GIGAOM PRO

Breaking down barriers and reducing cycle times with devops and continuous delivery

By Paul Duvall

November 1, 2012

This research was underwritten by New Relic.

cloud

TABLE OF CONTENTS

Executive summary

How is it that companies like Netflix, Amazon, Etsy, and Facebook regularly deliver new features to their users (in some cases, several times per day) while other companies must wait months or more to release software updates? The answer is that the companies listed here, like many others, have changed how they deliver software by dismantling the traditional silos that stifle collaboration and progress. Their development and operations teams are working together and learning from each other through what is now known as "devops," an industry buzzword that arose to describe a philosophy that many progressive companies and startups had long been practicing.

Some of these companies probably never called their approach "devops." They were just working collaboratively to meet business demand. As they did, they began considering leaner and more-innovative approaches for releasing software more quickly and more often.

This report describes the nascent topics of devops and continuous delivery while enumerating the challenges IT organizations are seeking to remedy by adopting these approaches. Aimed at technology executives and directors and those responsible for delivering features and stable software systems to their users, the report covers the benefits of a devops mindset, which encourages communication and collaboration by obliterating the silos that impede projects. Additionally, it cites the value provided to the business itself, how the approach works, and some of the components and tools that are used on projects to deliver new features to users in a stable environment. Finally, it touches on the concept that the future will involve less friction and more value from tools and infrastructure that support this approach.

What is devops? What is continuous delivery?

The best way to define devops might to define what it is not. Devops is not a methodology, a tool, or a process. It is a response to a continual frustration: a lack of collaboration and communication between development and operations on software projects that increase the time and labor involved in delivering and maintaining software systems. Devops is based on the belief that traditionally separated teams — with separate budgets and agendas — can learn best practices and patterns from one another. Moreover, devops seeks to improve and increase this communication and collaboration by breaking down the barriers and silos that often exist in traditional organizations. As these barriers are being undone in some organizations, traditional roles on projects are being redefined.

Many of the tenets of devops are influenced by the business that brings the two teams together. (For the purposes of this report, the "business" is the part of an organization that derives the most benefit — monetary or other — from the features of the software system.) Companies such as Netflix, Amazon, Etsy, or Facebook must continually deliver highly stable features, faster and more often. As a result, a business asks more and more from its IT organizations, which in turn requires that the two traditionally separate teams work together to enable the business to deliver more features while continually ensuring the stability of its systems.

Continuous delivery (CD) is the automated implementation of the build, deploy, test, and release processes. CD provides quick automated feedback on the production-readiness of a release candidate, with every change committed to the version-control system. CD embraces the devops philosophy and defines the steps and activities for regularly delivering features to users in stable environments.

As more teams embrace the devops philosophy and implement continuous delivery on their projects, more platforms and services will move toward a self-service model that encourages this collaboration.

Challenges for devops

While many development teams are now applying agile software development practices to projects, the development and operations teams on a given project are not always aligned. This results from competing incentives between features and stability and between incongruent processes and tools across teams and misaligned organizations.

Features vs. stability

Generally two organizations are responsible for delivering software to users: development and operations. Development teams are measured on the features they deliver to users, while operations teams are measured by the stability of the system after it has been delivered to users. These are often competing interests in organizations because the development team is not incentivized to ensure the stability of a system once it delivers it to the operations team. Likewise, the operations team is not afforded the appropriate incentives to care about the frequency of releasing new features to users, because its primary responsibility is ensuring the uptime of the system.

Incongruent processes and tools

Just as interests between features and stability often compete, tools and processes are often misaligned about satisfying an organization's interest in regularly delivering features. For instance, development and operations teams regularly have completely different sets of tools and processes for delivering software.

Table 1 is an example of the different development and operations toolsets and processes within an organization. Not all tool types or processes are shown. The first column shows the type of tool or process, the second column lists the corresponding tool or process that the development team used, and the third column shows what the operations teams used. This table exemplifies the disparity that often occurs as a direct result of reporting structures and the hierarchy within an organization.

