DevOps At Scale: Chapter 1

What is DevOps?

PLUTORA

Demands on software teams are as high as they've ever been. Businesses require that software has fewer bugs and more features. What's more, they need it faster than ever. Savvy software teams look for new ways to meet these challenges head-on. Sometimes, they try out new project management systems, like agile approaches. Other times, that means trying new tooling that makes it easier to manage their dependencies, like virtual application containers. They might transition to new programming languages or integrate new libraries.

One trend that's gained momentum over the past decade is adopting a DevOps style of resource management and deployment. Teams who find success with DevOps say that it helps them deploy more code more quickly—and they do so with less downtime and fewer issues. For anyone who's spent time following trends in technical management, DevOps is a concept that pops up repeatedly.

What Exactly Is DevOps?

DevOps is somewhat difficult to define, much like its cousin, agile software. The reality is that the definition will vary a bit from team to team. Each team will tweak DevOps methodology to suit their own unique needs. With that said, there are some unifying factors between DevOps implementations. For starters, DevOps means breaking through silos between your product management, development, and operations teams. Instead of working independently from one another, those teams will work together closely to define new features. Mature DevOps teams approach infrastructure management and deployment management by automating those processes.

It's important to remember that teams will be more or less mature in each of those categories. A team will naturally progress more quickly in one area over another when adopting a DevOps culture. There is no singular DevOps authority to say which teams "do DevOps." However, the most effective teams will move toward maturity in all of those areas. Their primary goals will be to remove walls between teams so that they can work more closely together. They'll seek to automate code testing and integrate security checks into the process of building code, instead of saving it for when the code is "done." The traits of highly successful DevOps teams are pretty easy to spot, even when they might not do everything perfectly.

Why Do Teams Choose DevOps?

In a traditional software life cycle, operations staff are left off projects until the very end. Their job is to deploy code that developers have written, with minimal input on the code's shape or behavior. Anyone who's worked in software for a decent length of time has seen the problems this approach causes. In the worst cases, the project itself needs to be reset because the code the developers delivered simply can't run in the environment mandated by the business.

Needless to say, that kind of outcome is a serious problem. It's almost always a guaranteed failure for the project. To eliminate these kinds of problems, the development team involves operations staff much earlier in the project's development. They work side by side with developers to make decisions about things like infrastructure requirements and software libraries.

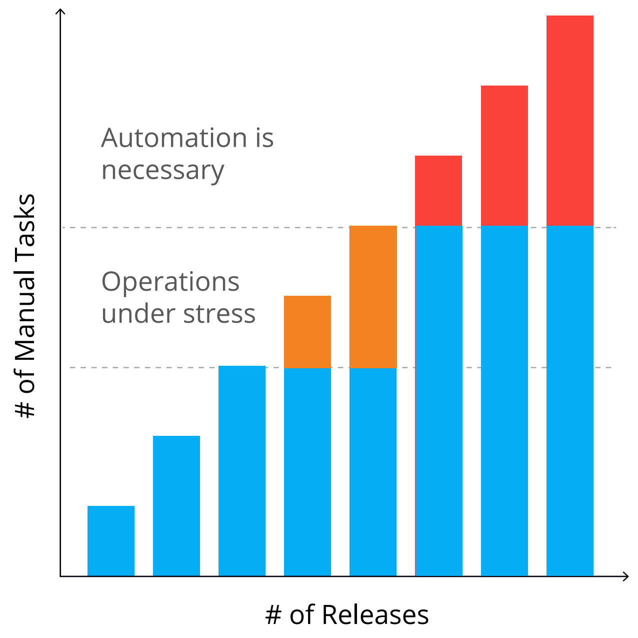
At the same time, many businesses transitioned to agile development methodologies. Those methodologies eschew making big up-front plans for projects, instead preferring an iterative development style.

When you combine these two mentalities, you create an environment where developers are working closely with operations staff to develop code and deploy it very quickly. That, at its heart, is what DevOps is all about. Teams that adopt a DevOps approach to code report a number of serious benefits.

How Do Agile and DevOps Fit Together?

The DevOps movement grew out of the agile software movement. As teams moved further away from big, world-altering releases, things became more difficult for operations teams. A team that releases every three months doesn't experience a lot of extra stress from any individual part of a release. That same team releasing code to production every two weeks will quickly find that there are parts of their release process that simply don't scale. Operations staff find themselves repeating time-consuming and tedious manual tasks for each release.

So, savvy operations teams began automating as many parts of their releases as they could. These teams embraced one of the core tenets of DevOps—continuous improvement—to constantly look for new ways to make their deployments simpler. Instead of manually provisioning servers, they started defining their environments using code. They adopted continuous integration systems to constantly test new code for fitness. Above all, they engaged teams outside the operations staff to knock down barriers to deploying new code safely.

