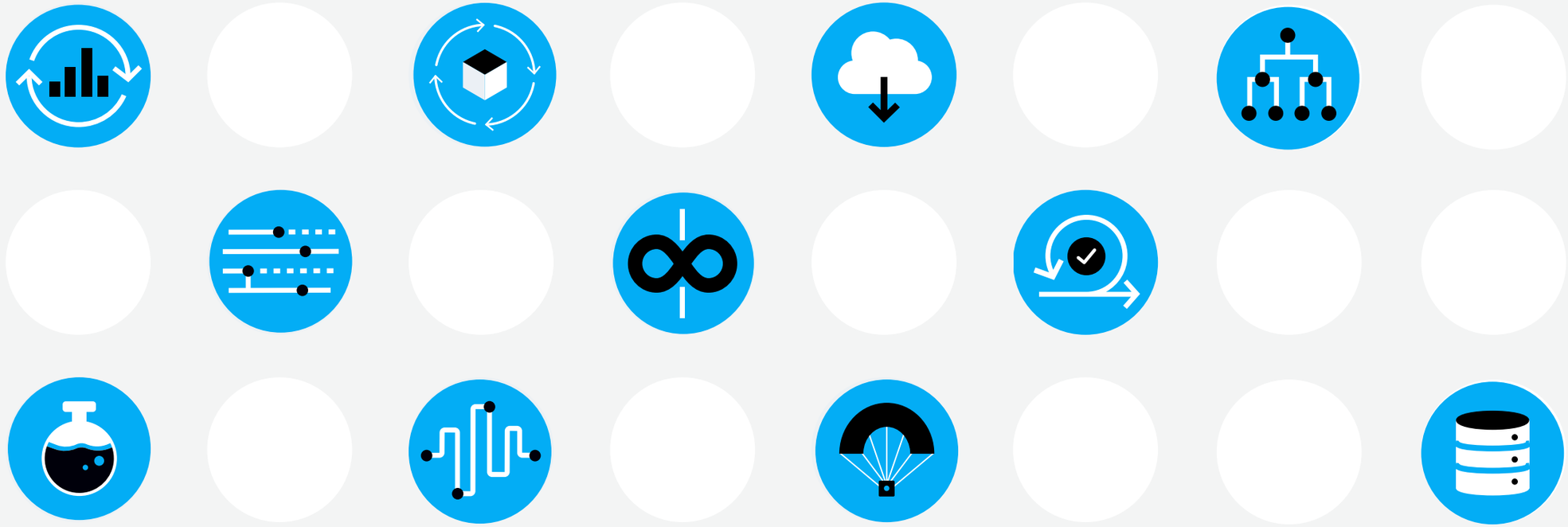


DevOps At Scale: Chapter 7

DevOps Tools: Why You Need Them

PLUTORA

Describing DevOps is not straightforward. It has many meanings. At times, technology professionals pay too much attention to tools and automation and not enough to the culture shifts, team dynamics, and management at the heart of true DevOps.

Further, a team that uses great tools doesn't necessarily use them well. For tools to help, teams need to bring the right mindsets, attitudes, and objectives to their DevOps journeys.

Nonetheless, without tools, it's hard to automate. Without automation, DevOps is more dream than reality. Thus, tooling is a critical part of making the transition. There are more important parts of DevOps than tools, but tools are very much necessary. Choosing the right tools is, therefore, critical to DevOps success.

There are a variety of tools that support a DevOps transformation, whether it's using a tool at the bottom level of daily operation or tying together the many other tools into a complete operation.

How Tools Support DevOps

DevOps causes a shift in perspective toward thinking of software as a means rather than an end.

Software only serves a purpose if it solves real human needs. To determine if a product meets the needs of its users, one simply needs to look at their willingness to pay for it. Therefore, it's important to be able to determine if changes to software lead to the tangible benefit of revenue.

DevOps is at its best when it does just this. While unifying the team, it automates and tracks the flow of features from check-in to delivery. To encompass this comprehensive enhancement of flow, DevOps needs controls and visibility in the entire process.

DevOps tools provide these very hooks into planning, development, testing, deployment, operating, and monitoring of applications. Furthermore, tools with a whole view of the entire process serve the need of orchestrating the many pieces involved. Organizations often lose themselves in the details of individual tools and processes unless they have something at this level.

The following sections offer more detail on tooling for individual parts of the DevOps process, as well as those offering support for higher-level coordination and management.

Agile Planning

In February 2001, a group of software luminaries met at the Snowbird ski resort. They were all using and experimenting with progressive ways of writing software in teams. A result of their collaboration was the Agile Manifesto. This document articulates preferring some emphases over others in the process of team operation. Among other values, agile software development prefers “responding to change over following a plan.”

Iterative and User-Focused

This principle yields an emphasis on iterative planning and iterative execution.

In addition to using an iterative approach, agility implies a focus on people. It emphasizes “individuals and interactions over processes and tools.” As such, agile teams don’t work from requirements documents constructed by an authority. Instead, they use user stories as their guide. User stories describe a person benefiting from a system or feature. Therefore, they put the planners and implementers into the mindset of serving that user.

