

Environment

Environment

Anonymous · 10/10/2026

Environment

Environment Variables and the `.env` Files

The variables of the application come from the real environment (the process, the web server) and from the `.env` files of the project. `Environment` reads the files **when it is created**, before the container is built and before the kernel boots, from the project directory (the one with the `vendor/` of Composer).

The files

They are loaded in this order, by the name of the environment of the kernel (`dev`, `prod`, `test...`), and **the last one that has a variable wins**:

Order	File	Use
1	<code>.env</code>	The defaults of the project. Committed.
2	<code>.env.<environment></code>	The values of one environment (<code>.env.prod</code> , <code>.env.test...</code>). Only the one of the environment of the kernel is read: <code>prod</code> never reads <code>.env.dev</code> .
3	<code>.env.<environment>.local</code>	The values of one environment on this machine. Not committed.
4	<code>.env.local</code>	The values of this machine, for every environment. Not committed.

- In **test** the `.local` files are not loaded (3 and 4), so the machine of whoever runs the tests does not change their result.
- A file that does not exist is skipped. Nothing is read from `.env.dist` or similar names.
- **A variable of the real environment is never replaced by a file**, but see the note below about `variables_order`.
- A variable that two files define has the value of the later one (`.env.local` over `.env`).

Only the files in the table are loaded. The environment that decides which ones is the **name of the kernel's environment**, not the `APP_ENV` variable.

APP_ENV

The kernel does not define `APP_ENV`: if nobody defines it, it stays undefined. The name of the environment of a web application is decided before the files are read:

`Derafu\Http\Runtime::getApplicationContext()` takes `APP_ENV` from `$_SERVER` and `$_ENV` (the real environment) and uses `dev` when it is not there, and that is the name that `new Environment($context['APP_ENV'], ...)` receives. So `APP_ENV=prod` written in a `.env` does not make the kernel `prod`: define it in the real environment (the web server or the container).

What the kernel does with them

- **Parameters:** every variable that is in `$_ENV` after reading the files becomes the parameter `env.<NAME>`. They are dumped to the [cache of the container](#) with it, so a change in a `.env` is not seen in production until the cache is cleared.
- **%env()% in the configuration:** `%env(NAME)%` and the typed forms (`%env(bool:NAME)%`, `int`, `float`, `json`, `string`, `default:...`) are read **when a service that uses them is created**, from `$_ENV`, `$_SERVER` and the process. That can be long after boot, when a page needs the service for the first time.
- **Environment::getEnv(\$name, \$default):** the same, with an optional type (`getEnv('bool:APP_DEBUG')`).
- **The super-globals:** reading the files writes the variables to `$_ENV` and `$_SERVER`. A test that checks the global state has to build the kernel before it takes its photo (see [testing a site](#)).

variables_order

The files do not replace the variables that are in `$_ENV` or `$_SERVER`, and those are the only places that are looked at. If the `variables_order` of PHP has no `E` (for example `GPCS`), `$_ENV` is empty, and a variable that exists only in the process environment (`getenv()`) is not seen: the value of the file is used. Under a web server the variables usually are in `$_SERVER`, so the real value wins; on the command line it depends on the configuration of PHP.

Environment Types

The kernel supports multiple environment types through constants in `EnvironmentInterface`:

- `LOCAL`: Local development environment.
- `DEVELOPMENT`: Development environment.
- `TEST`: Testing environment.
- `STAGING`: Staging environment.
- `QUALITY_ASSURANCE`: QA environment.
- `PREPRODUCTION`: Pre-production environment.
- `PRODUCTION`: Production environment.