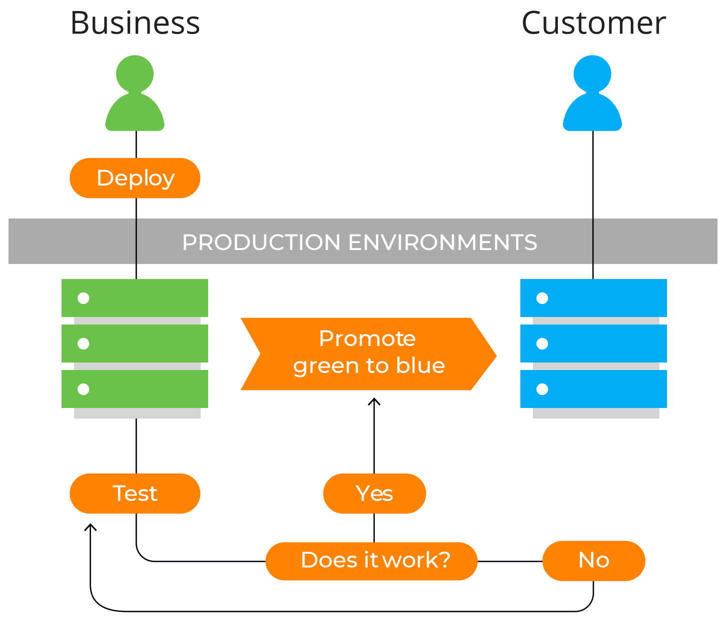


Make deployments simpler

1. Define environments using code
2. Adopt continuous integration
3. Work closely with other teams to define features

What Are the Core Tenets of DevOps?

As noted, each team defines DevOps a little bit differently. There is no one true DevOps method. With that said, there are common attributes that nearly all DevOps teams share. Each of these attributes is important and plays a key role in delivering the kind of results that DevOps teams love.



Use blue-green deployments to reduce downtime and risk

Continuous Integration / Continuous Delivery (CI/CD)

Continuous integration and continuous delivery (CI/CD) are a core part of the DevOps workflow. The goal of CI/CD is to reduce the time between code being written and deployed. In traditional waterfall project management teams, it might be months between a developer writing code and deploying it. Even in an agile team, the time window between writing code and deploying it can be weeks. CI/CD aims to reduce this time window down to days or even hours.

CI/CD itself is fairly simple in practice. It requires that developers write extensive software tests to confirm that when they make a change, the codebase still behaves properly. These tests run automatically every time the developer pushes code to source control. That's the CI part. When those tests are well designed, the development team is confident that the code they write will work properly in production.

Because the development team is confident about their code, the operations team can deploy as soon as tests pass. This is the CD part of CI/CD. On teams with mature CI/CD processes, these deployments can happen dozens of times per day. That kind of turnaround means customers are using new code as soon as possible.

The most mature software teams practice blue-green deployments, where they have two production environments running simultaneously. When the team releases a new version of the software, they deploy to the "green" instance while the "blue" handles customer requests. Once they've tested "green" to ensure it works, they promote "green" to "blue"—making the version that was previously the primary production environment into the secondary environment. Now the new version of the code is handling requests from customers without even a second of downtime. Advanced teams even use feature flags to ship unfinished features to production, but avoid turning them on. The newest features are always on production, and can be switched on as soon as they've completed testing, without needing a deploy.

Infrastructure as Code

CI/CD carries some important requirements. For starters, teams need it to be trivial to deploy any commit within an application. That requirement means that applications can't have laborious deployment processes or difficult-to-install libraries. If it takes an hour of labor to spin up a server for a service, it's very difficult to deliver the aforementioned blue-green deployments.

To solve this problem, operations teams adopt infrastructure as code (IaC). IaC is a system whereby operations teams define the services necessary to build and run an application directly within the code of the application itself. Then, when the code has finished testing, a software library translates that code into running servers, hard-working databases, and open network connections—in short, everything the team needs to deliver their work to the customer.

IaC is the core of any CI/CD pipeline. When a team truly adopts IaC, deploying a new service requires minimal to no intervention from a dedicated operations engineer. This is what we mean when we talk about operations teams adopting the patterns of the software development team. Instead of babysitting a server and making sure it's running smoothly, operations staff develop code to address problems before they ever crop up.

Cross-Team Collaboration

At its heart, DevOps enables collaboration between development and operations teams. The two teams work closely together to plan and code both the logic and environment that a new feature will run in. Instead of a horrible project in which the operations team has to tell stakeholders that a project needs to be restarted, they're involved with the planning from the word "go." This collaboration is the successful outcome of a DevOps culture, but it's also the grease that keeps the teams' wheels turning.

