Upgrading Applications

If you have an existing Zend Framework v2 application, and want to update it to the latest versions, you will have some special considerations.

Upgrading Zend Framework

Since the 2.5 release, the zendframework package has been essentially a "metapackage", defining no code, and only dependencies on the various component packages. This means that when you install `zendframework/zendframework`, you get the full set of components, at the latest 2.* versions.

With the release of version 3, we recommend:

- Removing the zendframework/zendframework package.
- Installing the zendframework/zend-component-installer package.
- Installing the zendframework/zend-mvc package.
- Installing each ZF component package you actually use in your application.

The process would look like this:

```
$ composer remove zendframework/zendframework
$ composer require zendframework/zend-component-installer
$ composer require zendframework/zend-mvc
# Repeat as necessary for components you use if not already installed
```

When you install zend-mvc, it will prompt you to add configuration for components; choose either `application.config.php` or `modules.config.php`, and re-use your selection for all other packages. This step ensures that the various components installed, and any news ones you add later, are configured in your application correctly.

This approach will ensure you are only installing what you actually need. As an example, if you are not using zend-barcode, or zend-permissions-acl, or zend-mail, there's no reason to install them.

Keeping the zendframework package

If you want to upgrade quickly, and cannot easily determine which components you use in your application, you can upgrade your zendframework requirement. When you do, you should also install the zend-component-installer, to ensure that component configuration is properly injected in your application.

```
$ composer require zendframework/zend-component-installer "zendframework/zendframework:^3.0"
```

During installation, it will prompt you to add configuration for components; choose either `application.config.php` or `modules.config.php`, and re-use your selection for all other packages. This step ensures that the various components installed, and any news ones you add later, are configured in your application correctly.

This will upgrade you to the latest releases of all Zend Framework components at once; it will also install new components developed as part of the version 3 initiative.

We still recommend reducing your dependencies at a later date, however.

Integration packages

During the Zend Framework 3 initiative, one goal was to reduce the number of dependencies for each package. This affected the MVC in particular, as a number of features were optional or presented deep integrations between the MVC and other components. These include the following:

Console tooling

If you were using the MVC console tooling, and are doing a partial update per the recommendations, you will need to install zend-mvc-console (https://zendframework.github.io/zend-mvc-console/).

Forms integration

If you were using the forms in your MVC application, and are doing a partial update per the recommendations, you will need to install zend-mvc-form (https://zendframework.github.io/zend-mvc-form/).

i18n integration

If you were using i18n features in your MVC application, and are doing a partial update per the recommendations, you will need to install zend-mvc-i18n (https://zendframework.github.io/zend-mvc-i18n/).

Plugins

If you were using any of the `prg()`, `fileprg()`, `identity()`, or `flashMessenger()` MVC controller plugins, and are doing a partial update per the recommendations, you will need to install

zend-mvc-plugins (https://zendframework.github.io/zend-mvc-plugins/).

zend-di integration

If you were using the zend-servicemanager <-> zend-di integration within your application, you will need to install zend-servicemanager-di (https://zendframework.github.io/zend-servicemanager-di/).

Autoloading

If you are doing a partial upgrade per the above recommendations (vs. upgrading the full zendframework package), one change is that zend-loader is no longer installed by default, nor recommended. Instead, we recommend using Composer for autoloading (https://getcomposer.org/doc/04-schema.md#autoload).

As such, you will need to setup autoloading rules for each module specific to your application.

As an example, if you are still defining the default `Application` module, you can add autoloading for it as follows in your project's `composer.json`:

```
"autoload": {
    "psr-4": {
        "Application\\": "module/Application/src/Application/"
    },
    "files":  [
        "module/Application/Module.php"
    ]
}
```

The above creates a PSR-4 (http://www.php-fig.org/psr/psr-4/) autoloading rule for the `Application` module, telling it to look in

the `module/Application/src/Application/` directory. Since the `Application\Module` class is defined at the module root, we specify it in the `files` configuration.

To improve on this, and simplify autoloading, we recommend adopting a complete PSR-4 directory structure for your module class files. As an example, to change the existing `Application` module to PSR-4, you can do the following:

```
$ cd module/Application
$ mv src temp
$ mv temp/Application src
$ rm -Rf ./temp
$ mv Module.php src/
```

Update your `Module.php` file to do the following:

- Remove the `getAutoloaderConfig()` method entirely, if defined.
- Update the `getConfig()` method from
  `include __DIR__ . '/config/module.config.php` to
  `include _DIR__ . '/../config/module.config.php.`

You can then update the `autoload` configuration to:

```
"autoload": {
    "psr-4": {
        "Application\\": "module/Application/src/"
    }
}
```

Afterwards, run the following to update the generated autoloader:

```
$ composer dump-autoload
```

The updated application skeleton already takes this approach.

Bootstrap

Because version 3 requires usage of Composer for autoloading, you can simplify your application bootstrap.

First, if you were using an `init_autoloader.php` file, you can now remove it.

Second, update your `public/index.php` to read as follows:

```
<?php

use Zend\Mvc\Application;

/**
 * This makes our life easier when dealing with paths. Everything is relative
 * to the application root now.
 */
chdir(dirname(__DIR__));

// Decline static file requests back to the PHP built-in webserver
if (php_sapi_name() === 'cli-server') {
    $path = realpath(__DIR__ . parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH));
    if (__FILE__ !== $path && is_file($path)) {
        return false;
    }
    unset($path);
}

// Composer autoloading
include __DIR__ . '/../vendor/autoload.php';

if (! class_exists(Application::class)) {
    throw new RuntimeException(
        "Unable to load application.\n"
        . "- Type `composer install` if you are developing locally.\n"
    );
}

// Run the application!
Application::init(require __DIR__ . '/../config/application.config.php')->run();
```

Scripts

The skeleton application for version 2 shipped three scripts with it:

- `bin/classmap_generator.php`
- `bin/pluginmap_generator.php`
- `bin/templatemap_generator.php`

If you are upgrading an existing application, these will still be present. However, if you are starting a new application, and used these previously, they are no longer present.

- `classmap_generator.php` was removed as it's unnecessary when using Composer for autoloading. When preparing a production installation, run `composer dump-autoload -o` and/or `composer dump-autoload -a`; both will generate optimized class map autoloading rules for you.
- `pluginmap_generator.php` was essentially obsolete due to the presence of `classmap_generator.php` anyways.
- `templatemap_generator.php` was moved to the zend-view component with the 2.8.0 release of that component, and is now available via `./vendor/bin/templatemap_generator.php`. Additionally, its usage signature has changed; please use the `--help` or `-h` switches on first invocation to discover how to use it.

Development mode

Version 3 of the skeleton application adds a requirement on zfcampus/zf-development-mode (https://github.com/zfcampus/zf-development-mode), which provides a way to store common development-specific settings in your repository and then

selectively enable/disable them during development.

If you are upgrading from an existing application, you can install this feature:

```
$ composer require zfcampus/zf-development-mode
```

Please refer to the package documentation (https://github.com/zfcampus/zf-development-mode) for details on how to setup your application configuration to make use of this feature.

Copyright (c) 2016 Zend Technologies USA Inc. (http://www.zend.com/)

Learn more about Zend Framework (http://framework.zend.com)