# The kernel service container (6.0.15, first slice)

Source file: doc/eng-us/bc/6.0/kernel-services.md Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/bc/6.0/kernel-services.md

Read this page if you work on the kernel's boot, write code that asks the kernel for shared objects, run Exponential under Velocity, or deploy an installation. It describes what the first slice of the kernel service container does, what it changes, and how to check it.

## In short

| | |
|---|---|
| What changed | The kernel has a service container: shared objects by name (`expKernelServices::get( 'db' )`), defined in `settings/kernelservices.ini`, made on first use in a scope (worker, request, prototype). The web kernel boots it and, with a **compiled container** in `var/cache/kernelservices/`, applies the recorded result of its two extension activation passes instead of looking up every extension directory on every request. On Exponential Velocity the settings files of a siteaccess are built once per worker. |
| Who is affected | Every installation, without anything to do: the container is compiled by the deploy and by `ezcache.php --clear-tag=ini`, and records itself on a siteaccess's first web request otherwise. Code that resets or injects INI state keeps working. |
| What stays the same | Without a compiled container, with `[ContainerSettings] Mode=dev` or `Mode=disabled`, or whenever a request does not start from exactly the recorded state, the kernel boots exactly as before. Every `eZ*::instance()` call keeps working. |
| How to check | `./console exp:services:check` (exit 1 on an error), `./console exp:services:list`, `./console exp:services:show <id>` |
| How to fix a stale container | `./console exp:services:compile`, or `php bin/php/ezcache.php --clear-tag=ini --allow-root-user` |

Not to be confused with `ezpKernelWeb::getServiceContainer()` / the `'service-container'` kernel setting, the injected Symfony container of the old legacy bridge, which keeps its meaning, nor with the `expservices` extension.

## Why

The web kernel's constructor did the same wiring on every request: `eZExtension::activateExtensions( 'default' )` looked up each of the 62 active extensions of this installation on disk (a directory scan per extension), the siteaccess was matched, the access extensions were activated, and then every settings object built so far was thrown away and built again. Measured before this change (phase marks in a copy of the constructor): the constructor took 17 to 23 ms of a fresh request, the default extension pass alone 8 to 17 ms. On Velocity none of it survived a request, because the pool puts every static back and removes every global after each request.

## What it does

### The compiled container is a recording, not an imitation

`exp:services:compile` starts a separate PHP process (`bin/php/services.php --kernel-services-record`) that boots the real web kernel once per siteaccess (`site.ini [SiteAccessSettings] AvailableSiteAccessList`), in recording mode, as the first request of a fresh process would. For each of the two passes of `eZExtension::activateExtensions()` it records:

- the global INI override directories **before** the pass,
- what the pass reads besides them (`[ExtensionSettings] ActiveExtensions`, `ExtensionDirectory`, `AdditionalExtensionDirectories`, `ExtensionOrdering`, and for the access pass `ActiveAccessExtensions`, plus the current siteaccess),
- the override directories **after** the pass, and the extension name, path and active-list caches.

It writes `var/cache/kernelservices/index.php` (the default pass, the same for every siteaccess, and a new **generation**) and one `<siteaccess>.php` per siteaccess (its access pass, and the service definitions and tags of `kernelservices.ini` as that siteaccess sees them). Each file is an array returned by `return [...]`, served from the opcode cache.

### A request applies a recording only from exactly the recorded state

`expKernelServices::bootWeb()` (called by the `ezpKernelWeb` constructor before it reads any setting) reads the index. `eZExtension::activateExtensions()`, when called for the global `site.ini` with a booted container, first asks `expKernelServices::applyExtensionStage()`. It applies the recording (sets the override directories, fills the extension caches, reloads `site.ini`) **only when** the override directories before the pass and the settings the pass reads are exactly the recorded ones, no extension was added at runtime (`$GLOBALS['eZActiveExtensions']`) and no settings were injected into the kernel. Otherwise it answers no, and the pass runs as it always did.

So a stale container cannot change what a request sees: a changed `ActiveExtensions`, another extension directory, an injected setting or an unusual call order all fall back to the live pass. What a recording cannot see is the disk: an extension directory added or removed while the settings stayed the same. That is what the deploy is for (it compiles again), as it is for the autoload maps.