Table 1. Example of an organization with incongruent processes and tools

Tool/process type	Tool/process: dev	Tool/process: ops
Issue tracking	JIRA	Custom issue-tracking tool
Version control repository	Subversion	ClearCase
Testing	JUnit	None
Monitoring	None	Nagios, dynaTrace, and several others
Static analysis	Sonar, etc.	None
Build	Maven	N/A
Deployment process (manual)	Excel spreadsheets and JIRA	Word
Deployment orchestration	Jenkins	AnthillPro
Infrastructure provisioning	Word	Word (but different processes for each)

Source: GigaOM Pro

It is typical to find projects in which the infrastructure provisioning and deployment are completely different for each team. For example, development and quality assurance (QA) might be using a tool such as Jenkins to run the build and deployment against an environment. Maven actually performs the build

and Capistrano performs the deployment, so several different developers (with some help from one of the operations engineers) would manually configure these development and QA environments over a period of a few weeks. Anyone on the team has the authority to modify either environment manually.

Several weeks later, the software is delivered to operations; once again, the engineers use a completely separate set of processes and tools. In this case, they use AnthillPro to run the deployment, which is a set of bash scripts and some manual configuration changes. The environment setup also consists of a concoction of bash scripts and manual configuration changes. These disparate processes and tools on the same project occur far too often and lead to problems in one environment that cannot be replicated in another. This situation, in turn, results in teams discovering real issues late in the process and thus being unable to assess potential problems appropriately and earlier in the delivery cycle.

Misaligned organizations

The development team's success is usually measured by the features it delivers. As a result, development usually considers its work done once it commits its code to the version control system or once QA has signed off on it. The development team usually reports to the director of engineering, the CTO, or another executive.

On the other hand, the operations team is often measured by the stability of the software system once it is delivered (and typically reports to the director of systems or operations CIO). Operations usually receives the software a few weeks or even a few days before it must deliver it to production. Because of the incongruent processes and tools and this organizational misalignment, defects are often discovered very late in the life cycle. Many times the operations team's knee-jerk reaction is to plan each detail for the next release at a minute level, but this plan frequently fails too, as the decisions are made too early in the process and are thus based on best guesses and not how the software system will actually be constructed.

Collaboration between development and operations

The most fundamental change in collaboration is for organizations to recognize they only have one team: a delivery team. An organization does not consist of development versus operations or database administrators (DBA) versus QA. The one delivery team is composed of developers, QA, DBAs, business analysts, operations engineers, and, ideally, customers or proxy users. This is contrary to the way most teams are designed; usually these isolated groups do not communicate until a few weeks prior to a release (if then). Collaboration means that software systems, teams, and individual team members act as more-holistic units than in traditional organizations that do not embrace devops.

Holistic software systems

Until all team members begin to view software systems more holistically, collaborating is just talk. The team must see the software system as the sum of its parts; it is not only the application source code but also the infrastructure, data, and configuration. For example, when a configuration or data change has an effect on the infrastructure, every team member should care. In this way they will stop thinking of certain tasks as outside their job description. While not every member will work to fix a specific problem, all team members should realize that any change affecting the viability of a release candidate affects each team member.

Poly-skilled team members

Another way delivery team members can be more collaborative is by increasing their skill sets in disciplines beyond those traditional specialties that have been established on projects. For example, developers might broaden their skills in infrastructure and databases, while operations engineers could increase their skills in software development and testing.

Cross-functional teams

A cross-functional team consists of engineers, managers, and analysts across disciplines, including development, quality assurance, database administration, business analysis, and operations, all working together as part of the same team. When this happens, the best practices and patterns from developers are applied to what traditionally were tasks for operations (e.g., test-driven infrastructures) and vice versa. This approach elevates the skill sets, ability to collaborate, and competency of all team members.

Benefits of the devops approach

A primary benefit of applying the devops approach is that the team can learn about problems and fix them much earlier in the process. The earlier a problem is detected, the less complex it is and the less time it requires for fixes.

