1v0 Ship v1.0 Faster

Automating Laravel Deployments with Deployer

Automating Laravel Deployments with Deployer

Manual deployments invite mistakes: forgotten cache clears, missing assets, downtime during uploads. **Deployer** is a PHP deployment tool that automates releases with a safe, repeatable, *zero-downtime* flow. In this guide you'll install Deployer, configure hosts, define tasks (Composer, Vite build, migrations, cache), and wire in a tiny UI to display the current deployed version.

1 — Install Deployer

You can install Deployer globally with Composer or use its PHAR. Composer keeps it versioned alongside your toolchain.

```
# Install Deployer globally
composer global require deployer/deployerCode language: Bash (bash)
```

This makes the `dep` command available on your machine (ensure `~/.composer/vendor/bin` or `~/.config/composer/vendor/bin` is on your PATH). Alternatively, download the PHAR and run `php dep.phar`.

2 — Initialize in Your Project

Deployer can scaffold a Laravel-ready config. Run this inside your project root to create

`deploy.php`:

`dep init --template=Laravel`Code language: Bash (bash)

The generated `deploy.php` contains common tasks (Composer install, symlink, caches). You'll customize hosts, shared files, writable directories, and hooks next.

3 — Configure Hosts & Paths

Tell Deployer where to SSH and where releases live on the server. Use a non-root user (e.g., `deploy`) with key-based auth.

```
// deploy.php (excerpt)
namespace Deployer;

require 'recipe/laravel.php';

set('application', 'your-app');
set('repository', 'git@github.com:your-org/your-repo.git');
set('keep_releases', 5);

host('your-server.com')
    ->set('remote_user', 'deploy')
    ->set('deploy_path', '/var/www/your-app');
```
Code language: PHP (php)

`deploy_path` will contain `releases/`, `shared/`, and a symlink `current`. Each deployment creates a timestamped release and atomically switches `current` to the new one for zero-downtime.

4 — Shared Files & Writable Directories

Persist environment files and user data across releases. Mark storage/cache as writable by the web server user.

```
// deploy.php (continued)
add('shared_files', ['.env']);
add('shared_dirs',  ['storage']);
add('writable_dirs', ['storage', 'bootstrap/cache']);
set('writable_mode', 'chmod'); // or 'acl' if setfacl is availableCode language: PHP (php)
```

Deployer creates symlinks from the new release to `shared/`, so uploads and logs survive code updates. Ensure ownership/permissions (e.g., `www-data`) are correct on the server.

5 — Composer, Vite Build & Optimizations

Hook a build step after Composer, then warm Laravel caches for faster boot times.

```
// deploy.php (build & optimize hooks)
task('build', function () {
    run('cd {{release_path}} && npm ci --no-audit --no-fund');
    run('cd {{release_path}} && npm run build');
});

after('deploy:vendors', 'build');
```

```
task('artisan:optimize', function () {
    run('cd {{release_path}} && php artisan config:cache && php
artisan route:cache && php artisan view:cache');
});

after('artisan:migrate', 'artisan:optimize');Code language: PHP (php)
```

`npm ci` ensures reproducible installs; `npm run build` compiles your assets (Vite). Running Laravel's cache commands after migrations keeps the app snappy on first request.

6 — Database Migrations & Queues

Migrate safely during deploy, then restart workers so they load new code. If you use Horizon, call its artisan command.

```
// deploy.php (migrate & queues)
after('deploy:symlink', 'artisan:migrate'); // Deployer's recipe uses
--force in prod

task('queue:restart', function () {
    run('cd {{release_path}} && php artisan queue:restart');
});
after('deploy:symlink', 'queue:restart');Code language: PHP (php)
```

`artisan migrate` runs with `--force` in production to apply schema changes. `queue:restart` signals workers to gracefully restart and pick up the new code without killing in-flight jobs.

7 — Expose Version & Rollbacks

Write the current Git commit to a file during deploy, show it in an admin page, and keep a few releases for quick rollbacks.

```
// deploy.php (version stamp)
task('app:stamp_version', function () {
    run('cd {{release_path}} && git rev-parse --short HEAD > REVISION
|| echo $(date +%s) > REVISION');
});
after('deploy:symlink', 'app:stamp_version');Code language: PHP (php)
```

This stores a short commit hash (or timestamp) in `REVISION` so you can see exactly what's live. Deployer also keeps previous releases; revert quickly with `dep rollback`.

```
// routes/web.php (admin-only version page)
use Illuminate\\Support\\Facades\\Gate;

Route::middleware(['auth'])->get('/admin/version', function () {
    abort_unless(Gate::allows('viewAdmin'), 403);
    $revision = base_path('REVISION');
    $value = file_exists($revision) ?
trim(file_get_contents($revision)) : 'unknown';
    return view('admin.version', ['revision' => $value]);
});Code language: PHP (php)
```

This route reads the `REVISION` file (written at deploy time) and displays it for admins. It's helpful when correlating bug reports to a specific release.

```
<!-- resources/views/admin/version.blade.php -->
@extends('layouts.app')
@section('content')
<div class="container">
  <h1 class="mb-4">Deployed Version</h1>
  <p><strong>Revision:</strong> {{ $revision }}</p>
  <p class="text-muted">Tip: Keep 5 recent releases to enable fast
rollbacks with Deployer.</p>
</div>
@endsectionCode language: HTML, XML (xml)
```

Keeping version info visible reduces confusion across teams and speeds up incident response. Since releases are kept on disk, you can roll back instantly if a migration or bug slips through.

8 — CI/CD Integration (Optional)

Trigger deployments from your pipeline after tests and builds pass. A common pattern is: test → build assets → create release artifact → `dep deploy`. For a full pipeline walkthrough, see Step-by-Step CI/CD Pipeline Setup for Laravel 12 on GitHub Actions.

```
# .github/workflows/deploy.yml (excerpt)
name: Deploy
on:
  workflow_dispatch:
  push:
    branches: [ \"main\" ]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: shivammathur/setup-php@v2
        with:
          php-version: '8.3'
      - name: Install Deployer
        run: composer global require deployer/deployer
      - name: Deploy to Production
        env:
          SSH_AUTH_SOCK: /tmp/ssh_agent.sock
        run: ~/.composer/vendor/bin/dep deploy production -nCode
language: YAML (yaml)
```

This minimal workflow installs PHP and Deployer, then executes `dep deploy`. Use OIDC/SSH keys securely via repository secrets; add caching for speed as needed.

Wrapping Up

Deployer brings reproducible, zero-downtime releases to your Laravel stack: atomic symlinks, shared storage, safe migrations, warmed caches, and quick rollbacks. Add a small version UI for visibility and trigger the process from CI/CD to move fast without breaking prod.

What's Next

- Step-by-Step CI/CD Pipeline Setup for Laravel 12 on GitHub Actions — run tests, build assets, and deploy automatically.
- How to Deploy a Laravel 12 App on DigitalOcean — server setup that pairs well with Deployer's symlink strategy.
- Deploying Laravel on AWS: Complete Guide (2025) — scale out with load balancers, TLS termination, and RDS.