### AutoRebuild: a siteaccess records itself

With `[ContainerSettings] AutoRebuild=enabled` (the default), a web request whose siteaccess has no current compiled file runs the passes live, records them, and writes the file (and a new index when the default pass had to be recorded too). The next request of that siteaccess applies it. This covers the first request after an install and an INI cache clear from the administration, which removes the compiled files but does not start PHP processes. A file that exists but does not match a request is left alone (only `exp:services:compile` replaces it), so two kinds of requests can never overwrite each other's recording in turn. Files are written to a temporary name and renamed; a file written by root (Velocity) is given to the owner of `var/`, so PHP-FPM can replace it later. Command-line kernels (tests, scripts) never write.

### The worker scope of the settings files (Velocity)

With `[ContainerSettings] WorkerScope=auto` (the default) and `EZP_INI_FILEMTIME_CHECK` false in `config.php` (the INI cache does not look at file times either), the container attaches a store to `eZINI` (`eZINI::attachSharedStore()`) at two moments: at boot, while the override directories are the defaults, and right after the constructor's `eZINI::resetAllInstances( false )`, when the access pass was applied from the recording. While attached, `eZINI::instance()` keeps a pristine copy of every settings object it builds and hands out copies afterwards; a request that changes its object changes only its copy.

The store is one object in `$GLOBALS['expKernelServicesWorker']` (`expKernelServiceScope::WORKER_GLOBAL`), which `expVelocity::DEFAULT_KEEP_GLOBALS` keeps across requests. So a Velocity worker builds each settings file of a siteaccess once, instead of on every request. On PHP-FPM a worker is one request long and nothing changes.

The store is detached (the next `instance()` reads the files again) by any change of the global override directories (`prependOverrideDir()`, `appendOverrideDir()`, `removeOverrideDir()`, `setOverrideDirs()` on a global instance, `resetGlobalOverrideDirs()`), by `resetAllInstances()`; `resetInstance()` drops that file's copies; `save()` and the INI cache clear empty it. A new generation (a compile, a recorded index) empties the worker's store on its next request. Keyed by siteaccess and the hash of the override directories: one worker serving several siteaccesses (host_uri matching) never hands one site's settings to another.

### Request scope: the seven resets in one place

The constructor reset seven request states by hand. They are now the request scope's reset hooks, `expKernelServiceScope::DEFAULT_RESET_HOOKS`, run by `expKernelServiceScope::resetRequest()` at the same place, in the same order, each only when its class exists, as before: `eZDBQueryCache::resetRequest`, `eZDBInterface::resetSQLProfile`, `expContentSecurityPolicy::reset`, `expAudit::resetRequest`, `expRenderStats::resetRequest`, `expRequestLog::reset`, `expRequestClassifier::reset`, and since 6.0.15 `expUserApproval::resetRequest` (the "Approve user accounts" configurations, which a worker kept before). Code adds its own with `expKernelServiceScope::onRequestReset( 'name', $callable )` (registering a name again replaces it; kept with the worker, so it also runs at the start of the next request on Velocity).

## The services

`settings/kernelservices.ini`, one `[Service_<id>]` block per service:

| Setting | Meaning |
|---|---|
| `Class` | the class (or interface) of the service |
| `Factory` | `Class::method` to call; without it, `new Class( Arguments )` |
| `Arguments[]` | `@<id>` another service, `%ini:<file>/<Section>/<Variable>%` a setting, anything else as written (`@@`, `%%` escape a leading `@` or `%`) |
| `Scope` | `worker` (one per worker and siteaccess; must hold nothing of a request), `request` (the default), `prototype` |
| `Lazy` | `true` (the default): made when first asked for |
| `Tags[]` | names that collect services (`expKernelServices::tagged( '<tag>' )`) |
| `Description` | one line for `exp:services:list` / `show` |

| Setting | Meaning |
| --- | --- |
| Enabled | `false` switches a service off |

Shipped: `db` (`eZDB::instance()`, request), `http` (`eZHTTPTool::instance()`, request), `ini.site` and `ini.i18n` (documentation only: every `ini.<name>` is `eZINI::instance( '<name>.ini' )`), `extensions.overrides` (`expKernelExtensionOverrideList`, worker: the compiled override directories and extension paths of the siteaccess, and whether this request applied them). `[Tags] TagList[]` lists the tags; `exp:services:check` warns about others.

