What is DevOps Culture?

How DevOps culture helps align people, processes, and tools toward a more unified customer focus.

TOM HALL
DevOps Advocate & Practitioner

DevOps is an agile approach to organizational change that seeks to bridge traditional, siloed divides between teams and establish new processes that facilitate greater collaboration. While DevOps is made possible by new tools and agile engineering practices, these are not enough to gain the benefits of DevOps. Without the right mindset, rituals, and culture, it's hard to realize the full promise of DevOps.

> People and culture are the top factors of a successful DevOps implementation.
> - Atlassian 2020 DevOps Trends Survey

What is DevOps culture?

At its essence, a DevOps culture involves closer collaboration and a shared responsibility between development and operations for the products they create and maintain. This helps companies align their people, processes, and tools toward a more unified customer focus.

It involves cultivating multidisciplinary teams who take accountability for the entire lifecycle of a product. DevOps teams work autonomously and embrace a software engineering culture, workflow, and toolset that elevates operational requirements to the same level of importance as architecture, design and development. The understanding that developers who build it, also run it, brings developers closer to the user,

RELATED MATERIAL

What is DevOps?

See article →

with a greater understanding of user requirements and needs. With operations teams more involved in the development process, they can add maintenance requirements and customer needs for a better product.

RELATED MATERIAL

Learn about the benefits of DevOps

See article →

At the heart of DevOps culture is **increased transparency, communication, and collaboration** between teams that traditionally worked in siloes. But there are important cultural shifts that need to happen to bring these teams closer together. DevOps is an organizational culture shift that emphasizes continuous learning and continuous improvement, especially through team autonomy, fast feedback, high empathy and trust, and cross-team collaboration.

DevOps entails **shared responsibilities**. Development and operations staff should both be responsible for the success or failure of a product. Developers are expected to do more than just build and hand off to operations -- they are expected to share the responsibility of overseeing a product through the entire course of its lifetime. They test and operate software and collaborate more with QA and IT Ops. When they understand the challenges faced by operations, they are more likely to simplify deployment and maintenance. Likewise, when operations understand the system's business goals, they can work with developers to help define the operational needs of a system and adopt automation tools.

Autonomous teams are another important aspect of DevOps. For the development and operations teams to collaborate effectively, they need to make decisions and implement changes without a cumbersome and lengthy approval process. This involves handing over trust to teams and establishing an environment where there is no fear of failure. These teams should have the right processes and tools in place to make decisions faster and easier, for each level of risk to the customer.

For example, a typical development workflow might require engagement from several contributors on different teams to deploy code changes. The developer makes a code change and pushes it to a source control repository, a build engineer builds and deploys the code to a test environment, a product owner updates the status of the work in an issue tracking tool, etc. An autonomous team will take advantage of tools that automate these processes, so that pushing new code triggers the building and deployment of a new feature into a test environment and the issue tracking tool is updated automatically.

For example, a team is handicapped by requirements like having to open a ticket with a separate operations team to make a trivial infrastructure change, such as a new DNS entry. A task that should take seconds to complete ends up taking days or weeks to satisfy. An autonomous team has the ability to implement such changes themselves, whether by having an individual on the team who has the correct skills and experience, or by having access to self-service tooling.

A DevOps team culture values fast **feedback** that can help with continuous improvement of a unified development and operations team. In an environment where the development and operations teams are in isolated silos, feedback about the performance and stability of application software in production is often slow to make it back to the development team, if it makes it at all. DevOps ensures that developers get the fast feedback they need to rapidly iterate and improve on application code by requiring collaboration between operations folks in designing and implementing application monitoring and reporting strategies. For example, any sufficiently capable continuous integration tool will enable automated build and testing of new code pushes and provide the developer with immediate feedback on the quality of their code.

Automation is essential to DevOps culture, since it allows great collaboration and frees up resources. Automating and integrating the processes between software

frees up resources. Automating and integrating the processes between software development and IT teams helps them to build, test, and release software faster and more reliably.

What are the benefits of DevOps culture?

