IN THIS CHAPTER

- Surveying the main tenets of DevOps
- Understanding DevOps values
- Seeing how your organization benefits

Chapter 1
Introducing DevOps

DevOps has transformed the way engineering teams collaborate to create and ship software. It's a broad and encompassing philosophy that inspires diverse implementations across the industry.

I define DevOps as an engineering culture of collaboration, ownership, and learning with the purpose of accelerating the software development life cycle from ideation to production. DevOps can enable you to reduce interpersonal friction, eliminate bottlenecks, improve collaboration, increase job satisfaction through engineer empowerment, and accelerate team productivity. DevOps is no silver bullet, but it can have massive impact on your organization and your products.

In this chapter, I emphasize the importance of culture over process and tooling, discuss the principles and values of DevOps, and dive into how your organization will benefit from a DevOps approach.

What Is DevOps?

This book has no exact DevOps prescription for you — because none exists. DevOps is a philosophy, one that that prioritizes people over process and process over tooling. DevOps builds a culture of trust, collaboration, and continuous improvement. As a culture, it views the development process in a holistic way,

taking into account everyone involved: developers, testers, operations folks, security, and infrastructure engineers. DevOps doesn't put any one of these groups above the others, nor does it rank the importance of their work. Instead, a DevOps company treats the entire team of engineers as critical to ensuring that the customer has the best experience possible. (You can find more about company culture in Chapter 2.)

DevOps evolved from Agile

In 2001, 17 software engineers met and published the "Manifesto for Agile Software Development," which spelled out the 12 principles of Agile project management (see the sidebar "The origins of Agile" in Chapter 7 for more details). This new workflow was a response to the frustration and inflexibility of teams working in a waterfall (linear) process. Working within Agile principles, engineers aren't required to adhere to original requirements or follow a linear development workflow in which each team hands off work to the next. Instead, they're capable of adapting to the ever-changing needs of the business or the market, and sometimes even the changing technology and tools.

Although Agile revolutionized software development in many ways, it failed to address the conflict between developers and operations specialists. Silos still developed around technical skill sets and specialties, and developers still handed off code to operations folks to deploy and support.

In 2008, Andrew Clay Shafer talked to Patrick Debois about his frustrations with the constant conflict between developers and operations folks. Together, they launched the first DevOpsDays event in Belgium to create a better — and more agile — way of approaching software development. This evolution of Agile took hold, and DevOps has since enabled companies around the globe to produce better software faster (and usually cheaper). DevOps is not a fad. It's a widely accepted engineering philosophy.

DevOps focuses on people

Anyone who says that DevOps is all about tooling wants to sell you something. Above all else, DevOps is a philosophy that focuses on engineers and how they can better work together to produce great software. You could spend millions on every DevOps tool in the world and still be no closer to DevOps nirvana. Instead, focus on your most important engineering asset: engineers. Happy engineers make great software. How do you make happy engineers? Well, you create a collaborative work environment in which mutual respect, shared knowledge, and acknowledgement of hard work can thrive. See Chapters 2 and 15 for more about how to create teams of happy, empowered engineers who embody a growth mindset and take pride in their work.

Company culture is the foundation of DevOps

Your company has a culture, even if it has been left to develop through inertia. That culture has more influence on your job satisfaction, productivity, and team velocity than you probably realize.

Company culture is best described as the unspoken expectations, behavior, and values of an organization. Culture is what tells your employees whether company leadership is open to new ideas. It's what informs an employee's decision as to whether to come forward with a problem or to sweep it under the rug.

Culture is something to be designed and refined, not something to leave to chance. Though the actual definition varies from company to company and person to person, DevOps is a cultural approach to engineering at its core.

A toxic company culture will kill your DevOps journey before it even starts. Even if your engineering team adopts a DevOps mindset, the attitudes and challenges of the larger company will bleed into your environment.

With DevOps, you avoid blame, grow trust, and focus on the customer. You give your engineers autonomy and empower them to do what they do best: engineer solutions. As you begin to implement DevOps, you give your engineers the time and space to adjust to it, allowing them the opportunities to get to know each other better and build rapport with engineers with different specialties. Also, you measure progress and reward achievements. Never blame individuals for failures. Instead, the team should continuously improve together, and achievements should be celebrated and rewarded.

You learn by observing your process and collecting data

Observing your workflow without expectation is a powerful technique to use to see the successes and challenges of your workflow realistically. This observation is the only way to find the correct solution to the areas and issues that create bottlenecks in your processes. Just as with software, slapping some Kubernetes (or other new tool) on a problem doesn't necessarily fix it. You have to know where the problems are before you go about fixing them. As you continue, you collect data — not to measure success or failure but to track the team's performance. You determine what works, what doesn't work, and what to try next time. In Chapter 3, you learn how to identify bottlenecks in your development process.

