Using Composer

Keeping your system up-to-date is important to make sure it is running optimally and securely. The update mechanism in eZ software 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, place the changes in your version control system (git, Subversion, Mercurial, etc.), pull in those changes to a staging environment and, when approved, put them in production.

> Composer is an opensource PHP packaging system to manage dependencies.
>
> 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

Composer is a command-line tool, so the main way to install it is via command line from inside the root directory of the (eZ) software:

Composer download in current folder:

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

By doing it this way you will need to execute further Composer commands using `php composer.phar`. If you'd rather prefer to install Composer globally on your machine instead of inside each and every project that uses it, then follow these instructions in online Composer documentation.

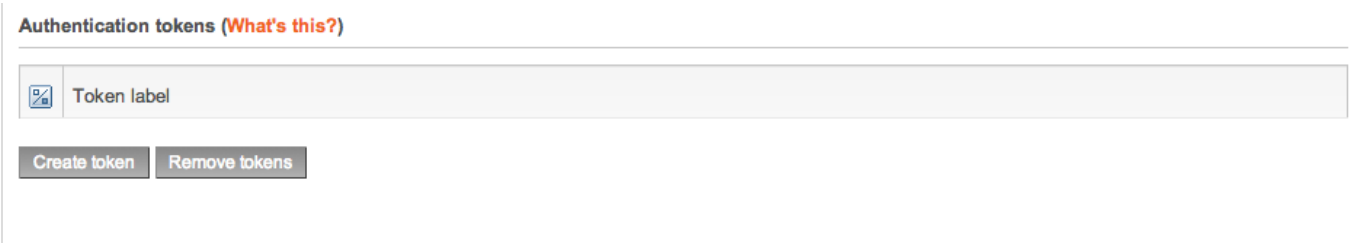
Prerequisite to using composer with eZ Enterprise software

This section describes features available only in eZ Studio.

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 eZ Enterprise subscribers 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 and look for the following on the *"Maintenance and Support agreement details"* screen:



1. Click "Create token" (This requires the "Portal administrator" access level.)
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 example would be "53-upgrade-project-by-partner-x"
3. Copy the password, **you will not get access to it again**!

After this, when running Composer to get updates as described below, you will be asked for a Username and Password. Use:

- as Username – your Installation key found above on the *"Maintenance and Support agreement details"* page in the service portal
- as Password – the token password you retrieved in step 3.

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

Composer will ask to do this for you on updates, however if it is disabled, you can create an `auth.json` file manually in one of the following ways:

Option A: Store your credentials in the project directory:

```
composer config http-basic.updates.ez.no <installation-key> <token-password>
```

Option B: If you'd rather want to install it globally in COMPOSER_HOME directory for machine-wide use:

```
composer config --global http-basic.updates.ez.no <installation-key> <token-password>
```

Update workflow Using Composer

This section describes the best practice for using Composer, essentially it suggests treating updates like other code/configuration/* changes on your project, tackling them on a development machine before staging them for rollout on staging/production.

1. Running composer update and version changes in development

Updating eZ software 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**
> This will load in all updated packages, from eZ as well as third-party libraries, both used by eZ and other you may have added. 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.

At this stage you might need to manually clear Symfony's `prod` environment class cache (cached interfaces and lazy services) in case the classes/interfaces in it have changed. This can be done in the following way:

Optional prod class cache clearing

```
rm -f app/cache/prod/*.php
```

When the update has completed and local install is verified to work, make sure to version changes done to the `composer.lock` file (if you use a version control system like *git*). This file 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
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` 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 the Pull Request workflow available via Github/Bitbucket.

2. Installing versioned updates on other development machines and/or staging -> production

Installing eZ software 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
Here the importance of `composer.lock` comes in, as this command will tell Composer to install packages in exactly the same version as defined in this file. If you don't keep track of `composer.lock`, it will instead just install always the latest version of a package and won't allow you to stage updates before moving towards production.

General notes on use of Composer

Installing additional packages via Composer

Requiring eZ software packages via Composer is also done in same way as requiring vanilla packages via Composer, and for illustration here is how you install the community-developed EzPriceBundle:

composer install (package installation)

```
php -d memory_limit=-1 composer.phar require --prefer-dist
ezcommunity/ez-price-bundle:~1.0.0@beta
```

Avoid: Dumping autoload

Avoid using the following command:

```
php -d memory_limit=-1 composer.phar dump-autoload --optimize
```

Warning

It causes a PHP warning "Ambiguous class resolution". For further information on this issue refer to Stash Github repository.

The dumped file will be too big and can cause an overhead for any request, even Cached requests.

Best practice for Bundles

Best practice for Bundles is described in Symfony documentation under Best Practices for Reusable Bundles, with eZ bundles there is some notable exceptions:

Documentation

- You may write your documentation using markdown (.md) if you prefer, however .rst is recommended if you plan to use writethedocs.org, as heavily used by many open source projects.

Git repository naming

- You may omit vendor name in repository naming, assuming vendor name is reflected in organization / user account it is attached to.
- You may also choose to follow composer package naming on repository name which is more relevant when trying to find a given package later.

Composer Metadata

For defining `"type"`, the following are at the moment known valid values:

- `ezplatform-bundle` | Symfony bundles that uses eZ Platform features
- `ezstudio-bundle` | Symfony bundles that uses eZ Studio features *(and optionally also eZ Platform features)*
- `symfony-bundle` | Standard symfony bundles as described in Symfony doc.

- For eZ Publish (legacy) and eZ Publish Platform there where also:
  - `ezpublish-legacy-extension` | For standalone 4.x (legacy) extensions, to be used with ezpublish-legacy-installer
  - `ezpublish-bundle | For eZ Publish Platform 5.x bundles, may optionally be a "legacy bundle".`