The most obvious and impactful benefit of embracing a DevOps culture is streamlined, frequent, and high-quality software releases. This not only increases company performance, but also employee satisfaction.

A DevOps culture fosters high levels of trust and collaboration, results in higher quality decision making, and even higher levels of job satisfaction, according to the book “Accelerate: Building and Scaling High Performing Technology Organizations.”

Embracing a DevOps culture is key to building a high-performing engineering organization without sacrificing employee contentment. It’s a win-win. For an engineer there is nothing like the feeling of frequently and easily deploying and running stable, high-performing software that makes its users happy, and for executives the improved business outcomes are a hit.

What are the challenges?

Fully embracing a DevOps culture usually requires individuals and teams to make significant changes to how they work, and therefore requires buy-in at the highest levels of the organization.

A grassroots effort can be, and often is, an important starting point for getting management and executive-level buy-in for a DevOps transformation. Often the most compelling argument in favor of broader DevOps adoption is when a few individuals or small teams adopt a DevOps approach and begin demonstrating success.

The high levels of autonomy and trust that are typical in a DevOps culture can be difficult to cultivate if there is a history of conflict between any of the individuals or teams involved. The more siloed the teams were before attempting to adopt a DevOps approach, the harder it will be to build connections.

Change is hard. Even in environments where there is a high level of harmony between the existing individuals and teams, if the benefits of change aren't clearly articulated and understood, it can be difficult to drive acceptance and willingness to put in the work.

Understandably, organizations with a strong engineering mindset often jump immediately to tools and technologies to solve business challenges. Yes, there are tools and technologies that can help your organization transition to a DevOps approach. But changing tools and technologies without changing the culture is often called “cargo-cult DevOps” since it changes the facade without addressing the weakness in the foundation.

Considerations for transitioning to DevOps culture

Open communication

One of the most fundamental challenges DevOps seeks to combat is the siloing of knowledge, experience, and work in different organizational units. When the programmers who write code and the system administrators who deploy and maintain it don't communicate, you likely have inefficiencies.

The ability to make mistakes

Many organizations, teams, and individuals put extraordinary pressure on themselves and each other to never make mistakes. If failure is not an option, an individual or team is less likely to attempt a novel approach to solve a problem or develop innovative features.

This mindset is reflected in the past obsession with measuring "Mean Time Between Failures" (MTBF) over "Mean Time to Recovery" (MTTR). MTBF uses tools like "root cause analysis" to identify the source of failures and attempt to prevent them from happening again. MTTR reflects a view of software applications as complex systems that are apt to fail in unpredictable ways and focuses on quick recovery when they do fail.

A "blameless retrospective" is a common feature in DevOps culture. Outcomes can be improved when a team meets at the end of a sprint or project to discuss what went well and what could be improved, in an open and safe environment.

> A blameless view of failure works so well in part because it adopts a growth mindset, acknowledging that mistakes happen but operating under the assumption that both people and organizations are capable of learning, growing and improving.
> - "Effective DevOps" by Jennifer Davis and Katherine Daniels

A new set of processes

Cultivating a DevOps culture requires developing new approaches to old problems. DevOps entails changing the siloed process of programmers writing application code and "throwing it over the wall" to an operations team who deploys and operates the application. The DevOps approach entails development and operations working together throughout the entire lifecycle of a project.

Continuous integration and continuous delivery (CI/CD) are commonly believed to be necessary to a DevOps culture. A third process, continuous deployment, is embraced and promoted by such large organizations as Netflix but not commonly adopted (or required) in most smaller companies. This is because continuously deploying new features into a production environment requires a high degree of confidence that new code has been thoroughly tested and can be deployed safely (e.g. behind a feature toggle). So, unless your organization deploys many times a day it may not be worth investing in the processes that support this approach.

In most cases, doing some variation of "trunk-based development" will vastly simplify your CI/CD effort. In this model, the team does away with long-lived feature branches and makes frequent commits to the "trunk" branch of the code.

An important component of trunk-based development is comprehensive automated testing: unit, integration, and regression. This helps to ensure that all new commits to the trunk branch have been thoroughly vetted when pushed to the repository.

