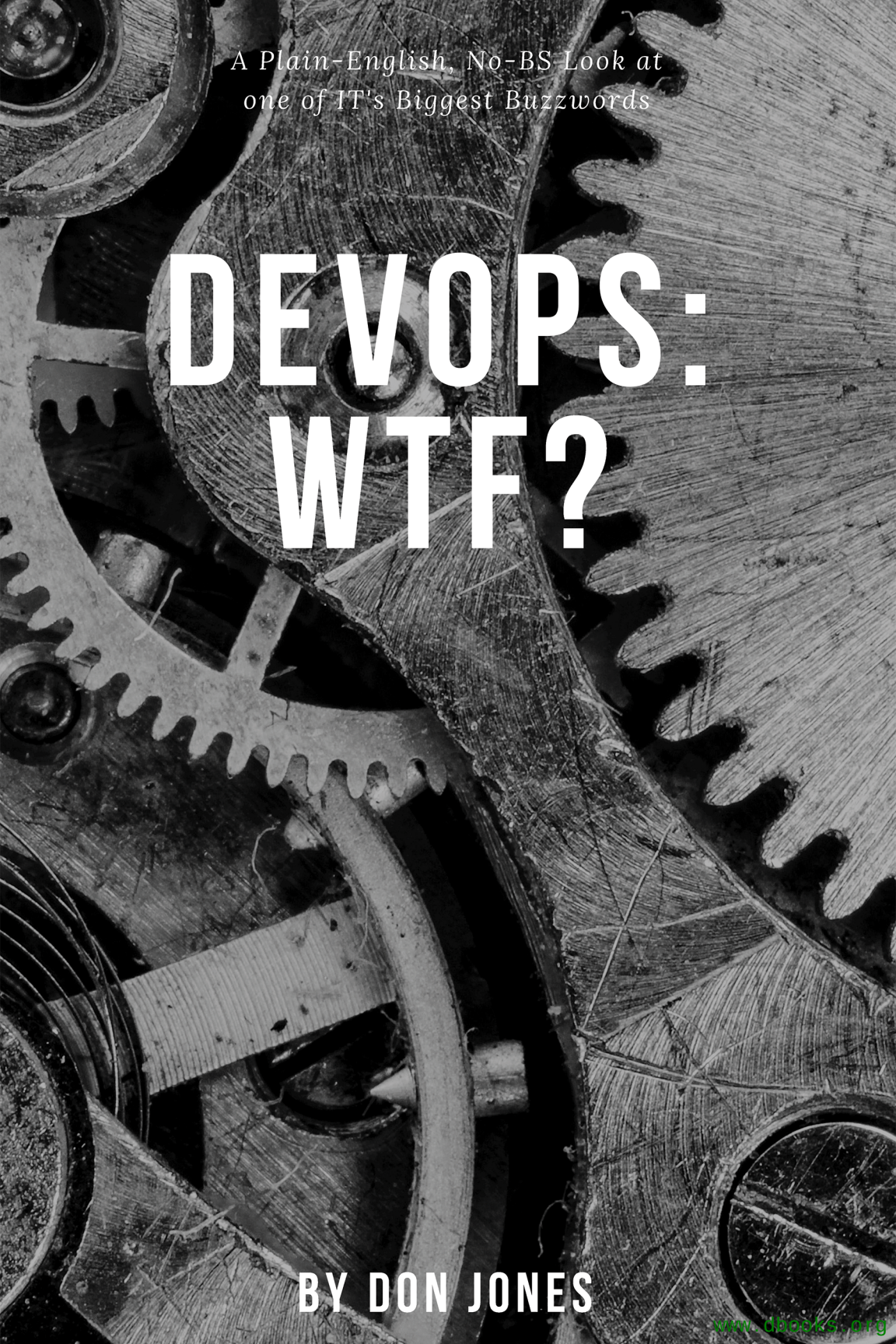

DevOps: WTF?

The DevOps Collective, Inc.

This book is for sale at http://leanpub.com/devopswtf

This version was published on 2018-07-11

Also By The DevOps Collective, Inc.