**PHP**

```php
$db = expKernelServices::get( 'db' );
$site = expKernelServices::get( 'ini.site' );
foreach ( expKernelServices::tagged( 'kernel.core' ) as $id => $service ) { /* ... */ }
expKernelServices::set( 'my.thing', $object );   // this request only, never compiled or kept
```

An extension adds services in `extension/<name>/settings/kernelservices.ini.append.php`. Run `./console exp:services:compile` (or deploy) afterwards; `exp:services:check` validates the definitions of every siteaccess: unknown scope, missing class or factory, an argument naming no service, cycles, a worker-scoped service taking a request-scoped one (it would keep one request's object for every later request), an `ini.<name>` service that is not `eZINI`, undeclared tags.

## Settings

**INI**

```ini
[ContainerSettings]
Mode=compiled          # compiled (default) | dev (nothing compiled is used) | disabled (no part
in the boot)
CacheDir=var/cache/kernelservices
AutoRebuild=enabled    # a web request records a siteaccess that has no current compiled file
WorkerScope=auto       # auto (on when EZP_INI_FILEMTIME_CHECK is false) | enabled | disabled
```

`[ContainerSettings]` is read before the extensions are active: set it in `settings/kernelservices.ini` or `settings/override/kernelservices.ini.append.php`.

## Console

**Shell**

```shell
./console exp:services:compile [--only=site,admin]   # record every siteaccess (or these) and wri
te the container
./console exp:services:list [--tag=<tag>] [-s <siteaccess>]
./console exp:services:show <id> [-s <siteaccess>]
./console exp:services:check                         # definitions + compiled files; exit 1 on an
 error
```

All four are `bin/php/services.php` (`exp:services`), with the verb taken from the name it was called by; `--json` prints the result as JSON.

## When it is rebuilt

| Event | What happens |
|---|---|
| `exp:velocity deploy` | step 3 (`ezcache.php --clear-tag=ini`) removes and compiles it; the new generation reaches every Velocity worker on its next request |
| `php bin/php/ezcache.php --clear-tag=ini` or `--clear-all` | the `kernelservices` cache item (tag `ini`, last of the INI caches) removes it and, on the command line, compiles it again |
| INI cache cleared in the administration | the files are removed; each siteaccess records itself on its next request (AutoRebuild) |
| `./console exp:services:compile` | compiled again |
| A settings file edited by hand | as for the INI cache today: clear the INI cache (with `EZP_INI_FILEMTIME_CHECK` false, nothing notices the edit otherwise) |

## Deploying this change

The slice adds kernel classes: deploy with `exp:velocity deploy --kernel --allow-root-user` (kernel autoloads, engine archive, PHP-FPM, Velocity restart). A Velocity worker that still runs with an older class map does not know the new classes; the constructor then loads the request scope class by path and leaves the container out (`class_exists()` guards), so a stale worker behaves as before until the restart.

## Measured

On the test installation (62 active extensions, 18 siteaccesses, SQLite), 21 October 2026, host load 4 to 8 on 12 cores, `nice -n 15`. Five requests (a cached and an uncached public page, the admin dashboard, a large help page, a JSON call) went through the kernel in the CLI with phase marks in a copy of the constructor, in two throwaway installations next to the live site that differ only in this change: the code before it, and the code with it and a compiled container. Medians of 10, before and after alternated.

- **PHP-FPM-like**: one kernel per PHP process, OPcache file cache (compiled once).
- **Velocity-like**: one process, 11 requests in a row, with what Velocity's pool does between requests emulated (every static back to its snapshot or declared default, every global removed except the keep list, a new request time); the first request left out.

