PHP Startup Internals

Tom Rochette <tom.rochette@coreteks.org>

November 13, 2015 — 0b83a5a8

- PHP
- MySQL
- Jenkins
- Apache/Nginx
- Linux (Ubuntu)
- node.js/io.js

My goal with this post (and any subsequent posts) is to share my thoughts and current practices on the topic of developing PHP applications in a startup environment.

Starting a new startup means making decisions. Which framework to choose, what tool to use, which programming language, what task should be done before this other task, etc.

Starting is often overwhelming. What should be done first? If we ignore all the questions about the business (what sector? any specific niche? what sort of product?), then the first thing that an individual or a team should aim for is to prepare for iteration.

0.1 The first step

Many would start by working directly on their first project. It makes sense since it is the primary goal of your startup to produce results. However, writing code without establishing some sort of workflow framework will be inefficient.

My first step is generally to setup Jenkins, a continuous integration tool. It allows me to setup automated testing and automated deployment to a development/staging area/environment. This is useful for two purposes:

1. Having an external "party" execute the test in their own environment (separate from mine). This validates that whatever is in source control will work on someone else computer.
2. It deploys automatically "stable" (in the sense that they pass testing) version to an online facing server. With automated deployment, it is possible for me to keep on writing code, have it tested and then deployed to a server where I can ask others to take a look at and provide feedback.

There are a couple of way to get setup.

0.1.1 The easy way

Everything will be setup on the same machine. Here is how it basically goes:

1. Install jenkins
2. Create two jenkins jobs, `project-name-develop` which takes care of building the `develop` branch of your repository and run the tests (basic continuous integration), and `project-name-develop-to-development`, which will again, build the `develop` branch of your repository but this time for the purpose of having it available online.

0.1.2 Jenkins installation

There won't be much to discuss here except a list of plugins that are almost mandatory (either because they make jenkins much more useful or allow you to more quickly diagnose issues).

- AnsiColor
- Checkstyle Plug-in
- Clover PHP plugin
- Credentials Plugin
- Duplicate Code Scanner Plug-in
- GIT client plugin
- GIT plugin
- HTML Publisher plugin
- JDepend Plugin
- JUnit Plugin
- Mailer Plugin
- Matrix Authorization Strategy Plugin
- Matrix Project Plugin
- Node and Label parameter plugin
- Parameterized Trigger plugin
- Plot plugin
- PMD Plug-in
- Self-Organizing Swarm Plug-in Modules
- Slack Notification Plugin
- SSH Credentials Plugin
- SSH Slaves plugin
- Static Analysis Utilities
- Throttle Concurrent Builds Plug-in
- Timestamper
- Violations plugin
- xUnit plugin

0.1.3 Jenkins jobs

I'll now go into more details as to what each does.

0.1.3.1 `project-name-develop`

1. Pull the latest revision from the repository
2. Download and update composer (if required)
3. Install dependencies
    1. bower install
    2. npm install
    3. composer install
4. Build assets to validate they compile
    1. Compile LESS into CSS
    2. Concatenate and minify JS
5. Prepare the application environment
    1. Migrate database
    2. Seed database
6. Run continuous integration tools to assert code quality
    1. phpunit
    2. phploc
    3. pdepend
    4. phpmd
    5. phpcs

6. phpcpd

An iterative cycle here should take less than 5 minutes (and a maximum of 30 minutes). The goal is to quickly know after pushing changes to your repository that nothing is broken.

0.1.3.2 `project-name-develop-to-development`

For this to work, you simply need to make a symbolic link from the jenkins project workspace to some path which apache/nginx makes available to external users. For example

```
/home/jenkins/workspace/project-a-develop-to-development/public -> /var/www/development/project-a
```

1. Pull the latest revision from the repository
2. Download and update composer (if required)
3. Install dependencies
    1. bower install
    2. npm install
    3. composer install
4. Build/Prepare website
    1. Compile LESS into CSS
    2. Concatenate and minify JS
5. Prepare the application environment
    1. Migrate database
    2. Seed database

An iterative cycle here should take less than 5 minutes. Anything that takes longer than that would be suspicious.

0.1.3.3 Overview

Now that you have both projects setup, here's how things work. First, `project-name-develop` is triggered every 1-5 minutes and checks the repository for changes. If changes are detected, a build starts and will verify that the current state of the code is valid.

Once the build finishes, if it is successful, `projecy-name-develop-to-development` will start (triggered on `project-name-develop` success). It will deploy the stable code so that users may test it.

A whole change cycle will generally take from 1 to 30 minutes depending on how many tests you have and how well you've been able to optimize your jenkins build workflow.

0.1.4 Speeding up your jenkins workflow

Here's a list of things to try/check:

- If you are running `phpunit` with code coverage, disable it and run it in a separate jenkins project. Code coverage is 2-5x slower than without it. When you are running the tests, you want to know the results fast and code coverage should not be a priority. Speed is the priority.
- If you are running tests against a database and the tests requires setting up and tearing down the database (either just truncating the tables or full DROP tables), search for ways to avoid hitting the database or how to improve performance. For example, if you are testing using SQLite, run an initial database migration and seeding and copy the resulting .sqlite file so that it can be copied on test setup instead of migrating/seeding every time.
- If migrating/seeding takes a long time, keep the resulting .sqlite file and only rebuild it if its source files (dependencies) have changed. On a project, you will run tests much more often than you will be rebuilding the .sqlite file, so it is worth investing in developing such a tool.
- Since `php` is single threaded, look for tools that will enable you to do multi-process php testing. An example of such tool is liuggio/fastest. Depending on the number of processors/cores you have available, you could see a 4-8x gain in speed.

- If you have the money/hardware, distribute testing over many machines. If you want a unified phpunit code coverage/results, you can use `phpcov` to merge separate test results into a single result file.