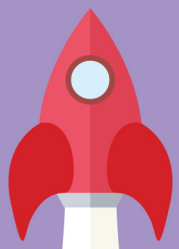
DEPLOYING PHP APPLICATIONS



BY NIKLAS MODESS

Deploying PHP Applications

Niklas Modess

This book is for sale at http://leanpub.com/deploying-php-applications

This version was published on 2022-06-15

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Niklas Modess by spreading the word about this book on Twitter!

The suggested hashtag for this book is #deployphpapps.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#deployphpapps

Contents

Introduction

Background

The PHP language is in an exciting phase right now. After lagging behind on the more traditional software development practices for too long. Techniques that have been more or less considered as given in other communities. Continuous integration, package management, dependency injection, and adopting object-oriented programming to its full extent. But as of PHP 5.3 (released on 30 June 2009), there are no more excuses as to why you can't write modern and clean code. The community has risen up to the challenge. Many people have stepped forward, teaching and building tools for accomplishing these practices.

Yet we seem to forget deployment in this. It's time to bring it to the table for discussion. Great tools and services are out there. But few resources are available on setting up, maintaining, and optimizing your deployment process. I hope to shed some light on what is essential and how you achieve it in this book.

Deploying is not about pushing changes to a production server but is an integral part of the software development process[1]. You need to put thought into each step. Applications are often unique little snowflakes, and you need to tend to their particular needs. When working in a team, you need an excellent deployment process. The team should work in specific ways to enable the process to benefit all parties involved.

Every software application has a life cycle that the deployment process supports and is necessary for. You want to be able to maintain the application, roll out new features and do bug fixes without constant pain and headaches while doing it. You should be able to manage your branches, push your code, run the appropriate tests, migrate your database, and deploy the changes quickly and confidently. It will be a breeze with a good deployment process and a workflow that enables this. If you can do a fast and easy rollback, your confidence in pushing code will benefit greatly.

In a more agile software development world, being able to deploy is essential. Release cycles are getting shorter and shorter, and some organizations even push it to the limit with continuous delivery[2]. In an environment with short release cycles, the importance of the deployment process intensifies. Not being able to deploy or roll back fast enough could slow down your entire development process.

[1] http://en.wikipedia.org/wiki/Software_development_process
[2] http://en.wikipedia.org/wiki/Continuous_delivery

Who it's for

You are already familiar with PHP, and you're not afraid of the command line. But you could also be a manager of a software development team that is a part of deployment regularly.

Legacy is not solely a code issue but a process issue as well. Are you looking to streamline your deployment process, or do you want to scrap your current one and start fresh? Then this book is for you.

Outside its scope

I consider server provisioning an almost crucial part of deploying your application, but it's too big of a topic for this book. It could, without a doubt, be a book on its own. I also don't want this book to focus on any framework or tool but keep the content and name broad instead of something like *Deploying PHP applications to Amazon Web Services with Ansible*.

The tools and commands used will be outside the scope unless it's a deployment tool (then it will have a dedicated section). I will make examples with Git, Composer, Grunt, PHPUnit, and other tools. If you want to learn more about the tools, many books, screencasts, and blog posts are to find. Google is your friend.

Assumptions

I know it's not nice to make assumptions about people or software. But nevertheless, I'm still going to do it to some extent in this book. I will make them when I approach examples, but I'll not judge you, your team, or your application.

Where you deploy to

You will need a hosting environment that you are somewhat in control of. If you cannot install software or run commands, it will limit what you can achieve. Whether it's a hosted server, co-location server, or a VPS does not matter. However, to use everything in this book, you need control over it for installing software, changing configuration, etc.

Git

The base of some topics discussed will use Git as version control. Why? Because I think it enables workflows that are best suitable for a good deployment process. There's a chapter on Git for version control and branching strategy for a good deployment process. I could've named it *Git version control*. But I'll leave the name without Git in it since there are perhaps many things you can apply to other version control systems. Other than Git, I have worked with Subversion and Perforce, but when I found Git and started incorporating it into my workflow, I never looked back.

Both ends

There will also be an assumption about your application that it's not a pure backend application. If your application is a REST API, for example, with no frontend, it will not matter, though. I will give general examples of managing builds for your frontend, but all the commands used will be arbitrary.