Constructor = `ezpKernelWeb::__construct()`; boot = process start to the module run (constructor, request init, pre-dispatch).

| Request | PHP-FPM constructor | PHP-FPM boot | Velocity constructor | Velocity boot |
|---|---|---|---|---|
| cached page / | 25.6 -> 13.4 ms (-48%) | 49.4 -> 34.2 ms (-31%) | 22.3 -> 5.4 ms (-76%) | 30.8 -> 14.0 ms (-55%) |
| uncached page | 21.6 -> 13.1 ms (-40%) | 41.9 -> 35.6 ms (-15%) | 19.8 -> 5.1 ms (-74%) | 27.4 -> 13.5 ms (-51%) |
| admin dashboard | 23.1 -> 12.6 ms (-46%) | 38.3 -> 26.5 ms (-31%) | 18.2 -> 4.7 ms (-74%) | 20.8 -> 6.8 ms (-67%) |
| help page | 30.0 -> 13.9 ms (-54%) | 49.5 -> 32.3 ms (-35%) | 22.9 -> 5.1 ms (-78%) | 26.3 -> 7.5 ms (-72%) |

| Request | PHP-FPM constructor | PHP-FPM boot | Velocity constructor | Velocity boot |
|---|---|---|---|---|
| JSON call | 25.8 -> 12.9 ms (-50%) | 41.8 -> 27.8 ms (-34%) | (ends the process: not measurable this way) | |

The largest single saving is the default extension pass (8 to 17 ms before, about 0.5 ms after: no directory lookups). On Velocity the kept settings files remove most of the rest of the constructor. Whole requests: a few percent on PHP-FPM; on Velocity the emulated requests were also faster to render (the cached page 272 -> 206 ms total), but those totals vary with the content and are not claimed here.

A Velocity regression pass in the same way: one emulated worker served `/`, `/bold`, `/classic`, `/starter`, `/admin/user/login`, the editor host's sign-in, `/Fit-Healthy`, `/bold/user/login` and a query-string page twice over; each page matched the page a fresh process rendered for the same request, with the right siteaccess per request (site, bold, classic, starter, admin, editor) and both passes applied. The one difference, `/starter`, is a teaser chosen at random and differed the same way before the change.

Over HTTP before the change (from another host, medians of 10): an uncached public page 176 ms server time on PHP-FPM and 178 ms on Velocity, the admin dashboard 93 and 161 ms. The HTTP numbers after can only be taken once the change is deployed.

## Classes

| Class | File | What |
|---|---|---|
| `expKernelServices` | `kernel/classes/expkernelservices.php` | the facade (`get`, `has`, `tagged`, `set`, `reset`) and the web kernel's compiled boot |
| `expKernelServiceContainer` | `kernel/classes/expkernelservicecontainer.php` | definitions, scopes, arguments; no reflection at runtime |
| `expKernelServiceScope` | `kernel/classes/expkernelservicescope.php` | worker / request / prototype, the worker store, the request reset hooks |
| `expKernelServiceCompiler` | `kernel/classes/expkernelservicecompiler.php` | definitions from INI, validation, recording, compiled files, the cache item |
| `expKernelExtensionOverrideList` | `kernel/classes/expkernelextensionoverridelist.php` | the `extensions.overrides` service |
| `Exponential\Command\Kernel\Services` | `kernel/private/classes/commands/services.php` | `exp:services:*` |

Hooks in existing code: `eZINI` (`attachSharedStore()`, `detachSharedStore()`, `releaseSharedStore()`, `flushSharedStore()`, `sharedStoreState()`; `instance()` uses the store while attached), `eZExtension` (`activateExtensions()` asks the booted container; `exportCaches()`, `seedCaches()`, `clearCaches()`), `ezpKernelWeb::__construct()` (request scope, `bootWeb()`, `enterSiteAccess()`), `eZCache` (item `kernelservices`), `expVelocity::DEFAULT_KEEP_GLOBALS` (`expKernelServicesWorker`), `expCacheCatalogue`.

## Not in this slice

The console's own start-up (a compiled command map, commands run in the console's process), lazy event listeners and mobile detection, template operator tags, and worker-scoped design and override lookups on Velocity are the next candidates, in that order of measured value. `eZScript` (the command line) does not boot the container yet.

## Tests

`tests/tests/kernel/classes/expKernelServicesTest.php` (DB-less, suite `kernel-classes`): definitions from INI, validation (and that the shipped `kernelservices.ini` has no error), scopes, arguments and errors, the request reset hooks and their order, eZINI's worker store (attach, copies, detach on every change, flush), compiled files (stamp, generation, format), `applyExtensionStage()` applying only from the recorded state, the kernel wiring, and that nothing changes without a booted container.

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.