucket Pull Request workflow.

Installing versioned updates on other development machines and/or staging

Installing eZ Publish Platform packages via Composer is nothing different then [installing vanilla packages via Composer](#), and for illustration here is how you install versioned updates:

composer install (package installation)

```
php -d memory_limit=-1 composer.phar install --no-dev --prefer-dist
```



Tip

Tip: Here the importance of `composer.lock` comes in as this command will tell composer to install packages in exactly same version as defined in `composer.lock`. If you don't keep track of `composer.lock` it will instead just install always latests version of a package, hence not allow you to stage updates before moving it towards production.

Installing eZ packages via composer

Also requiring eZ Publish Platform packages via composer is nothing different then [requiring vanilla packages via Composer](#), and for illustration here is how you install a eZ package:

composer install (package installation)

```
php -d memory_limit=-1 composer.phar require --prefer-dist ezsystems/ezfind-ls:5.3.*
```

Notes on using Composer with Legacy

(eZ Publish) Legacy by design places all important customizable folders within it's own structure. This is not ideal with Composer, as installation and updates might cause them to become as provided by packages again.

To make sure you are safe from this, and to allow you to version these custom folders independently, **it is recommended that you use symlinks** and keep your custom settings, extensions and design **outside of the `ezpublish_legacy` folder**. To not have to manually deal with these symlinks it is recommended to use Composer `post-install-cmd` and `post-update-command` script commands to make this automated.



Affected extensions

All extensions not provided via composer is affected by this, currently the following extensions from eZ is not provided via composer and needs to be setup like proposed in this section: *eznetwork*, *ezrecommendation* and *ezma*.

Below is a illustration on how this can be setup, see [Composer documentation](#) for further info.

Example on composer.json symlinking

```
{
  "scripts": {
    "post-install-cmd": [
      "MyVendor\\MyClass::symlinkLegacyFolders"
    ],
    "post-update-cmd": [
      "MyVendor\\MyClass::symlinkLegacyFolders"
    ]
  }
}
```



This functionality is desirable to have out of the box in the future, so community contributions on this topic is welcome to find a good standard convention and script to handle it.