



Push deploy a Laravel app for free with GitHub Actions



MAY 22, 2020 / [SAMUEL ŠTANCL](#)

For many teams, it makes sense to use services like [Ploi](#) or [Forge](#). They provision servers for you, configure push deploys, deal with backups, and

Newsletter



Join 31,000+ others and never miss out on new tips, tutorials, and more.

SUBSCRIBE

LARAVEL NEWS PARTNERS

SHIFT



bagisto



BeyondCode

RingCentral
DEVELOPERS

many other things.

You can also use a Platform-as-a-Service like [Heroku](#). PaaS is close to serverless in the sense that you don't think about servers, but the primary difference between PaaS and serverless is that serverless scales automatically, whereas PaaS merely hides the fact that there are servers to deal with. And — speaking of serverless — you can, of course, use a service like [Laravel Vapor](#) and get push deploys.

However, if configuring servers is something you're used to, you might be **interested only in the push to deploy** part of these services. And as such, you might not want to pay for the services above only to use a single feature.

Luckily, **configuring push deploys is super easy to do yourself — and free!** — if you're using GitHub. Specifically, we're going to be using GitHub Actions.

Prerequisites

This article assumes you know how to configure web servers, and as such it will only guide you through configuring Continuous Deployment part — not the actual web servers.

1. A configured web server running PHP

SimpleBackups

Kirschbaum

 TIGHTEN

 DevSquad

 *Honeybadger.io*

 Scout APM

 codecourse



 Akaunting

2. A GitHub repository

The git setup

- Some `public/` assets are in `.gitignore`, but are built on GitHub
- `master` branch is used for development
- `production` is pushed and deployed
- `deploy` is created by the Action, by taking `production` and adding a commit with the built assets

So the code flows like this: `master --> production --> deploy`.

A few notes

We compile front-end assets inside the GitHub Actions — not your computer, nor your server. This means that you **don't have to run npm locally** to deploy, and that the server's **downtime is as short as possible**.

We run tests locally, not in the CI Action. The reason for this is to allow you to deploy hotfixes even if they break some tests. If you want to run tests inside the Action, then simply look up some GitHub Actions phpunit workflow and copy the necessary steps. You can use [this example](#).



Show your support of Laravel News with an official t-shirt!

Laravel Jobs



Senior Software Engineer

Remote

Supporting Cast

1. Server deployment script

Add this bash script to your repository, and name it ``server_deploy.sh``.

This script will be executed on the server to pull and deploy the code from GitHub.

```
#!/bin/sh
set -e

echo "Deploying application ..."

# Enter maintenance mode
(PHP artisan down --message 'The app is being (quickly!) updated')

# Update codebase
git fetch origin deploy
git reset --hard origin/deploy

# Install dependencies based on lock file
composer install --no-interaction --prefer-dist --optimize-autoloader

# Migrate database
php artisan migrate --force

# Note: If you're using queue workers, this is the place to
# ...

# Clear cache
```

Full Stack PHP (Mid-Level/Senior) Programmer

Remote
PhoneBurner

Software Developer

Remote, (US & Canada Only)
Patient Prism

Web Application Developer

Chicago Loop
Leading Real Estate Companies of the World

Multiple Laravel Developers (Part-Time)

Remote, USA Only
The Great Escape Room

Beginner Tailwind

Make good looking designs quickly. All without a single line of custom CSS.

Learn Tailwind CSS and master everything from components to complete designs

```
php artisan optimize

# Reload PHP to update opcache
echo "" | sudo -S service php7.4-fpm reload
# Exit maintenance mode
php artisan up

echo "Application deployed!"
```

The process explained:

1. We're putting the application into maintenance mode and showing a sensible message to the users.
2. We're fetching the ``deploy`` branch and hard resetting the local branch to the fetched version.
3. We're updating composer dependencies based on the lock file. **Make sure your ``composer.lock`` file is in your repository, and not part of your ``.gitignore``.** It makes sure the production environment uses the *exact* same version of packages as your local environment.
4. We're running database migrations.
5. We're updating Laravel & php-fpm caches. If you're not using PHP 7.4, change the version in that command.
6. We're putting the server back up.

Note that the server is always on the ``deploy`` branch. Also note that we're putting the server down for the **shortest duration possible** — only for the ``composer install``, migrations and cache updating. The app **needs** to go down to avoid requests coming in when the codebase and database are not in sync — it would be irresponsible to simply run those commands without putting the server into maintenance mode first.

And a final note, the reason we're wrapping the ``php artisan down`` command in ``(...) || true`` is that deployments sometimes go wrong. And the down command exits with ``1`` if the application is already down, which would make it impossible to deploy fixes after the previous deployment errored halfway through.

2. Local deployment script

This script is used in your local environment when you want to deploy to production. Ideally, if you work in a team, you'll also have a CI Action running ``phpunit`` as a safeguard **for pull requests** targeting the ``production`` branch. For inspiration, see the link to the Action example in the *How it works* section above and make it run on ``pull_request`` only.

Store the local deployment script as ``deploy.sh``:

```
#!/bin/sh
set -e

vendor/bin/phpunit

(git push) || true

git checkout production
git merge master

git push origin production

git checkout master
```

This script is simpler. We're running tests, pushing changes (if we have not pushed yet; **it's assumed we're on `master`**), switching to **`production`**, merging changes from **`master`** and pushing **`production`**. Then we switch back to **`master`**.