About the author

I have been developing PHP applications for over 15 years now. During this time, I've created and deployed various applications. The scale of these applications has been from a few hundred users to over 250 million users.

Oh, by the way, I'm from Stockholm, Sweden. That means I'm writing a book in my second language, and I would appreciate all the help I can get regarding spelling and grammar. If you find anything, please create an issue in this repository[3] or fork it, fix it and send me a pull request. Thank you!

Code samples

In the repository on github[4], you can find code samples structured by chapter. Any substantial amount of code used in the book is available for reference and use.

Thanks to

My friend and talented designer extraordinaire **Joakim Unge**, for the awesome cover image. Look at it. It's a fucking rocket ship! Nowadays, he's a developer, and you can find him at https://joakimunge.se/[5].

My sister **Jenny Modess**, the talented copywriter, for proofreading from a non-technical perspective. Keeping my spelling, grammar, and storytelling in check!

[3] https://github.com/modess/deploying-php-applications
[4] https://github.com/modess/deploying-php-applications
[5] https://joakimunge.se/

1. Automate, automate, automate

I want to get this off the bat right away. The most crucial ingredient in your deployment process should be to **automate everything** to **minimize human errors**. If your deployment process involves manual steps, you will have a bad time, and shit will hit the inevitable fan. Nobody is perfect. People can, and will, forget if they need to remember.

An automated deployment process will boost the confidence of the person deploying. Knowing that it will take care of everything is a significant contributor to feeling safe during a deployment. Of course, other unexpected issues can arise, but with a flexible process with logging and notifications, you can relax and figure out what went wrong.

1.1 One button to rule them all

You should have a **one** button to push or **one** command to run to deploy your application. If you don't, something is wrong, and you should automate all steps. Perhaps the single manual intervention I would consider okay is *"Are you sure? This will push stuff to production. [N/y]"*. This might be a bold statement, but I truly believe in it.

Even if you're the sole developer on a project and deploying the application each time, I still say it's terrible to have manual steps. When it comes to teams, it becomes a lot worse. If not all team members are familiar with the deployment steps, they won't be able to deploy. Team members come and go. Whenever a new team member arrives, they will need to learn how to deploy correctly. Sure there can be documentation for it. But whenever someone is familiar enough with it, they will start to deploy without it.

1.2 Example of a manual step

Let's take an example of a revision number for your assets in code. First, I would say something along these lines is relatively common practice. I sure have seen something like this way too often. This revision number handles cache busting, so browsers do not keep serving old assets after deploying.

This is a constant in a class somewhere for managing static assets versions.

```
class Assets
{
    const REVISION = 14;

    // [...]
}
```

Then it's applied to serving the static assets.

```
<link rel="stylesheet" type="text/css" href="style.css?v=<?=Assets::REVISION?>">
```

This is truly as bad of a manual step as you could have. It's easy to forget since it requires a code change. It might break the users' experience serving cached and outdated assets if it's forgotten. A manual step where you have to run a command connected to the deployment would be better since it would be easier to remember. When you already have the command line in front of you, you will be more prone to remember it.

Since it requires a code change, another issue will occur. That is when you remember the step in the middle of the deployment before pushing changes to production. You stop before pressing the button, screaming, "Shit, the assets revision number!". You have to deal with changing the code, committing it, and pushing the code through a new deployment. In a perfect world, you would've already merged a release branch and tagged it (discussed in chapter 2), so how would you approach that now? Reset your repository, remove the tag, commit and create the release branch again? Or would you commit and push, knowing that it will break traceability[6] for this specific deployment? This is a problem that goes away with automation.

1.3 Time is always a factor

You might say that you don't have time to automate all the trivial steps or commands. For example, if you spend one hour automating a command that takes one second to run, you would have to run that command 3601 times before you have saved time on it. Yes, I did the math. While this is true, I would say that it isn't the whole truth.

We need to consider the time spent on dealing with issues arising when forgetting it. Time spent on cleaning up. Can you measure terrible user experience when your application breaks or behaves incorrectly? You can't. In the previous example, you can't tell how much time you will spend correcting that error. Neither can you tell the impact on the users? If the problem isn't caught in time, it could persist for a long time.

[6] http://en.wikipedia.org/wiki/Traceability#Software_development