www.phparch.com

January 2019
Volume 18 - Issue 1

php[architect]

DevOps Depths

Diving in the OPcache

Making Executable Images

Jenkins Automation

There Are No Snow Days When You Work Remote

ALSO INSIDE

Internal Apparatus:
How PHP Works: Show Me the Code

Education Station:
DevOps and You

The Workshop:
The Road to 7.3, Part One

Free Sample Article

Security Corner:
Strong Security Stance in the New Year

finally{}:
Resolutions of Collaboration

PHP[TEK] 2019
Conference

Save the Date!

The only conference that provides
deep-dive, sequential sessions for senior developers and
entry-level, need-to-know topics for beginners.

A conference experience tailored to meet your learning needs.

tek.phparch.com

CONFERENCE

Loudermilk Conference Center
May 21-23, 2019
Atlanta, GA

DevOps and You

Chris Tankersley

The tech industry is always awash with new ideas that are actually old. One which gained traction in the last ten years is the idea of "DevOps." This term is the combination of "Development" and "Operations" and is meant to show these two roles can be combined for more efficiency.

In the past (and, still, in many places now), most organizations split their teams into two roles: one team made up of the developers and another one for operations roles. Developers would work on an application, be it an internal application or a website, and then hand it off to operations. Operations handled all of the hardware and installed applications. When you needed more RAM, they would provision it. Need updates installed? That was all them.

This division was in part due to how many organizations were structured but also fed into existing stereotypes. Developers did not care about performance or security. Operations didn't know how to build tools. They were two teams ever at odds with each other, but dependent on the other's tool sets.

Rise of the Webmaster

The funny thing is, "DevOps" is the same thing many older web developers have already done. When I started creating websites, I not only had to learn HTML but also how to get them out on the internet. Sure, I could have run my server at home from the confines of my dial-up connection, but then no one could experience my awesome Final Fantasy VII Fan Site.

Early web developers had to learn how to use things like FTP to get their files online. Once you learned how to get your files on the server, you had to keep them up-to-date. We joke all the time about editing in production, but for many people "being online" was not a constant. We edited our local copy and then pushed it to the server. For me, that meant learning and scripting FTP commands.

PHP filled this niche in very well. It was straightforward to set up and did not require near the amount of extra work other languages like ASP or Perl needed to execute. You could get away with just FTPing files up, and they were immediately available.

It was not uncommon for web developers to also have to learn how email servers worked to send and receive messages. Why? Because computers are all alien devices and, of course, the web developer knows how to make the server work. So you learned how email servers worked, which meant learning Linux or Windows even in depth.

Then you had to secure the server after your first hack. You did all of this while refining your deployment processes because now you had less time in the day as you split between development, server work, and routine things. Burnout loomed on the horizon.

For many people, getting a job that removed all this work was a good thing. It meant you could focus on what you wanted, either the operations side or the development side while ignoring the other stuff. That was another team's problem.

Lean Manufacturing

In many "enterprise" environments, things were more segregated. I put enterprise in quotes because being "enterprise" is a state of mind, not anything directly related to a company's size. There are many advantages to organizing teams into specific disciplines, but it can create a lot of red tape.

For example, at a previous job when we needed to get vendor software installed, we had to deal with the operations team. They would take our specs for the software and purchase a server. That server would then get delivered, and installed by them. They would provision the OS and give us just what access we needed (at the time, jailed FTP access). That was all you got. You rarely got full shell access, so tools like Composer or Git were impossible to run yourself.

Many companies were already subscribing to the "Lean Manufacturing" tenets companies like Toyota had helped pioneer and champion. Ironically, according to devops.com[1], Toyota was inspired by the assembly lines Henry Ford pioneered, reinforcing that what is old is new again. Toyota saw many ways they could increase efficiency and reduce costs by making changes to their processes.

Lean Manufacturing led to "Continuous Improvement," which meant evaluating ways to reduce waste (or "muda") by:

- Keep inventory at a minimum
- Minimizing the order queue
- Maximizing the efficiency of the manufacturing process

Waste means many things, not just physical waste of space and parts. Taiichi Ohno, the "father" of the Toyota Production System, saw seven forms of waste:

1. Transport: Moving things around more than required
2. Inventory: All of the components in all stages of work

1 devops.com: https://phpa.me/origin-devops

3. Motion: People/equipment moving more than required
4. Waiting: Dead time between steps
5. Overproduction: Making more than is needed
6. Over Processing: Doing more to a product than required
7. Defects: Time spent fixing mistakes

Do some of these things sound familiar? These forms of waste are found in most teams' development processes. Developers and system operators saw many of these tenets that were being used and wanted to use them to improve their processes.