Creating HTML Reports in Windows PowerShell

A Unix Person's Guide to PowerShell

The Big Book of PowerShell Error Handling

DevOps: The Ops Perspective

Ditch Excel: Making Historical and Trend Reports in PowerShell

Secrets of PowerShell Remoting

The Big Book of PowerShell Gotchas

The Monad Manifesto, Annotated

Why PowerShell?

Windows PowerShell Networking Guide

The PowerShell + DevOps Global Summit Manual for Summiteers

Why PowerShell? (Spanish)

Secrets of PowerShell Remoting (Spanish)

DevOps: The Ops Perspective (Spanish)

The Monad Manifesto: Annotated (Spanish)

Creating HTML Reports in PowerShell (Spanish)

The Big Book of PowerShell Gotchas (Spanish)

The Big Book of PowerShell Error Handling (Spanish)

Contents

An Agile Notice

This book is written and published on Leanpub.com, an "Agile-publishing" platform. That means you may be seeing an in-progress book; you may run across typos, incomplete sections, or "stubs" where I've included some notes about my future content. It's all part of the authoring process, and I'm glad you've come along for the ride.

Because this book isn't destined for a "traditional" publishing platform, it'll remain here on Leanpub. That means you can *always* submit suggestions, thoughts, or corrections via the book's "Email the Author" link on Leanpub. And, if you've purchased the book (even if you've paid $0), you can always come back and download the latest and greatest (if you allow them, Leanpub will notify you via email of major new builds).

Introduction

I wrote this book because, it seems, "DevOps" is creating a lot of anxiety amongst the IT professionals of the world. It's also causing a bit of a boom for technology marketing people, who in turn are making the whole concept confusing for businesses and professionals alike. I've answered so many questions about DevOps at conferences, in online forums, and in blog comments that I found myself retyping almost the same things over and overâ€¦ which made me feel that a small book might be in order.

This is an effort to set the record straight for someone who might be feeling a little vague or uncertain about what DevOps is, whether or not they're "doing DevOps," and so on. It's meant to be useful to business leaders, technology leaders, and technologists. I'll keep it concise and straightforward. Because The DevOps Collective is a nonprofit, and because we don't sell anything, I can also keep this completely noncommercial, vendor-neutral, and honest.

I hope this helps!

What is DevOps?

DevOps—a portmanteau of "Development" and "Operations"—is an IT management philosophy. That's right: it's a *philosophy*. It's not a tool, although you can use tools to implement the philosophy. It's not a job role, although you'll need people to make it happen. Like many philosophies, DevOps isn't for everyone, any more than vegetarianism or Buddhism are for everyone.

DevOps was created as a *reaction* to something that was happening in some IT shops. As some organizations started adopting Agile software development approaches, they found themselves pumping out lots of new software builds in a fairly short period of time. Where an old-school software development team might produce a major software build once every year or three, an Agile dev team might pop out new builds every day, or every week. You see this a lot on your mobile phone, probably, with new app updates coming down pretty much every day.

The problem is that a lot of IT organizations don't trust always their developers. Everyone's been burned when, after toiling away in their code editors for two years, the dev team finally announced that a new software version was ready—only to have the Operations team deploy it and immediately discover dozens of critical, job-destroying bugs. "Change is bad!" became the mantra, and Operations—often through management frameworks like ITIL—became the gatekeeper for deploying software safely, reliable, and most importantly in a way that could be rolled back if everything went askew. As a result, all the Agile development techniques in the world didn't matter, because Operations wasn't prepared to deploy new software versions every fifteen minutes.

DevOps was specifically created not so much to take Ops out of the loop, but to make Ops more automated. DevOps was designed to

trust the developers, and to let them more or less autonomously roll out new builds whenever they wanted to. Developers check in code, a miracle occurs, and that code is in production. The "miracle" is DevOps, plus all the tooling and processes that support the DevOps philosophy.

DevOps is, then, a philosophy specifically suited for solving a particular type of problem: rapidly deploying code whenever the developers say it's ready. If that isn't a problem you have, or if that outcome isn't what you want, then DevOps isn't for you. That's important to understand, because DevOps *isn't* for every organization, and even within organizations that use it, DevOps isn't for *every* software product that they create. DevOps works to create certain, specific outcomes, and it's only useful when you want those outcomes. DevOps also creates its own sets of risks and compromises, and if you don't like those, then DevOps isn't for you. Despite what some marketing folks and pundits might have you believe, DevOps isn't some shiny new magic bullet for IT; there are *plenty* of products and organizations who will not like it, use it, or appreciate it.

