Dependency injection and services: what exists

An honest answer to "where is the service container?" - the legacy kernel has none of its own; classes are found through a generated class map, singletons and handler classes named in INI settings; how to write testable...

Source file: `doc/eng-us/help/web-framework-developers/dependency-injection-and-services.md` Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/web-framework-developers/dependency-injection-and-services.md

This page is for developers who expect a service container: Symfony's DI component, Laravel's container and service providers, Spring's application context, Laminas' service manager. **The Exponential 6 kernel has no general-purpose dependency injection container.** It grew up before containers were common in PHP. What it has instead is described below, together with a way to write code that is as testable as container-wired code.

Contents

1. What the kernel uses instead
2. Handler classes named in settings
3. Writing services in an extension
4. Replacing an implementation
5. Inside Symfony
6. What to tell your team

What the kernel uses instead

Container feature	What the kernel does
Autoloading of services	a generated class map: `autoload/ezp_kernel.php` for the kernel, `var/autoload/ezp_extension.php` for extensions (`php bin/php/ezpgenerateautoloads.php -e`), plus Composer's autoloader
Shared services	static `instance()` methods: `eZDB::instance()`, `eZINI::instance( 'site.ini' )`, `eZHTTPTool::instance()`, `ezpEvent::getInstance()`, `eZTemplate::factory()`
Configurable implementations	handler classes named in INI settings, created by `eZExtension::getHandlerClass()`
Decorating or replacing a service	`site.ini [RunnableSettings] Implementation[]` for commands, cronjob parts and views; a kernel class override as the last resort
Tagged services, collectors	lists in INI settings (`Listeners[]`, `AvailableDataTypes[]`, `RuleProviders[]` ...)
Configuration parameters	`eZINI` values (Configuration)

Handler classes named in settings

Many kernel subsystems ask an INI setting which class to use, and `eZExtension::getHandlerClass()` creates it. The static cache, for example:

PHP

```php
$options = new ezpExtensionOptions( array( 'iniFile'     => 'site.ini',
                                           'iniSection'  => 'ContentSettings',
                                           'iniVariable' => 'StaticCacheHandler' ) );
$handler = eZExtension::getHandlerClass( $options );
```

This is service location by configuration: swapping the implementation is a settings change in your extension, not a code change in the kernel. The same mechanism is behind the session handler (`site.ini [Session] Handler`), the mail transport (`site.ini [MailSettings] TransportAlias[]`), the cluster file handler (`file.ini [ClusteringSettings] FileHandler`) and the search engine (`site.ini [SearchSettings] SearchEngine`); login handlers are named in `site.ini [UserSettings] LoginHandler[]` and found by directory. **Setup > RAD** lists every one of these extension points with its setting (Extension points).

Writing services in an extension

You can write container-style code without a container:

1. Put the work in plain classes with constructor parameters (`expDemoArchiveService`), not in views.
2. Give each a small static factory that builds it from settings, for production use: `expDemoArchiveService::fromSettings()` reads `eZINI` and passes the values to the constructor.
3. Views, commands, cronjob parts, fetch functions and queue handlers call the factory; tests call the constructor with fakes (Testing).
4. Keep no per-request state in static properties: under Velocity a process serves many requests (Servers and deployment).

The kernel's own services (`Exponential\Service\Trash`, `Exponential\Service\SessionGarbageCollector` in `kernel/private/classes/services/`) follow the "one class for view, command and cronjob part" rule, as static helpers (Services).

Composer libraries work in an extension, because `autoload.php` loads Composer's autoloader. If a library you use brings its own container, keep it inside your extension; the kernel will not see it.

Replacing an implementation

You want to	Use
Swap a kernel subsystem (sessions, mail, file storage, search, login)	its handler setting
Change one kernel command, cronjob part or view	`Implementation[<class>]=<your subclass>` in `[RunnableSettings]`

You want to	Use
Change any other kernel class	a kernel class override through the override autoload (`EZP_AUTOLOAD_ALLOW_KERNEL_OVERRIDE` and `php bin/php/ezpgenerateautoloads.php -o`): the heaviest mechanism, a last resort

Inside Symfony

When the kernel runs inside a Symfony platform installation through the legacy bridge, the web kernel can receive Symfony's service container (`ezpKernelWeb::hasServiceContainer()`, `getServiceContainer()`), and Symfony code around it uses dependency injection as usual. That is a property of the bridge setup, not of a stand-alone Exponential 6 installation (Symfony legacy bridge).

What to tell your team

- There is no container to register services in; wire your classes with factories and settings.
- Look for an existing handler setting before you subclass a kernel class.
- Never edit `kernel/` or `lib/`; an update would replace your change.

Related pages

- Extension points: handlers, INI registration and overrides
- INI plugin point pattern
- Singleton pattern
- Free documentation: this page is free to copy, translate and improve under the GNU FDL 1.3.

> **Become an Exponential Partner**
>
> Partners share their information, provide funding, contribute back and develop the products with us, and that is how Exponential grows. A partner website and dashboard are planned for mid-2027: please become a partner today to speed up our growth rate and land new features faster.
>
> Partners: start here | How to apply

Exponential Copyright © 1998-2026 Exponential Foundation and 7x
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
This documentation is free documentation, licensed under the GNU Free Documentation License 1.3 or later:
https://www.gnu.org/licenses/fdl-1.3.html.