In 2008, Andrew Schafer hosted a "Birds of a Feather" meeting to talk about something called "Agile Infrastructure." Agile had been around in software development in various forms since 1957 at IBM[2], but in the land of operations, it was a bit different.

One person showed up, a Belgian project manager named Patrick Debois. Debois was frustrated by the roadblocks he encountered while working on a government project to migrate data centers. Debois and Shafer formed the "Agile Systems Administrator" group.

Nothing much of import happened until they saw a 2009 presentation by Flickr employees John Allspaw and Paul Hammond. Allspaw and Hammond talked about how Flickr ran multiple deployments throughout the day. This presentation, "10+ Deploys Per Day: Dev and Ops Cooperation at Flickr" is regarded as a seminal moment in the creation of DevOps. Debois ended up creating the DevOpsDays conference after this.

What is DevOps?

Unfortunately, there is no formal definition for DevOps. The closest thing we have is from Len Bass, Ingo Weber, and Liming Zhu from the Software Engineering Institute, which is that DevOps is "a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality."

The definition is vague, but my hot take on this is that nearly ten years later DevOps has turned into a catchall term for what we used to call a Webmaster, but with a bunch of cool new toys to play with. I feel we are getting away from the original ideas that worked in Lean Manufacturing and that we learned in Agile.

To me, DevOps is not a job title. You should not be hiring someone that is "DevOps certified" or is a "DevOps engineer." DevOps is a process everyone should follow to empower the team to deploy and deliver software quickly and efficiently without the cruft that came with the tools from much of the nineties and early 2000s.

Removing Waste

If you want to embrace DevOps truly, you first need to look at your waste. In the following articles, I will go into more detail, but DevOps' goal is removing barriers. We want to make it as easy as possible for developers to push code out, and to define the systems they need.

Transport

How hard is to move code back and forth? In an ideal setup, a developer can check out their code from a repository, do their work, and check it back in. From there it should be a highly automated, simple process to move the code into production. You do not need fancy tools for this.

Once a developer is done with a branch, what are the steps to get that code onto a staging server? Who or what runs the unit tests for verification? Is it hard? How many steps are there? Start by looking at the process to remove what is unnecessary and automate where possible.

For example, you can set up a Jenkins install that pulls down code from each branch as it is checked in, and runs all of your unit tests. If it's green, you can look into having it automatically pushed to a staging server. You do not need anything complicated like Kubernetes or Docker. Instead, sprinkle some automation into your existing workflow.

Inventory

Slim down your code inventory. Use something like tombstones[3] to look for and remove dead code. Check your code coverage for code which doesn't execute during tests. Is it just a lack of coverage or is it because it no longer gets called?

Do you have code that is commented out? Delete it! There is a reason we have source control systems; use them.

How much of your code can you move to third-party packages? It may be worth it to start to move to those packages and get rid of the time it takes to maintain things. On the flip side, do you componentize the internal code of your application? Why? It may be easier to have a monolithic application than a small main app that pulls in a bunch of internal dependencies.

When it comes to your repository, keep it lean as well. Delete old branches which are no longer used. If you are using something like GitFlow, should you be? Do not make code management any more complicated than needed.

Motion

How many places must you check to get information? If your code is in Github, but your continuous integration is in Jenkins, and your code deployment is controlled by Kubernetes, you may need to go to three different places to find out the state of your application. Find ways to bring all this information together into one place.

If you find yourself having tens of tabs open during the day just to check things, take a step back. Figure out what information is essential. Can you consolidate some of these things? What about removing tools altogether?

Waiting

PHP developers do not get to use the "I'm compiling!" excuse for their downtime, but we sure get to use "it's building

2 *since 1957 at IBM:* https://phpa.me/wikip-agile-software

3 *tombstones:* https://phpa.me/-tombstone-programming

DAYCAMP 4 DEVELOPERS

BEYOND PERFORMANCE

JANUARY 18, 2019

Developers, teams, and companies have to be on the lookout for ways to make their applications perform faster; they have to look "Beyond Performance".