Effective DevOps teams don't end their collaboration just between developers and operations. Because the technical team can turn around new feature requests much more quickly, it's easier for technical teams to collaborate with other parts of the business. Instead of waiting months to see a new bit of software deployed, stakeholders can test new code within days or weeks of requesting it. This means that business teams place more trust in technical teams, and both teams can plan more effectively.

A healthy DevOps culture approaches the ideal of an agile software environment. The technical teams sit down with business stakeholders on a regular basis and hear what the business needs them to work on. They then do a bit of planning and hop right into coding. That code is deployed as soon as it's finished, and the business can start using it right away. Stakeholders within the business then collect feedback on the new code and formulate new requests from the technical team. The process begins anew, and the software gets better at every step.

System Performance Measurement

Once they've adopted DevOps for a few weeks or months, some organizations start to feel stuck. They have a difficult time knowing whether their approach is working or they should tweak it. Their teams started with grand ideas of CD and pain-free deployments, but the reality has been far from ideal. Instead, it's been a lot of hard work, and their software doesn't feel like it's progressed nearly as quickly as expected.

Many organizations abandon the experiment here. They simply forsake their new processes and go back to the comfortable confines of their previous project management systems. The best organizations take a different approach, though—they seek to measure their DevOps team through the adoption of targeted metrics.

Some of these metrics will be on the process side. The team will measure things like how long code sits between the developer committing it to the repository and the operations staff deploying it. They'll measure how much accumulated downtime accrues due to deploying new software, and the rate of new defects in the code. Plutora's platform tracks these kinds of metrics, making it easy to see where your team is succeeding as they adopt new iterative processes.

Many metrics will also be on the technical side. They might measure average response times for a critical service between different deployments. That'll tell the team whether new features meaningfully degraded the overall performance of the system.

By measuring these things, effective management teams can build a cohesive picture of the health of their system and their development team.

Product-Oriented Development

In many enterprises, projects rule the day. Each new version of software is developed via a project methodology. A project management office organizes the features that they think need to be included. Eventually, after several rounds of politicking, these are sent to the developers, who implement them without any feedback from stakeholders. Once the code is "finished," it moves on to all sorts of regulatory checks, where security and compliance teams ensure it meets external requirements.

After passing all those gates, only then does the software move on to the testing team. Once again, a significant amount of code is rewritten to fix bugs. The process repeats itself with user acceptance testing. If any bugs slip through to the end user, they'll need to wait for fixes until the next release comes out—if the project management office includes those fixes.

Product-oriented development turns those systems on their heads. Instead of planning all the features of a release at the beginning, product-oriented development seeks to deliver the simplest set of features that can meet their customers' immediate needs, then improves them over time. The development time is reduced significantly because releases aren't bloated with constant inclusions of features that the PMO determines can't wait for another few months. Compliance timelines are shortened because there is less code to evaluate, and testing is simpler for the same reason.

The change to product-oriented development increases the release cadence significantly. Not coincidentally, adopting a product orientation is a core part of moving to an agile development culture. Those same changes are critical for teams adopting a DevOps mindset and its focus on faster releases and continuous improvement.

Continuous Improvement

All of these individual tenets come together to work for the cornerstone of DevOps: continuous improvement. Continuous improvement is the principle of striving to do better at what you do every time you do it, and it's the cornerstone of DevOps because all the other tenets build off of it. CI/CD means that it's easier to continuously improve the overall quality of the code. IaC makes it easier to continuously improve the deployment process. Cross-team collaboration means that it's easier to make sure the team is always working on the most valuable features and fixes.

At the core of continuous improvement is system performance measurement. Teams that know which parts of the process to measure and do so effectively can target their work to improve those processes. Teams that don't know how or what to measure fail at creating an effective DevOps culture.

The hard part of adopting a continuous improvement mindset is that it's something that each member of the team needs to do. Each team member needs to be motivated to constantly look for ways to improve their craft. The team members need to be empowered to hold one another accountable when another team member doesn't do their best work. This doesn't have to mean calling each other out for poor performance, but it does mean that team members need to be able to recognize that something isn't up to par and stop the team from deploying that code until it's fixed.

How Can a Team Start With DevOps?