Persuasion is key to DevOps adoption

Selling the idea of DevOps to your leaders, peers, and employees isn't easy. The process isn't always intuitive to engineers, either. Shouldn't a great idea simply sell itself? If only it were that easy. However, a key concept to always keep in mind as you implement DevOps is that it emphasizes people. The so-called "soft skills" of communication and collaboration are central to your DevOps transformation. Persuading other folks on your team and within your company to adopt DevOps requires practicing good communication skills. Early conversations that you have with colleagues about DevOps can set you up for success down the road — especially when you hit an unexpected speed bump.

Small, incremental changes are priceless

The aspect of DevOps that emphasizes making changes in small, incremental ways has its roots in lean manufacturing, which embraces accelerated feedback, continuous improvement, and swifter time to market. When I talk about DevOps transformations, I like to use water as a metaphor. Water is one of the world's most powerful elements. Unless people are watching the flood waters rise in front of them, they think of it as relatively harmless. The Colorado River carved the Grand Canyon. Slowly, over millions of years, water cut through stone to expose nearly two billion years of soil and rock.

You can be like water. Be the slow, relentless change in your organization. Here's that famous quote from a Bruce Lee interview to inspire you (`https://www.youtube.com/watch?v=cJMwBwFj5nQ`):

Be formless, shapeless, like water. Now you put water into a cup, it becomes the cup. You put water into a bottle, it becomes the bottle. You put it in a teapot, it becomes the teapot. Now, water can flow or it can crash. Be water, my friend.

Making incremental changes means, for example, that you find a problem and you fix that problem. Then you fix the next one. You don't take on too much too fast and you don't pick every battle to fight. You understand that some fights aren't worth the energy or social capital that they can cost you.

Benefitting from DevOps

This entire book dives into how you and your team can benefit from implementing DevOps in your organization. Beyond the human component, which enables faster delivery, improved functionality, and fearless innovation, DevOps has technical benefits.

Continuous integration and continuous delivery (CI/CD) are closely aligned with DevOps. Continuous software delivery removes many of the bottlenecks often seen in teams that deploy infrequently. If you create automated pipelines that pass new code through a robust test suite, you can feel more confident in your deployments. (I talk more about CI/CD in Chapter 11.)

DevOps also enables faster recovery from incidents. You will inevitably experience a customer-impacting service disruption at some point, no matter how well tested your code is. But teams who work in a DevOps methodology find resolutions faster through better coordination, more open accessibility, shared learning, and better performance monitoring.

Engineering is not the only side of your organization that benefits from DevOps. The business side of your organization will see fewer customer complaints, faster delivery of new features, and improved reliability of existing services.

DevOps enables you to do more with the resources you already have. It accepts the reality of constraints and shows you how to succeed within your unique environment.

Keeping CALMS

As you begin to get familiar with DevOps, you'll likely come across a model called CALMS. It stands for culture, automation, lean, measurement, and sharing, and it's a helpful framework through which to understand the DevOps principles and evaluate your DevOps success as you apply those principles throughout your organization.

Culture

Your culture needs to be collaborative and customer centered, which means your engineers understand that the purpose of technology is to make your customers' lives easier. If customers don't find value in the product, the product will fail. Technology is secondary to this goal. The best DevOps cultures are extremely collaborative and cross-functional, with people from different teams and varying skill sets working together to engineer a better product. Listening is a major component of communication, and an easy litmus test of culture is to listen to conversations. Are people constantly talking over each other? If so, I bet you have opportunities for major cultural improvements ahead.

Automation

Rote tasks are an engineer's worst nightmare, not only because they're, well, boring, but because they're inefficient. Engineers speak computer so that they can make computers do the jobs that people don't want to do. Usually the lowest-hanging fruit

for improvements in automation are code builds, automated testing, deployments, and infrastructure provisioning. I dig deeper into identifying low-hanging fruit in Chapter 3.

Lean

Lean doesn't refer just to lean manufacturing. It applies more widely to the nature of DevOps teams, which are agile and scrappy. Lean teams eschew low-impact activity because it doesn't provide value to the customer. Another aspect of lean is how it keeps to the goal of continuous improvement. Everyone embraces a growth mindset and earnestly wants to improve.

Measurement

Data is critical to DevOps. Measuring progress through data will inform nearly every aspect of your organization's transformation. Keep in mind, though, that progress should never be tied to individual performance. Think of it as tracking your progress along an endless marathon rather than as a way of knowing when you're "done." You're never done. No one is.