That is the theme of this next Day Camp 4 Developers and that is what our speakers will help you do, look beyond performance. We are countering everything from what makes PHP tick, to multi-threaded, non-blocking programming in PHP.

High Performance Web Services With Php
Demin Tin

How PHP Ticks
Sara Golemon

Asynchronous Expressive
Matthew Weier O'Phinney

Evented Architectures
Jesse Decker

Php + Redis + Nginx = Ludicrous Speed
Jason McCreary

Register Today

https://phpa.me/daycamp-4-devs

and deploying!" in many modern workflows. Is anything blocking a developer from working while other processes run? Why?

I am by no means advocating working ourselves to the bone—nor ignoring breaks, but are developers stuck frequently? Can your developers not check something until the code is deployed to a test server? At my day job, a deploy takes upwards of twenty minutes, but during that time I can work on other things.

Waiting may be something as simple as waiting for your continuous integration server to give you the green light. Can this be sped up? Are developers blocking because they must wait for the CI to run? Alternatively, is this the only way for them to know their tests pass?

Overproduction

Are you building just the parts you need to? This is a general Agile Development practice. I'm not talking about actual features on the board. When building something out in the code are you adding interfaces for the sake of having interfaces, or do you need them? Should a process need as many objects, factories, subclasses, or traits as you are coding?

Agile and Iterative development tells us we should not be building any more than we need to. There are reasons we have ideals like KISS[4]

and SOLID[5]. Do not over-engineer your project any more than it needs.

On the systems side, do not over-engineer your deployment process. Often, I see people struggling to implement Docker when they shouldn't even be using Docker in their process. Why are they using Docker? Someone told them they needed it without considering if it makes sense. Do you need Kubernetes when a simple Docker stack would work? If not, do not use it!

Look at the all the tools you use, and see if they make your life easier or harder.

Over Processing

Are you building features or changes you *think* are needed, or you *know* are needed? What does your backlog look like for the next sprint? As a developer, you might have little say over the features included in a release, but make sure whatever you are working on is just what is needed.

Building or planning features which never come to fruition wastes time. You may think you need to support multiple API backends for some data, but it turns out you might never need to. Don't build more support into an application than needed.

Defects

Defects manifest in different ways in the real world. It can be a failed part breaking during your normal day-to-day process, a stick of RAM dying in the middle of your game, or a chair leg coming off due to a loose bolt. Whatever it is, the defect takes time out of your day to deal with it.

The same thing happens in development. When you discover a bug, it should be taken care of right away (or as quickly as possible). Doing so diverts a developer's attention and takes valuable time away from working on new features.

Unit tests, feature tests, system tests, and others exist to help. If you start to use practices like this, you can reduce the waste that comes from dealing with bugs once released into the wild.

DevOps—Lean Programming and Lean Architecture

The original ideals of DevOps were to reduce the friction between developers and system operators. We so often get wrapped up in processes and rules that we forget development, especially for web developers and the growing number of software companies, should be about making a product as efficiently, quickly, and with the highest quality we can.

DevOps is not some new or revolutionary idea we just figured out ten years ago; it's just a new term for the same thing we always wanted. It's an easy way to get our code into production without being hindered by someone else.

The first thing we need to do is reduce waste. With less waste, we can have simpler systems. With simpler systems, things break less often, and we can go home on time.

