1v0 Ship v1.0 Faster

Step-by-Step CI/CD Pipeline Setup for Laravel 12 on GitHub Actions

Step-by-Step CI/CD Pipeline Setup for Laravel 12 on GitHub Actions

Continuous Integration and Continuous Deployment (CI/CD) ensures your Laravel 12 projects are always tested, built, and deployed automatically. With **GitHub Actions**, you can build pipelines that handle testing, asset compilation, and server deployment whenever you push to your main branch. In this tutorial, we'll build a full CI/CD pipeline step by step, integrating with services like DigitalOcean, AWS, and even Docker-based flows.

1 — Why Use CI/CD with Laravel?

- **Consistency:** Every push runs tests in the same environment (Docker/Ubuntu runners).
- **Speed:** No more manual deployments—production updates are automated.
- **Quality:** Run PHPUnit, PHPStan, and Laravel Dusk before deploys.
- **Scalability:** Works equally well with EC2, DigitalOcean, or any cloud provider.

If you've already set up DigitalOcean Deployments or AWS Deployments, CI/CD ensures those environments are updated safely and automatically.

2 — Basic GitHub Actions Workflow

Create a workflow file in your Laravel repo at `.github/workflows/ci.yml`. This runs tests and builds assets on every push to main.

```yaml
# .github/workflows/ci.yml
name: CI

on:
  push:
    branches: [ "main" ]
  pull_request:

jobs:
  build-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Set up PHP
        uses: shivammathur/setup-php@v2
        with:
          php-version: '8.3'
          extensions: mbstring, bcmath, pdo_mysql

      - name: Install Composer dependencies
        run: composer install --no-interaction --no-progress --prefer-dist

      - name: Run tests
        run: php artisan test
```
Code language: YAML (yaml)

This workflow installs PHP 8.3 with required extensions, pulls dependencies, and runs Laravel's test suite. It ensures every commit is validated before deploying.

3 — Building Frontend Assets

If your app uses Vite for assets, add Node to the pipeline and build before deployment.

```
- name: Set up Node.js
  uses: actions/setup-node@v4
  with:
    node-version: 20

- name: Install NPM packages
  run: npm ci

- name: Build assets
  run: npm run buildCode language: YAML (yaml)
```

This step ensures your CSS and JS are compiled, ready for production. Combined with `php artisan config:cache` and `route:cache`, it results in fast deployments (see High Traffic Optimization).

4 — Deployment to DigitalOcean (via SSH)

One simple approach is deploying with SSH and Rsync. Store your server's SSH key in GitHub Secrets.

```
- name: Deploy to DigitalOcean
  uses: appleboy/ssh-action@v1.2.0
  with:
    host: ${{ secrets.DO_HOST }}
    username: ${{ secrets.DO_USER }}
    key: ${{ secrets.DO_SSH_KEY }}
    script: |
      cd /var/www
      ./deploy.sh
```
Code language: YAML (yaml)

This uses the `ssh-action` to connect and run your `deploy.sh` (see #51 for a full zero-downtime script). It's straightforward and great for single-server setups.

5 — Deployment to AWS (CodeDeploy)

For AWS, we recommend using CodeDeploy. Combine this with OIDC for GitHub Actions so no static IAM keys are needed.

```
- name: Configure AWS creds
  uses: aws-actions/configure-aws-credentials@v4
  with:
    role-to-assume: arn:aws:iam::123456789012:role/GitHubDeployRole
    aws-region: us-east-1

- name: Upload to S3
  run: aws s3 cp deploy.zip s3://your-bucket/deploy.zip

- name: Trigger CodeDeploy
  run: |
    aws deploy create-deployment \
      --application-name laravel-app \
      --deployment-group-name laravel-asg \
      --s3-location bucket=your-
bucket,key=deploy.zip,bundleType=zip
```
Code language: YAML (yaml)

This triggers a rolling deploy across your Auto Scaling Group without downtime. For the full

AWS walkthrough, see AWS Guide.

6 — Caching for Faster Pipelines

Caching Composer and NPM dependencies drastically speeds up builds. Use GitHub’s cache action.

```
- name: Cache Composer
  uses: actions/cache@v4
  with:
    path: vendor
    key: ${{ runner.os }}-composer-${{ hashFiles('**/composer.lock')
}}

- name: Cache NPM
  uses: actions/cache@v4
  with:
    path: ~/.npm
    key: ${{ runner.os }}-npm-${{ hashFiles('**/package-lock.json')
}}
```
Code language: YAML (yaml)

Each run reuses cached dependencies unless `composer.lock` or `package-lock.json` changes, saving minutes. This is part of larger optimization strategies—see 10 Proven Ways to Optimize Laravel for High Traffic.

7 — Deployment Status UI (Optional)

You can add a simple admin-only UI in Laravel to display the last GitHub Actions deployment status by consuming the GitHub API.

```php
// routes/web.php
use Illuminate\Support\Facades\Http;

Route::middleware(['auth', 'can:viewAdmin'])->get('/deploy-status',
function () {
    $response = Http::withToken(config('services.github.token'))
->get('https://api.github.com/repos/your-org/your-repo/actions/runs?pe
r_page=1');
    $run = $response->json('workflow_runs.0');
    return view('admin.deploy', ['run' => $run]);
});
```
Code language: PHP (php)

This route calls the GitHub Actions API for the latest workflow run and passes it to a Blade view. Store a GitHub PAT in `config/services.php` for authentication.

```xml
<!-- resources/views/admin/deploy.blade.php -->
@extends('layouts.app')
@section('content')
<div class="container">
  <h1 class="mb-4">Latest Deployment</h1>
  <p><strong>Workflow:</strong> {{ $run['name'] }}</p>
  <p><strong>Status:</strong> {{ $run['status'] }} / {{
$run['conclusion'] ?? 'in-progress' }}</p>
  <p><strong>Commit:</strong> {{ $run['head_commit']['message'] }}</p>
  <p><a href="{{ $run['html_url'] }}" class="btn btn-theme"
target="_blank">View in GitHub</a></p>
</div>
@endsection
```
Code language: HTML, XML (xml)

With this UI, your team can see at a glance if the last deployment succeeded—without leaving your app. For debugging failed jobs, you can also integrate with Using Laravel Telescope to Debug Performance Issues.

Wrapping Up

A well-structured GitHub Actions pipeline makes your Laravel 12 workflow smooth: test every commit, build assets, cache dependencies, and deploy automatically to DigitalOcean or AWS. This not only saves time but also ensures reliability and confidence with every release.

What's Next

- How to Deploy a Laravel 12 App on DigitalOcean — pair CI/CD with DO Droplet deployments.
- Deploying Laravel on AWS: Complete Guide (2025) — full AWS production pipelines.
- Laravel Deployment Checklist for 2025 — run this before every release.