Continuous integration is the process of automating the integration of code changes from multiple contributors in a software project. This extends beyond development teams to the rest of the organization. For example, product teams coordinate when to sequentially launch features and fixes and which team members will be responsible.

Continuous delivery is an organizational methodology that brings together engineering and non-engineering teams like design, product, and marketing to deliver a product. Environments without CD encourage “over the wall” behavior where developers focus on the QA team as the primary user experience. It means the “trunk” branch of your repository is in a “deployable” state at all times.

Continuous deployment allows code changes to be automatically deployed into production when they are made, either hidden behind a feature flag, deployed to a small percentage of customers, and/or easily rolled back. This gives teams greater flexibility to respond to changing markets and customer demands since teams can react to customer feedback and rapidly deploy and validate new features. They can also easily rollback features, allowing teams not to be hampered by breaking the build.

Feature flag, feature toggle, or dark deployment are common ways of ensuring new application features do not appear or function when deployed to the production environment, but can be turned on with minimal effort. This strategy enables continuous deployment because there is very little risk of adversely impacting users. It's also common to restrict features to a subset of the user base by segmenting them by geography or running separate server instances and releasing features to only one server that is accessible to users.

An updated toolchain

Most software development teams are using at least some type of version control, issue tracking, and application monitoring tools. All of these are important tools to support a DevOps culture, but perhaps the most important addition to the traditional toolset is software to support CI/CD. Having an automated workflow that takes a commit, tests, and deploys it is really the only way to get the fast feedback a DevOps culture requires.

DevOps - a cultural shift that yields results

Developers have been chasing the dream of delivering software more frequently, with less effort, and fewer bugs for decades. Now, the tools and practices to make this a reality are finally here.

Atlassian found that organizations practicing DevOps say they ship higher quality deliverables (61%), with increased deployment frequency and faster time to market (49%). And it’s not just organizations who reap the benefits, practitioners say they’ve learned new skills (78%) and received a raise (48%).

Cultivating a DevOps culture can be challenging, but the rewards in increased satisfaction for developers, managers, and customers alike are worth it.

Are you looking to improve your DevOps culture? Start with the Service Team Health Monitor. Also, practice communicating, collaborating, and brainstorming with colleagues with the Top 4 Plays for Building a DevOps Culture.

TOM HALL

Tom Hall is a DevOps advocate and practitioner, voracious reader, and amateur pianist.
Among his accomplishments over the past 20 years are certifications from Novell, EMC, VMware, and AWS. He helped organize DevOpsDays in Atlanta in 2016 and in Austin, TX in years since.

SHARE THIS ARTICLE

NEXT TOPIC

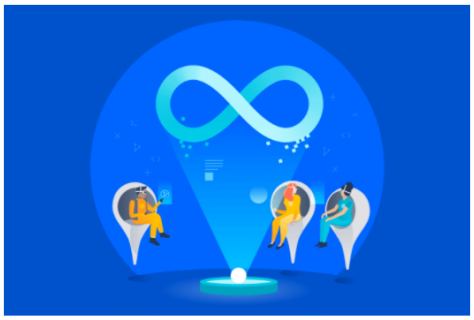
DevOps Best Practices →

Recommended reading

Bookmark these resources to learn about types of DevOps teams, or for ongoing updates about DevOps at Atlassian.

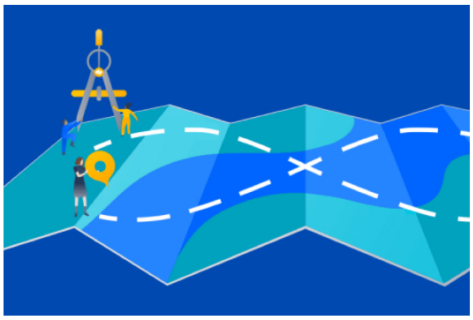
DevOps community

Learn more →



Simulation workshop

Learn more →



Get started for free

Learn more →

Sign up for our DevOps newsletter

Email address

Sign up