DevOps: By the Numbers

Puppet Labs, a vendor in the DevOps tooling space, produces a "State of DevOps Report" each year. I'm quoting from their 2016 and 2017 reports, which can be found here[1].

Remember: DevOps is all about enabling the rapid deployment of software. When that software breaks, DevOps is meant to equally support a rapid deployment of a replacement version, or of the last-known-good version. So these numbers need to be taken with those goals in mind.

Organizations that employ a DevOps approach tend to deploy software versions up to **two hundred times** more frequently. They deploy up to **2,500 times faster**, and their mean time to recover from a failure is up to **96 times faster**, although they're less than **one-fifth** as likely to actually have a failure in production. High-performing DevOps shops are at the tip of the iceberg, deploying multiple software versions *per day*, often with less than an hour's notice. They recover from failures in under an hour, with no more than 15% of their changes resulting in a failure.

These numbers are a big deal.

But these numbers *aren't just DevOps*. In fact, arguably, these numbers aren't DevOps at all; they're the result of a high-performing, Agile development team *that has been enabled by DevOps*. In other words, it's the development team that makes the new deployments, deploys them, fails or doesn't fail, recovers from failures, and so on; DevOps is only there to *enable* those developers to make it all happen.

You might ask, "how can DevOps possibly get change failures under 15%?" It can't. But an Agile development team producing daily

[1] https://puppet.com/resources/whitepaper/state-of-devops-report

builds tends to make *small* daily builds, with just tiny changes. Tiny changes are less likely to fail. DevOps *enables* them to do those daily deployments. It's when developers start stacking up weeks… months… and years of changes into a single deployment that you're more likely to have a profound failure, simply because you've got more "moving parts" in each change release. When a change does fail, it's easier to track it down if you only deployed one change; if you deployed hundreds of changes, tracking down the problematic one is harder, and takes longer to recover from.

So DevOps *enables* these numbers by encouraging Agile, small-iteration development and deployment. The benefit is from Agile, but you can't really "do" agile without having DevOps there to a actually deploy the agile code.

What Isn't DevOps?

DevOps isn't a software development methodology; it *supports* Agile methodologies. DevOp isn't a software deployment methodology, either. Although DevOps concerns itself with software deployment, that concern is basically, "make it fast and seamless;" you can use any *methodology* that meets that criteria.

DevOps isn't a tool you can buy, although tools—often many of them in a "technology stack"—can help bring DevOps to life.

DevOps isn't a job. It's a philosophy. However, you will have to have people whose jobs involve making the DevOps tooling and processes work on a daily basis.

DevOps isn't automation. DevOps *includes* automation, often a whole lot of it, but automation alone doesn't "do DevOps." If you work in a company that uses shell scripts to deploy new servers, that isn't DevOps, although that activity might be one that a DevOps shop also used. DevOps is all about *deploying software seamlessly, rapidly, and on-demand*. DevOps doesn't seek to address the many other aspects of modern IT management.

DevOps isn't The Cloud. You can "do" DevOps internally on-prem and internally, although many cloud-deployed products can benefit from a DevOps approach. Many cloud providers offer specific, DevOps-friendly services designed for products that move at an Agile cadence.

DevOps isn't a management framework like ITIL, although it can exist within such a framework, or exist without any such framework. DevOps doesn't tell you how to "run IT" in your company; it's an *approach* to solving a specific problem, which is, "how can we safely and consistently deploy software a lot faster and with a lot less effort?"

What Does DevOps Look Like?

I tend to evaluate DevOps best as a kind of organization chart. There are certain org charts that, to me, scream "we're doing DevOps!" and certain ones likewise scream the opposite.

DevOps is centered around code, and its deployment, which means DevOps is centered around a *product*. A product might be an app, or it might be some kind of infrastructure-creation system ("Infrastructure as Code," which I'll discuss later), or something like that. With that in mind, the team of people involved are usually called the *product team*.