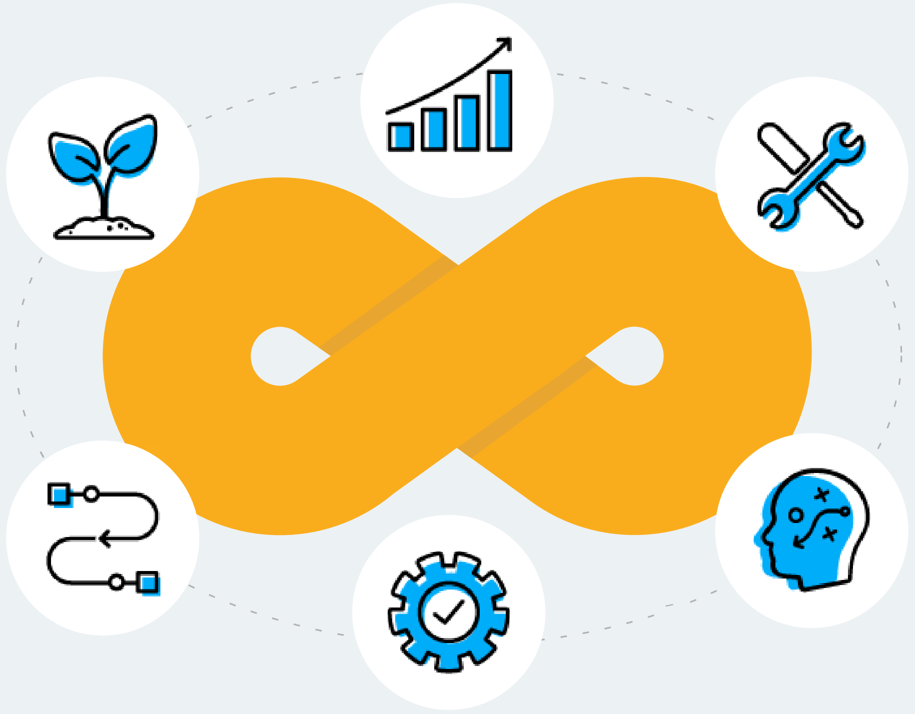
Starting down the DevOps path doesn't need to be difficult or a big project. Remember that the grease between the gears of a DevOps team is collaboration, and the cornerstone principle is continuous improvement. It's possible to take the first step toward a DevOps culture simply by opening lines of communication between developers and operations staff. Inviting administrators to a planning meeting to solicit their input on a new feature goes a long way. It builds goodwill and starts a habit of collaboration between teams. That team can do a little better in subsequent planning cycles by limiting the size of features and building on lessons learned from their previous deployments.

Once teams have started collaborating, the rock starts rolling downhill. Team members will identify new ways to improve processes. Those improvements feed into other processes and other parts of the code. The result is a team that's adopted the principle of continuous improvement. It's important to remember that the concepts surrounding DevOps didn't spring fully formed from the head of Zeus. They were developed by real people doing real work over years of iteration. Any team can walk that same path and learn many of those same lessons over time. Team leadership that recognizes the benefits of the DevOps culture will step up and learn from people who've gone before to help their team adopt DevOps more quickly.

DevOps Is Never Ending

The start of this paper posed the question, what is DevOps? The truth is that DevOps is many things to each team that adopts it. The universal constant, though, is that DevOps isn't a fixed target. Each team must constantly adjust their expectations and practices to improve their results. Even teams that appear to have their entire DevOps approach figured out will regularly reevaluate their processes.

The other parts of this series of white papers explore the different ways that mature teams seek to improve their DevOps processes. Some will focus on processes, some on technology, but all of them will cover how to think about improving DevOps over time. That's because DevOps is a never-ending process of team improvement. Each team has to investigate how to do that for themselves, and it's something they'll figure out eventually. The good news is that there are many people who've come before who can help show new teams the way.



Want to learn more about DevOps?

Check our series of white papers about DevOps to learn from the foundations, to the cultural change and maturity model.

1. What is DevOps?
2. The Benefits of DevOps Strategy
3. DevOps Methodology: Aligning your Organization
4. DevOps Tools: Why You Need Them
5. The DevOps Maturity Model
6. DevOps Pipeline: The Functional Building Blocks
7. Mastering the DevOps Process
8. Making your DevOps + Agile Transformation a Success

Visit www.plutora.com/devops to learn more.

About Plutora

Plutora, the market leader of value stream management solutions for enterprise IT, improves the speed and quality of software creation by capturing, visualizing and analyzing critical indicators of every aspect of the delivery process. Plutora orchestrates release pipelines across a diverse ecosystem of development methodologies, manages hybrid test environments, correlates data from existing toolchains, and incorporates test metrics gathered at every step. The Plutora Platform ensures organizational alignment of software development with business strategy and provides visibility, analytics and a system of insights into the entire value stream, guiding continuous improvement through the measured outcomes of each effort.

PLUTORA

Learn more: www.plutora.com

Email: contact@plutora.com