

Docs

Deployer Project

PHP Deployer for multiple sites

Anonymous · 13/10/2026 · #php

Table of Contents

Deployer Project 3

Introduction 4

Configuration 6

Usage and Deployment 10

Docs » System Administration Category » Deployer Project

Deployer Project

PHP Deployer for multiple sites

Anonymous · 13/10/2026 · #php

PHP Deployer for multiple sites

Last updated on 13/10/2026

Docs » System Administration Category » Deployer Project » Introduction

Introduction

Deployment tool built on PHP Deployer to manage the deployments of several websites on a single server with shell actions

Anonymous · 13/10/2026

PHP Deployer for multiple sites

github	derafu/deployer	last commit	march	code size	17.3 KiB	open issues	0	downloads	2
downloads	1 this month								

Derafu Deployer is a PHP deployment tool built on top of [Deployer](#) that simplifies managing deployments for multiple websites on a single server.

Features

- Deploy multiple sites from a single configuration.
- Deploy individual sites or all sites at once.
- Support for different deployment environments (development, production).
- Shared files and directories between releases.
- Writable directories configuration.
- Asset building support for sites with Node.js/npm.
- Run custom commands during the deployment.
- OPcache reset after deployments.
- Simple and flexible configuration.
- Support for different Git branches per site.

Requirements

- PHP 8.4 or higher.
- SSH access to your servers.
- Git repositories for your projects.

Installation

```
composer create-project derafu/deployer
```

Note: The tool is designed to be used standalone, not inside other project.

Last updated on 13/10/2026

Configuration

Configuration

Anonymous · 13/10/2026

Configuration

The configuration can be stored in multiple files:

- `sites.php`: Legacy file and currently **deprecated**.
- `config/sites.yaml`: New file for the sites configuration when you need only one environment.
- `config/ABC.sites.yaml`: File for separated environments sites configuration.

The ABC is called the *source* of the configuration. For example, `dev.sites.yaml` for the development environment, `prod.sites.yaml` for the production environment, etc.

Sites Configuration

Create a YAML file and configure the sites you want to deploy using the following structure in each file:

- A key with the site/domain name and a value with the repository URL.
- A key with the site/domain name and a value with an array of detailed configuration options.

For example, create the file `config/sites.yaml` with the following content:

```
# Simple configuration with just the repository URL.
www.example.com: git@github.com:example/example.git

# Extended configuration with options.
www.complex-site.com:
  repository: git@github.com:example/complex-site.git
  branch: dev
  deploy_path: /var/www/custom/path/complex-site
  shared_files: ['.env', 'config/settings.php']
  shared_dirs: ['var/uploads', 'var/logs']
  writable_dirs: ['var', 'tmp', 'var/cache']
  writable_mode: chmod
  writable_use_sudo: false
  writable_recursive: true
  writable_chmod_mode: 0775
```

Add a site to the configuration via CLI

For simple configurations you can use:

```
./site-add.sh www.example.com git@github.com:example/example.git
```

This will always add the configuration to the `config/sites.yaml` file.

Using sources

You can use sources to create a configuration for multiple environments. For example, you can create a source for the development environment and a source for the production environment.

- `config/dev.sites.yaml`: Development environment configuration.
- `config/prod.sites.yaml`: Production environment configuration.

This file can have the same sites, but with different configuration options. Then you can use the `--source` option to specify the source of the configuration to select the appropriate environment for the site.

Available Configuration Options

Option	Description	Default
repository	Git repository URL	<i>Required</i>
branch	Git branch to deploy	main
deploy_path	Deployment path on server	<code>/var/www/sites/[site-name]</code>
shared_files	Files to share between releases	<code>[]</code>
shared_dirs	Directories to share between releases	<code>[]</code>
writable_dirs	Directories to make writable	<code>['var', 'tmp']</code>
writable_mode	Mode for writable directories	chmod
writable_use_sudo	Whether to use sudo for writable directories	false
writable_recursive	Apply writable permissions recursively	true
writable_chmod_mode	Chmod mode for writable directories	0777

Deployer Server Configuration

The server configuration is defined in the file `deploy.php`. By default, a local environment (localhost) and a remote environment (if `DEPLOYER_HOST` is set) are configured:

```
// Default local environment.
host('localhost')
    ->setRemoteUser('admin')
    ->setPort(2222)
    ->setLabels(['stage' => 'local']);

// Remote environment (only if DEPLOYER_HOST is set).
if (getenv('DEPLOYER_HOST')) {
    $stage = getenv('DEPLOYER_STAGE') ?: 'prod';
    host(getenv('DEPLOYER_HOST'))
        ->setRemoteUser(getenv('DEPLOYER_USER') ?: 'admin')
        ->setPort(getenv('DEPLOYER_PORT') ?: 2222)
        ->setLabels(['stage' => $stage]);
    set('default_selector', 'stage=' . $stage);
}
```

You can modify these settings or add additional environments as needed.

We recommend to use the environment variables and not to hardcode the values in the file `deploy.php`.

Variable	Description	Default
DEPLOYER_HOST	Remote host	
DEPLOYER_USER	Remote user	admin
DEPLOYER_PORT	Remote port	2222
DEPLOYER_STAGE	Environment stage	prod

Actions (shell scripts)

Actions allow you to run custom commands during the deployment. For example, you can run a custom command to rebuild the cache after the deployment.

There are three types of actions:

- `initial`: Run initial actions before the deployment, just after the code is updated.
- `final`: Run final actions after the deployment, just before the symlink is created.
- `success`: Run success actions after the deployment, just before the success message.