A product team often consists of several roles. You might have a project manager, a product manager, several developers (although not more than can eat only two pizzas[2], a UX designer, and so on. You'll also have some Operations people, such as a database administrator, someone in charge of the product's deployment, and so on. Everyone needed to take the project from "code" to "in production" is on the team. You **do not** simply have a dev team that codes, and then tosses the code "over the wall" to someone else. DevOps eschews silos and focuses on bringing everyone together who contributes to the product's existence and success.

Very notable is the inclusion of Operations in the product team. "Hey, let's use this new user interface library that looks cool!" might be all the discussion a pure-dev team has before jumping in and using said new library; the presence of an Operations person *on the team* lets them raise concerns like, "guys, that thing is a bear to deploy and requires a full system restart every time we do." This

[2] https://www.theguardian.com/technology/2018/apr/24/the-two-pizza-rule-and-the-secret-of-amazons-success

is the heart of DevOps: *everyone*, developer and Operations alike, becomes concerned with the outcome of the product: getting it to production. There's no, "hey, we're done coding, let's hand it off to another team to deploy;" the *same team* is responsible for every aspect of the product's lifecycle.

Some team members might work on multiple product teams. A project manager might manage more than one products, for example, and an Operations specialist might serve on several product teams.

Each kind of role will often form a cross-team *guild*. For example, all of your developers might have their own guild, which meets regularly and discusses things of particular importance to their role, such as coding standards, decisions on which support libraries to adopt, and so on. These guilds create cross-team consistency, and they help pollinate new ideas, new standards, and new approaches.

The overall IT department will include other support teams that *aren't* part of the DevOps universe. Messaging systems administrators, some database administrators, some server administrators, and so on will be responsible for maintaining shared infrastructure. If they're not contributing to a product, then they're not DevOps. An organization can have many DevOps-style teams, and many *non* DevOps-style teams, as needed to support whatever functions they have.

Technologically, DevOps works a bit like this:

1. Developers check in code.
2. An automated *build pipeline* takes the code, compiles the project, and spins up a virtual testing environment.
3. Within that environment, automated *unit tests* run to test the code. Failures cause a report to be sent to the developers, and the pipeline stops.
4. A successful test run results in the project being deployed into production, often into a software deployment system of some kind.

Developers are in full control: they check in code, and if all the tests pass, the code goes live. Period. Anything less isn't "DevOps," although not being DevOps doesn't mean "valueless."

"What about a developer mistake that the unit tests don't catch?" Ah, this is where the real heart of DevOps lies. First, you might revert your source control to the previous version, let the automated build pipeline do its thing, and deploy that version. Then (and you could do this bit first, if you felt you could do so quickly), *you write a new unit test to catch this new bug*. You confirm that the existing code *fails* the test, thus validating the unit test. Then you fix the code until the test passes, at which point the build pipeline will also deploy it into production.

Automated tests are the "institutional memory" of development. They ensure that a bug only gets deployed *once*; after that, the revised unit tests will "catch" the bug if it comes up again. Automated tests never forget to test some particular condition (like people do).

Who is DevOps For?

DevOps is good for supporting Agile development teams that rapidly produce small, incremental new software builds on a fast and regular cadence. *That's it.* In fact, you can make a checklist out of it:

- Do we develop software on an Agile cadence?
- Do we produce new builds daily or weekly?
- Is each new build relatively small in terms of the changes it contains?
- Do we trust automated build pipelines, including automated unit testing?
- Are we willing to be cool when a bug is deployed, knowing we can quickly re-deploy the last-known-good version?

If you answer "no" to *any_of those, then DevOps might not be for you. All of those numeric benefits to DevOps _only apply* if you can answer "yes" to all of the above. You simply *can't* achieve the benefits without doing the whole deal.

Who is DevOps Not For?

Plenty of organizations should not use DevOps. Take a company that makes software used to run nuclear reactors. This is not "fast-moving" software where a degree of failure can be tolerated; this is not a DevOps product. Ditto the company that makes MRI controller software for hospitals. Ditto the bank that uses decades-old midrange computers, programmed in COBOL, to run their back end.