Instead of regarding the data you collect as a measure of how poorly you're doing, think of it as gauging your improvement. Celebrate the wins. That approach bolsters the entire team and keeps your engineers happy, motivated, and productive. I guarantee you're doing some things well, and highlighting the good is important. I talk about what you can measure in Chapter 5.

Sharing

DevOps was founded because operations and development had some conflict. They lacked common ground and were incentivized based on different standards. Operations folks are typically measured on the reliability and availability of an application, whereas developers are, more often than not, incentivized to create new features for the application. (I talk more about how operations and development are measured in the next section.) You know what the biggest threat to uptime is? Deployments. Developers initiate deployments with new code releases. Thus, operations folks hate developers. Now, it's not usually that bleak, but there's a seed of truth in that. The friction makes solving problems nearly impossible and turns everything into a blame game. DevOps seeks to change that atmosphere completely and create an environment in which both teams teach each other and feel empowered — thus building a single team through which everyone contributes.

AN ENGINEER'S TALE: WHAT DROVE ME TO DEVOPS

I want to let you in on a little secret. I came to DevOps by accident. Yep! Totally an accident. But I think my story speaks volumes about the power of the DevOps movement and community.

I was a backend Java engineer at a small company with a traditional engineering team. The team consisted of a dozen developers and two operations folks. (Sounds to be about the usual ratio, right?)

The code had a bug. I updated the code that selected preview images in the application. Yet, the home page wasn't displaying the changes, and ops blamed me. I looked into it and concluded that it was a content delivery network (CDN) issue. Because of access constraints of the developers on my team, I couldn't mitigate the problem myself. I needed the ops team.

The ops expert felt that this was a code issue and refused to help me. We went back and forth three times before I went into a closet and angrily typed an abstract. *Humpty Dumpty: A Story of DevOps Gone Wrong* was my first tech talk and was inspired by my personal experiences and frustrations with developers facing off against operations folks.

In that company, and so many others, the operations team was a bottleneck. They prevented me from doing my work. It wasn't their fault, though. The people involved highlighted the problem, but the problem itself was a process issue.

My experience at that job led me to DevOps, which piqued my interest. In the course of learning about DevOps, I found incredible relief in the discovery that the issues I had encountered weren't about just me! I wasn't a bad developer. I was just a human, and other engineers felt the same frustrations in their jobs. It is my greatest wish that this book can both reassure you that your experience is valid and common as well as show you some approaches that can make your job just a little bit more awesome.

Solving the problem of conflicting interests

On traditional engineering teams, developers (those who write the code) and operations engineers (those who deploy systems and maintain infrastructure) are on opposing sides of a never-ending war. Okay, that's not exactly accurate. But they don't get along, and that's because they're measured by different criteria.

Developers are typically measured by the number of features they release or the number of bugs they fix. (Evaluating developers by lines of code written is a terrible idea. Many times, the best developers delete more lines of code than they add.)

Unfortunately, code quality and reliability aren't typically measured. As a result, developers naturally prioritize the work that will make them look productive. They don't spend time refactoring code to make it more readable or paying off technical debt accrued from the last big product push.

In contrast to how developers are measured, operations teams are typically measured by site reliability and uptime. You've likely heard of the five 9s: 99.999 percent availability. The five 9s means that your site can be down for only five minutes per year. Five minutes . . . *per year.* That's a lot to ask. It's also expensive to maintain because of the number of storage and compute resources you must have at your disposal, not to mention the personal impact it has on the operations individuals tasked to keep availability at that level. Those people are often asked to take on heroic efforts and respond to problems regardless of the day, time, existing workloads, or personal obligations.

To make the conflict clear: In traditional engineering organizations, developers must deploy new code to release new features. But deploys are the most common action that initiates service disruptions and site outages.

Two problems come from this situation:

- **Responsibility is siloed.** Developers don't know how to release or support their code, and they lack systems knowledge that enables them to understand infrastructure requirements. Most developers don't know (or care) how their code actually runs. Their job is done.
- **The goals and incentives are in opposition.** Developers toss code over the operations team and expect them to deploy the code and ensure that it runs perfectly. Operations folks are incentivized by uptime, availability, and reliability. They often assume that the code is poorly written and they'll be yelled at (or fired) for an incident that isn't their fault.

Do you see why you hear audible sighs when developers and operations teams interact? DevOps seeks to eliminate both the challenges created by siloed responsibility and opposing goals. By aligning incentives, sharing knowledge, removing barriers, and respecting different roles, DevOps can dramatically improve the interpersonal communication and cooperation on your team.