3. The GitHub Action

Store this as ``.github/workflows/main.yml``

```
name: CD

on:
  push:
    branches: [ production ]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
        with:
          token: ${ secrets.PUSH_TOKEN }
      - name: Set up Node
        uses: actions/setup-node@v1
        with:
          node-version: '12.x'
      - run: npm install
      - run: npm run production
      - name: Commit built assets
        run: |
          git config --local user.email "action@github.com"
          git config --local user.name "GitHub Action"
          git checkout -B deploy
          git add -f public/
          git commit -m "Build front-end assets"
          git push -f origin deploy
      - name: Deploy to production
        uses: appleboy/ssh-action@master
```



```
with:
  username: YOUR USERNAME GOES HERE
  host: YOUR SERVER'S HOSTNAME GOES HERE
  password: ${ secrets.SSH_PASSWORD }
  script: 'cd /var/www/html && ./server_deploy.sh'
```

Explained:

1. We set up Node.
2. We build the front-end assets.
3. We force-checkout to ``deploy`` and commit the assets. The ``deploy`` branch is temporary and only holds the code deployed on the server. It doesn't have a linear history — the asset compilation is never part of the ``production`` branch history — which is why we always have to **force** checkout that branch.
4. We **force-push** the ``deploy`` branch to ``origin``.
5. We connect to the server via SSH and execute ``server_deploy.sh`` in the webserver root.

Note that you need to store two **secrets** in the repository:

1. a **Personal Access Token** for a GitHub account with write access to the repository
2. the SSH password

If you want to use SSH keys instead of usernames & passwords, see the [documentation for the SSH action](#).

Usage

With all this set up, install your Laravel application into ``/var/www/html`` and checkout the ``deploy`` branch. If it doesn't exist yet, you can do ``git checkout production && git checkout -b deploy`` to create it.

For all subsequent deploys all you need to do is run this command from the ``master`` branch in your local environment:

```
./deploy.sh
```

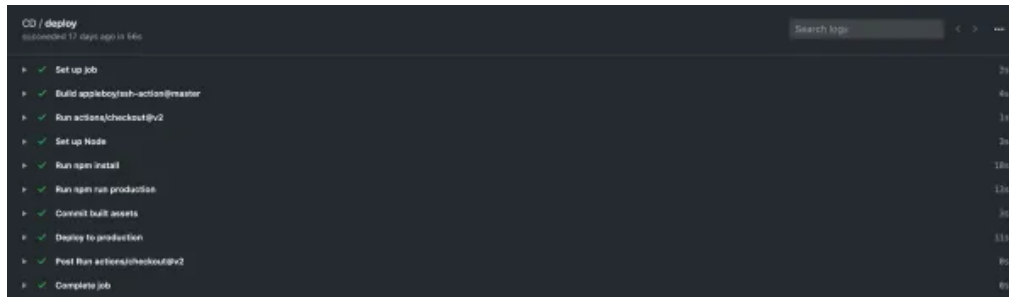
Or, you can merge into ``production``. But know that it will not run tests unless you configure the action for that, as mentioned above.

Performance and robustness

This approach is robust since it makes it impossible for a request to be processed when the codebase and database are out of sync — thanks to ``artisan down``.

And it's also very fast, with the least amount of things happening on the server — only the necessary steps — which results in minimal downtime.

See how this action runs:



The screenshot shows a GitHub Actions workflow run titled 'CD / deploy' that was triggered 17 days ago. The workflow consists of 11 steps, all of which completed successfully. The 'Deploy to production' step is highlighted, showing it took 13 seconds. The steps are as follows:

Step	Duration
Set up job	3s
Build appleboy/ssh-action@master	46s
Run actions/checkout@v2	1s
Set up Node	2s
Run npm install	18s
Run npm run production	12s
Commit built assets	3s
Deploy to production	13s
Post Run actions/checkout@v2	8s
Complete job	8s

The **Deploy to production** step took only 13 seconds, and the period when the application was down is actually shorter than that — part of the 13 seconds is GitHub setting up the ``appleboy/ssh-action`` action template (before actually touching your server). So usually, the application would be down for less than 10 seconds.

Filed in: [News](#)



Slash Your Laravel Application Monitoring Spending with Scout APM! (sponsor)

Introducing PestPHP: Screencast Series by Michael Dyrynda



Scout APM is excited to announce its 80//50//0 promotion to help companies looking to reduce infrastructure spending...

Introducing Pest is a video series by Michael Dyrinda that walks you through setting up and using PestPHP, a new test...



LINKS

NEWSLETTER

ADVERTISE

ARCHIVE

JOBS

YOUR ACCOUNT

CONTACT

NEWS

TUTORIALS

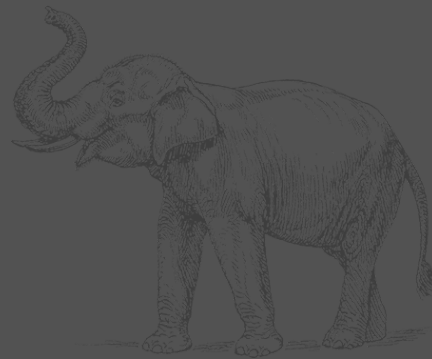
PACKAGES

BOOKS

INTERVIEWS

TRAINING RESOURCES

APPLICATIONS



© 2012 - 2021 LARAVEL NEWS — BY ERIC L. BARNES - A DIVISION OF DOTDEV INC.

DESIGN & FRONT-END CODE BY

zaengle