The one caution I would offer is, "well, our business can't survive or tolerate even the slightest momentary failure in our software, so we can't use DevOps." That's *probably* not true, because *all* software has a bug of some kind at some point. In most non-DevOps shops, those bugs take a long time to fix, even longer to manually test and verify, and even longer to deploy; the *point* of DevOps is to acknowledge that to fail is human, and to fail quickly, revert quickly, and move on is DevOps. I have never seen a single piece of software with zero bugs, and I've worked in some incredibly critical environments; with DevOps, you simply fail with more grace. This presumes, of course, that you're "doing" DevOps *because* you have an Agile team capable of rolling out rapid, small, iterative releases of their software. All the DevOps in the world won't help if it's supporting a team of legacy-style "Waterfall" coders who aren't allowed to write a line of code without six weeks of planning meetings. One of the reasons DevOps shops experience fewer bugs is because they're working in smaller increments of change. "Smaller" means "humans can wrap their heads around it easier," which means, "less likely to do it wrong."

What About "Infrastructure as Code?"

I personally dislike that term for its syntactical inaccuracy. "As" is used to introduce a simile, which is a kind of comparison. Infrastructure is not code, nor is code infrastructure; they do not serve the same purpose, and they are not comparable. It's like saying, "apples as wheelbarrows." There's no comparison.

A more syntactically correct phrase might be, "Infrastructure *from* code," which accurately describes the activity of writing computer code that results in the creation, modification, or maintenance of infrastructure elements. Code which provisions new virtual servers and deploys software to them is, "infrastructure from code."

Any organization can benefit from adding DevOps to "infrastructure from code" (IfC). The code, in this case, *is* the product, and while it might be created and maintained by people other than traditional developers, it's still code. They should still be checking it into source control, and they can absolutely use automated build pipelines—including automated *infrastructure validation* tests in lieu of unit tests—to validate that the code produced the desired results, and then pushing that to production.

As a means of configuring the infrastructure, IfC has distinct advantages, including the ability to potentially run the code to re-create an entire environment, in a new location, as a disaster recovery response. DevOps merely adds the automated build-test-deploy pipeline, just as it does for a more traditional Agile-cadence software product.

What Does DevOps Include?

Other than the team, which I've discussed previously here, DevOps shops use a variety of tools to create the automation that makes DevOps work.

You'll start with a source code repository of some kind, which might mean something like Git, GitHub, Microsoft Visual Studio Team Services, and so on. Some products are better-suited for a DevOps-style environment than others, so do your research.

Continuous Integration, or CI, tooling comes next. Brand names include Jenkins, Team City. CircleCI, TravisCI, and so on. These coordinate the creation of virtual environments for testing, running of unit tests, and so on. CI helps developers remain more productive by allowing them to *rapidly* and *continually* test their code, with the presumption that the automated unit tests serve as a kind of "functional specification" against which the code is tested.

Environmental tools can help, too, such as tools like Vagrant that help manage container-based environments for code to run in. These can facilitate spinning up consistent dev, test, and production environments, for example.

Software deployment is the last bit of the puzzle, and it differs widely depending on the product being deployed. Existing systems like Microsoft System Center Configuration Manager might be used for in-house software deployment, but if you're deploying to a cloud environment, you might use a cloud service specifically tailored to DevOps approaches.

Is DevOps a Job Role?

Nominally, no. However, DevOps does involve a lot of technology components, processes, and know-how, and it's not unusual for a "DevOps Engineer" to be the person in charge of making all that work. But you can certainly "do" DevOps without a "DevOps Engineer." It's entirely possible for a "normal" Operations team to take on those tasks, since DevOps is really just about smoothing the path between developers and deployment. It's safe to say, though, that you can't simply hire a "DevOps Engineer" and legitimately be "doing" DevOps; DevOps exists to support Agile software development, so you kind of have to have your developer team ramped up before being able to rapidly deploy their code will do you any good.

In Closing

DevOps can be a useful approach, and thousands of companies are successful with it right now. The advantages are clearâ€”but only if you're willing to commit to the entire picture. That means DevOps isn't for everyone, and the smartest thing you can do is decide, up front, if it's a path you and your organization should pursue for some of your products.