The actions are defined in the `.deployer/actions` directory. Each action is a Shell script file:

- `.deployer/actions/initial.sh`: Initial actions script.
- `.deployer/actions/final.sh`: Final actions script.
- `.deployer/actions/success.sh`: Success actions script.

All the actions are optional, the deployment will continue if the action file is not present.

Last updated on 13/10/2026

Docs » System Administration Category » Deployer Project » Usage and Deployment

Usage and Deployment

Usage and Deployment

Anonymous · 13/10/2026

Usage and Deployment

Usage

List Available Sites

To see the list of configured sites and usage information use `derafu:sites:list` command.

```
vendor/bin/dep derafu:sites:list
```

You can filter the list of sites by source using the `--source` option.

```
vendor/bin/dep derafu:sites:list --source=prod
```

Deploy a Single Site

Local Environment (localhost)

```
vendor/bin/dep derafu:deploy:single --site=www.example.com
```

Remote Environment

This usually is used for production environments, but can be used for any remote environment.

```
DEPLOYER_HOST=hosting.example.com vendor/bin/dep derafu:deploy:single --  
site=www.example.com
```

You can also specify the SSH user and port with the environment variables `DEPLOYER_USER` and `DEPLOYER_PORT`.

```
DEPLOYER_HOST=hosting.example.com DEPLOYER_USER=deployuser DEPLOYER_PORT=22  
vendor/bin/dep derafu:deploy:single --site=www.example.com
```

You can also specify the source of the configuration to select the appropriate environment for the site.

```
DEPLOYER_HOST=hosting.example.com DEPLOYER_USER=deployuser DEPLOYER_PORT=22  
vendor/bin/dep derafu:deploy:single --source=prod --site=www.example.com
```

Deploy All Sites

You can deploy all sites of the configuration in deployer, or only the sites of a specific source. It's similar to the deploy a single site, but using `derafu:deploy:all`.

Local Environment (localhost)

```
vendor/bin/dep derafu:deploy:all
```

For a specific source:

```
vendor/bin/dep derafu:deploy:all --source=prod
```

Remote Environment

```
DEPLOYER_HOST=hosting.example.com vendor/bin/dep derafu:deploy:all
```

For a specific source:

```
DEPLOYER_HOST=hosting.example.com vendor/bin/dep derafu:deploy:all --source=prod
```

And you can also specify the SSH user and port with the environment variables `DEPLOYER_USER` and `DEPLOYER_PORT`.

Unlock a Deployment

If a deployment gets stuck or locked, you can unlock it `--unlock` option.

```
DEPLOYER_HOST=hosting.example.com vendor/bin/dep derafu:deploy:single --  
site=www.example.com --unlock
```

Using aliases

You can create aliases for easy deployment. For example:

```
DEPLOYER_DIR="$HOME/dev/php/deployer"  
alias site-deploy-stage="cd $DEPLOYER_DIR && DEPLOYER_HOST=example.com  
DEPLOYER_USER=admin DEPLOYER_PORT=2223 vendor/bin/dep derafu:deploy:single --site"  
alias site-deploy-prod="cd $DEPLOYER_DIR && DEPLOYER_HOST=example.com  
DEPLOYER_USER=admin DEPLOYER_PORT=2224 vendor/bin/dep derafu:deploy:single --site"
```

Also, you can specify the source of the configuration to select the appropriate environment for the site.

```
DEPLOYER_DIR="$HOME/dev/php/deployer"
alias site-deploy-stage="cd $DEPLOYER_DIR && DEPLOYER_HOST=example.com
DEPLOYER_USER=admin DEPLOYER_PORT=2223 vendor/bin/dep derafu:deploy:single --
source=dev --site"
alias site-deploy-prod="cd $DEPLOYER_DIR && DEPLOYER_HOST=example.com
DEPLOYER_USER=admin DEPLOYER_PORT=2224 vendor/bin/dep derafu:deploy:single --
source=prod --site"
```

You can use the aliases to deploy the site to the appropriate environment.

```
site-deploy-stage www.example.com
site-deploy-prod www.example.com
```

Deployment Process

For each site, the deployment process performs the following steps:

1. **Info** (deploy:info): Shows information about the deployment.
2. **Setup** (deploy:setup): Sets up the deployment environment.
3. **Check Remote** (deploy:check_remote): Verifies SSH connection and deployment path.
4. **Lock** (deploy:lock): Locks the deployment.
5. **Release** (deploy:release): Creates a new release.
6. **Update Code** (deploy:update_code): Fetches code from the Git repository.
7. **Initial Actions** (deploy:initial_actions): Runs custom commands in the initial of the deployment.
8. **Environment** (deploy:env): Sets up the environment variables.
9. **Shared Files/Dirs** (deploy:shared): Links shared files and directories.
10. **Writable Dirs** (deploy:writable): Makes specified directories writable.
11. **Vendors** (deploy:vendors): Installs PHP dependencies with Composer.
12. **Assets** (deploy:assets): Builds frontend assets if package.json exists (`npm install && npm run build`).
13. **Final Actions** (deploy:final_actions): Runs custom commands in the final of the deployment.
14. **Symlink** (deploy:symlink): Creates a symlink to the new release.
15. **Unlock** (deploy:unlock): Removes the deployment lock.
16. **Cleanup** (deploy:cleanup): Removes old releases (keeps 5 by default).
17. **OPcache Reset** (opcache:reset): Resets the OPcache.
18. **Success Actions** (deploy:success_actions): Runs custom commands in the success of the deployment.
19. **Success Message** (deploy:success): Shows the success message.

Last updated on 13/10/2026