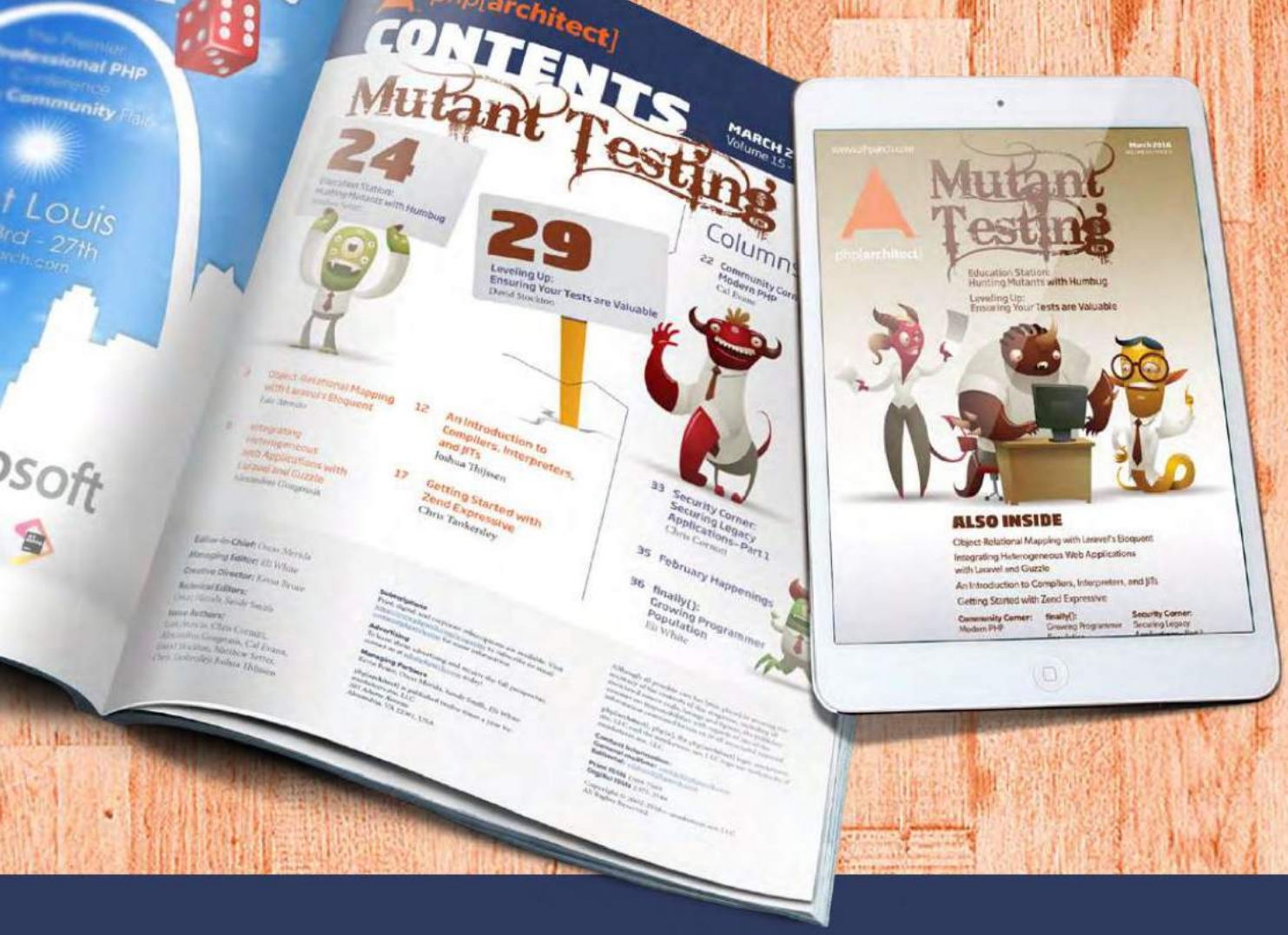
Chris Tankersley is a husband, father, author, speaker, podcast host, and PHP developer. He works for InQuest, a network security company out of Washington, DC, but lives in Northwest Ohio. Chris has worked with many different frameworks and languages throughout his twelve years of programming but spends most of his day working in PHP and Python. He is the author of Docker for Developers and works with companies and developers for integrating containers into their workflows. @dragonmantank

Related Reading

- *Love/Hate—The Dysfunctional Relationship We Have With Tools* by Shahina Patel. February 2018. https://www.phparch.com/magazine/2018-2/february/
- *Education Station: Rock Your Deployments With Rocketeer* by Matthew Setter. April 2017. https://www.phparch.com/magazine/2017-2/april/
- *Deploying with Ansible* by Ramon de la Fuente. August 2016. https://www.phparch.com/magazine/2016-2/august/

4 *KISS: https://phpa.me/wikip-kiss-principle*

5 *SOLID: https://phpa.me/wikip-solid*



Borrowed this magazine?

Get php[**architect**] delivered to your doorstep or digitally every month!

Each issue of php[**architect**] magazine focuses on an important topic that PHP developers face every day.

We cover topics such as frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.

Digital and Print+Digital Subscriptions Starting at $49/Year

http://phpa.me/mag_subscribe

php[**architect**]