Learning about problems earlier

Quick feedback shrinks the time between the introduction of a defect and when it is discovered and fixed. When employing continuous delivery, both the development and operations teams are notified when any problem is introduced into any part of the software system, whether the problem occurs in the application code, infrastructure, data, or configuration. Using this approach, team members learn of problems much earlier than they would during traditional projects, when the software is "thrown over the wall" at the conclusion of coding (which might be weeks or months after the source code was written).

Less-complex problems

Because the frequency of change is almost constant with teams who embrace devops, the change sets are smaller. The software is regularly integrated with tests, so when problems do occur, they are often less complex than they would be during traditional projects, because the defect may have only been introduced a few minutes prior to its discovery. Thus, fewer changes will have occurred. That proximity of introduction and discovery makes problems easier to troubleshoot.

Less time fixing problems

Because teams that embrace devops are cross-functional, team members also spend less time fixing a problem: They do not need to wait for a separate team in a different part of the organization to troubleshoot and fix it. Everyone is on the same team. With changes occurring more frequently and in smaller batches, problems are less complex and team members spend less time assessing and fixing defects.

Business value

Most of the reasons why teams should adopt devops and continuous-delivery philosophies revolve around reducing the life cycle of software delivery. With continuous delivery, teams can deliver more features more often. The systems are more stable, and the software is always ready to release.

Deliver features more often

Since all the steps required to deliver software are automated in continuous delivery, a software release can occur with every good change (i.e., a change that does not fail the collection of scripts and tests). Automating the process removes the operational constraint of needing a separate effort to release software, so the business can choose when to release its latest features to users.

More-stable systems

Because every change can be tested against the entire canonical software system (not just the application code), systems are more stable. As with problem solving, this is because as soon as an error is introduced into the shared code base, team members receive feedback. As previously mentioned, this feedback includes application code, data, and configuration changes. Moreover, as test-driven techniques are applied to the infrastructure and data, exposed errors are discovered and resolved quickly.

Software is always ready to release

With a staged and automated-delivery pipeline, releasing software does not require a separate effort, so the most recent version is always ready for release to users. The business can then choose when these features are released.

How devops and continuous delivery work

The most obvious way to apply a devops mindset is simply to compel development and operations to work together more often so that the teams share practices, patterns, tools, and processes. To effect real change, however, the implementation must be incorporated into every facet of the software systems, the teams, their processes, and their toolsets.

Three key changes make this process work in a technology organization. First, a company must change its organizational structure to align with what it is delivering to the business. Second, each of the software delivery processes must become continuous. Third, teams must continually improve existing processes so that software is always ready for release.

Change organizational structure

The IT organization needs to align with the business. The business cares about the delivery of new features into stable environments and not about the different reporting structures that are incentivized in completely different ways. Organizations must break down these barriers to create a single cross-functional software-delivery team. Many years ago Amazon took this concept a bit further and required that developers take software all the way to production, referring to this idea as "You build it, you run it." Typically, this kind of organizational change can only come from the top of the organization, because the CEO is the only executive common to both the development and operations teams.

While this change in organizational structure might be the first order of business in a perfect world, this realignment rarely happens. As an alternative, teams can begin working together across organizations in a collegial manner and demonstrating results from this collaboration through reduced cycle times and more-repeatable processes. While this change can be difficult, it is often a more effective approach than making such a significant organizational structure change without the support of measurable internal successes.

Make processes continuous

Teams can apply five steps to achieve a continuous process (examples of processes might be build, deploy, create infrastructure, define databases, insert data, run tests, and run static analyses) and reduce manual error-prone efforts. As a result of this realignment, the development and operations teams will both be working to regularly deliver high-quality features in a stable environment. While these five steps are

generally applied in the order listed below, some teams may choose an alternate order of execution and still achieve the desired results.

- **Document.** Document all the steps required for completing a process (building, deploying, environment provisioning), writing the steps so they can be scripted later.
- **Test.** While some engineers choose to document first, others immediately test, because their first step is documenting the test (especially if they are using an acceptance-test tool).
- **Script.** A developer or operations engineer describes the steps in a text-based automated script.
- **Version.** All scripts and tests are committed to a version-control system, which ensures that every significant change is versioned so that any team member can recreate the software system as it existed at any specific time. This ensures that the knowledge in creating the software system — whether the application code, data, infrastructure, or configuration — is not confined to one individual's memory or recorded in outdated written procedures.
- **Continuous.** Once a process has automated tests and is scripted and versioned, the process can be made continuous. This assumes that the process is headless (i.e., the process runs without human intervention). Once it is headless, it can be configured into a continuous integration (CI) system so that it can be run with every change.

Continually improve

Continuous processes are maintained and improved just as software systems are, so instituting a formal process for continually improving these systems and processes is important. Consider supporting updated versions of the software being deployed, speeding up test execution times, and supporting different platforms. Although the effort in instituting this formal process will not be nearly as large as the initial effort to move to a continuous-delivery model, it is equally important if the delivery team is to be in a position to release software after every good change.

Tools and components

The key tools required for development and operations teams when moving toward continuous delivery include build, CI, database, dependencies, deployment, hardware, infrastructure automation, monitoring, static analysis, and testing. Each of these tools, described below, interfaces with infrastructure components.

Build

Building is the process of converting source code into executable software. The output of a typical build is some kind of binary, such as a WAR file, a DLL, or something similar. Developers define their process for building this software with a tool such as Ant. The build tool also provides the central mechanism for calling out to other parts of the delivery process, including dependencies, database changes, and static analysis as well as creating environments and running tests.

Continuous integration

CI is the process of building software based on every change. Figure 1 illustrates a delivery pipeline as defined in the Jenkins CI server. This delivery pipeline enables an automated delivery of software in stages. Each stage assesses the risk of the release candidate and fails the pipeline when it discovers any error.

Figure 1: Delivery pipeline as defined in Jenkins

Source: Stelligent

Database

Tools that run scripts to create, upgrade, and downgrade databases can be used to ensure the database is always in a known state. A tool like Liquibase eases the management of these processes and provides mechanisms for integrating with build scripts.

Dependencies

Dependency-management tools like Nexus provide repositories for hosting and publishing libraries that are used by the software system under development or other software that might consume the software distributions being published. The purpose of these dependency-management systems is to provide a canonical source for binaries and prevent errors due to corrupt files, incorrect versions, or other reasons.

Deployment

Deployment addresses the steps for running software in an existing environment. Rudimentary steps for a web deployment might consist of stopping the servers, copying a WAR file, applying some configuration, and starting the server again. An example of a tool that supports deployment automation is Capistrano.

Hardware

There are many ways to procure hardware, for example through a virtualization tool, a cloud provider, or access to the physical machine.

Infrastructure automation

Infrastructure automation defines the configuration for an environment. An environment may consist of one node or thousands. With infrastructure automation, this configuration can be applied in a consistent manner across thousands of nodes. A tool like Puppet (see disclosure) supports infrastructure automation.

Disclosure: *Puppet Labs is backed by True, a venture capital firm that is an investor in the parent company of this blog, Giga Omni Media. Om Malik, the founder of Giga Omni Media, is also a venture partner at True.*

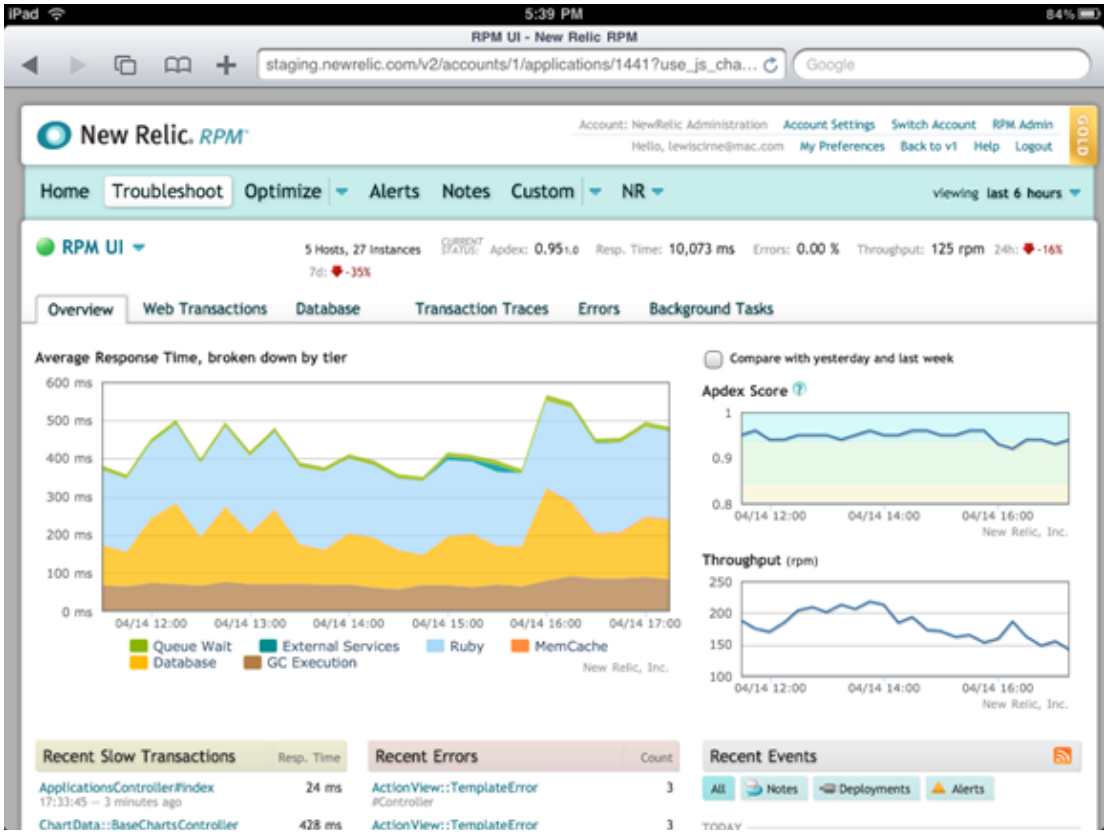
Monitoring

The two types of monitoring tools are application and system. Application monitoring takes a top-down approach, and system monitoring takes a more bottom-up approach. With application-monitoring tools such as New Relic's (Figure 2), teams can monitor the performance of the application from the user's perspective, including page-load times, database-transaction times, trending, and so on. System monitoring focuses on factors such as CPU load, utilized memory, and disk space.

Software-as-a-Service (SaaS) offerings for monitoring tools are breaking down the traditional barriers that may have prevented teams from utilizing them previously. Teams of any size, from hundreds of users to millions, can benefit from using these tools without the up-front investment in time and expense that is typical with traditional enterprise-monitoring tools.

As a result of the constant feedback elicited through a continuous-delivery approach, monitoring becomes more vital to the business as it is able to tweak features, test scenarios, and deliver different features to different users based on application monitors. An application-monitoring tool can sit at the epicenter of an effective cross-functional team that adopts devops and continuous delivery.

Figure 2. New Relic application-monitoring dashboard



Source: New Relic

Static analysis

Static-analysis tools analyze source code to identify potential problems such as coding style, cyclomatic complexity, duplication, test coverage, and dependency analysis. Build scripts can be configured so that builds fail based on predefined thresholds such as high cyclomatic complexity, low code coverage, or too much code duplication. Static-analysis reports can be integrated with CI tools so that team members can obtain trending information from the built-in CI dashboards. Sonar is an example of a static-analysis tool that development teams frequently use.

Testing

Many types of testing tools are available. In continuous delivery, there is a focus on automated testing. Both developers and testers use tools for unit, component, load and performance, function, acceptance, and so on. While testing tools are numerous, examples include JUnit, Selenium, and Cucumber.

Tool catalog

Table 2 lists some of the tools that support processes for both development and operations. As teams become more cross-functional and poly-skilled, team members will share tools that may have traditionally been used by only one of the teams prior to moving toward a devops mindset. The purpose of Table 2 is to document the types of tools more than to identify a specific example tool. The example tool is only provided as an illustration of the type of tool.

Table 2. Tool types and tool examples

Tool type	Example tool
Build	Ant
CI	Jenkins
Database — change management	Liquibase
Dependencies	Nexus
Deployment	Capistrano
Hardware	Amazon Web Services
Infrastructure automation	Puppet
Monitoring — application	New Relic
Monitoring — other	Nagios
Static analysis	Sonar
Testing	Cucumber
Version control system	Git

Source: GigaOM Pro

Tools in Table 2 might interface with some of the infrastructure components in Table 3. Several of these infrastructure components have APIs or other mechanisms that automated tools can use to apply configuration or otherwise affect the behavior of the component.

Table 3. Infrastructure components

Infrastructure component type	Example tool
Application server	Tomcat
Web server	Apache HTTP
RDBMS	MySQL
NoSQL database	MongoDB

Source: GigaOM Pro

The future of devops and continuous delivery

Software systems are continually moving toward self-service models. Increasingly, vendors will provide services that development and operations teams have traditionally provided, resulting in those teams focusing more on developing new features for the business and getting more value and less friction from the tools and infrastructure that support their ability to deliver these features.

Key takeaways

Executives and other organizational decision makers must:

- Mitigate incongruent processes and tools
- Appropriately align organizations to deliver new features while ensuring stable environments

Team members can increase collaboration by:

- Convincing the organization to view software systems more holistically
- Creating cross-functional teams
- Increasing the skill sets of engineers and other team members across disciplines

The benefits of the devops and continuous delivery approaches are:

- Teams learn of problems earlier
- The problems are less complex to fix
- The problems take less time to fix

The business values of applying a devops mindset and continuous-delivery model are:

- Features are delivered more quickly and more often
- Operating environments are more stable
- Software is always ready to release

To implement continuous delivery:

- Change the organizational structure, either all at once (this is seldom possible) or incrementally
- Make processes continuous through documentation, testing, scripting, versioning
- Continually improve so that cycle times are always being reduced

The tools and components of a continuous delivery process include:

- Build
- Continuous integration
- Database
- Dependencies
- Deployment
- Hardware
- Infrastructure automation
- Monitoring
- Static analysis
- Testing

The future of devops and continuous delivery is a progressive move toward increasing self-service offerings from vendors. Developers will be involved in more and more activities that were previously considered operations tasks (such as application monitoring and infrastructure provisioning).

About Paul Duvall

Paul Duvall is the CEO of Stelligent, a company that specializes in continuous delivery in the cloud. Duvall has worked in virtually every role on software projects: developer, project manager, architect, and tester, and he has been a featured speaker at many leading software conferences. He is the principal author of *Continuous Integration: Improving Software Quality and Reducing Risk* (Addison-Wesley, 2007; Jolt Award winner in 2008) and "DevOps in the Cloud LiveLessons" (Addison-Wesley, 2012). Duvall contributed to the UML 2 Toolkit (Wiley, 2003) and authored several article series for IBM developerWorks including Automation for the people and Agile DevOps. He is passionate about devops, continuous delivery, the cloud, and automation, and he blogs at stelligent.com/blog.

About GigaOM Pro

GigaOM Pro gives you insider access to expert industry insights on emerging markets. Focused on delivering highly relevant and timely research to the people who need it most, our analysis, reports, and original research come from the most respected voices in the industry. Whether you're beginning to learn about a new market or are an industry insider, GigaOM Pro addresses the need for relevant, illuminating insights into the industry's most dynamic markets.

Visit us at: pro.gigaom.com