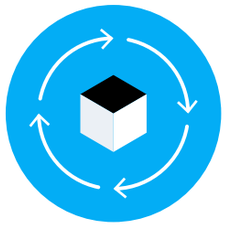
With that in mind, planning in agile means iteratively delivering features that solve a real need for real users as defined in a user story. Tools for supporting this must support the user story definition and assignment to an iteration (or sprint, to borrow the common term used primarily in the popular Scrum form of agile).

Agile Estimation

Also, DevOps tools for agile planning need to take estimation into account. Velocity is a popular metric for assisting with both estimation and tracking progress. It’s easy to misunderstand, though.

Teams often view velocity as a target or a metric for their performance. It’s not useful for those purposes. Rather, it should be viewed as a lagging indicator that puts estimation into a historical context for enhanced utility. Teams estimate the level of effort involved in user stories. They then use their historical velocity (or rate of delivering units of estimation per iteration) to determine which stories they can accomplish in a given period for greater predictability.

Good agile planning tools enable delivery, tracking, and visibility into planning and subsequent execution.



Source Code Repository

Software changes over time. This shouldn't surprise anyone interested in DevOps. With experience in creating software comes the realization that an ability to track history and jump around to different versions of the source code is indispensable. Experienced technologists may remember a time when they weren't familiar with version control. This usually meant making copies of source code directories in various states of change. It was hard to manage and hard to understand. Such manual and ad hoc systems usually failed to handle getting back to known good states, as well as managing history. It's useful to envision the story of creating a version control system from scratch and the concerns one would face in such an endeavor.

Software teams simply need to be able to experiment and either move forward or return without difficulty. They need to know where they've been and make fixes to any point in history along the way. They need to be able to select some changes and not others.

This makes effective version control fundamental.

There are many options available for software version control. Because of its embrace of open source, distributed nature, high performance, and remarkable support for multiple workflows, Git has emerged as the de facto standard.

Modern teams want more than just version control, though. GitHub, Bitbucket, and Azure DevOps are tools that extend Git. They offer access control and collaboration tools for teams working in source code. These features include visualizing source code changes and history, issue tracking, and pull requests (also called merge requests) that enable code review in integrating work on different features and from different team members.

Build Server

Building software sounds simple. It often is. There are many different configurations, though, that a build can take. You can make build output with debugging symbols, or not. Some build parameters control optimizations and platforms. Further, manually building can result in building the wrong version of your code. Put simply, if your build isn't automated, you frequently run into trouble with inconsistent build results. Many organizations have had problems because of deploying something that wasn't built correctly.

Build servers solve this problem.

By using software to build in a known and consistent configuration, builds are repeatable and automatically use the expected version of the code. Thus, teams no longer find themselves guessing

about what they're building and subsequently deploying. Additionally, build servers can run automated tests after every build.

Continuous Integration

Furthermore, having a build server enables continuous integration. A dedicated build server can trigger builds for every commit that gets pushed to the canonical repository. This means that every time there's an update to the source code, it's integrated into a build that's ready for automated testing, manual testing, acceptance testing, and possibly even immediate deployment to production. Teams that work in isolation tend to have problems when they try to integrate. Continuous integration means integrating work early and often so issues get addressed as quickly and effectively as possible.

Deployment Server

Deployments of software can be cumbersome activities requiring multiple people and many hours.

However, deployments can also be automatic. Different teams have different needs, and the conditions for deployment vary. For some teams, passing automatic tests in build and following automatic release to test environments mean a build is ready to be released to production. For others, there are manual gates that need to be cleared before deployment can proceed. In any case, deployments need to be easy, repeatable, and without undue burden to team members. The deployment server exists for this purpose.

Deployment servers automate the tasks associated with deployment so that deployments can be triggered based on events, like passing tests or the click of a button.

The deployment server can be a stand-alone software product running on a separate server or bundled into a unified product with the build server. There are options for both build and deployment servers that run in the cloud or on premise.

Continuous Deployment

As an added benefit, deployment servers enable continuous deployment. As mentioned before, it's desirable for some teams to deploy their software automatically as soon as it's complete. For such teams, continuous deployment is a reality enabled by the deployment server. Completed testing on completed builds triggers automatic deployment. This puts value in the hands of users more quickly than any other method of deployment.

Monitoring and Logging

Modern teams realize that extensive automation is a double-edged sword. Though automating has huge and obvious benefits and is necessary for DevOps, it comes with a downside. When build, deployment, operation, and management of infrastructure and applications aren't done manually, it can be challenging to know what is happening and what has happened. With added complexity in large-scale software projects, visibility is both difficult and critical.

For these reasons, application logging that tells the story of what has happened and what is happening is table stakes. It's necessary, but not sufficient on its own, though.

Merely logging well in applications doesn't solve the need for visibility across redundant instances of running applications. The many applications and services running in modern environments exacerbate this problem. Logs need to be centrally visible and aggregated in ways such that team members and leadership can make sense of overall system health and status.

Log and data aggregation services exist to fill these needs. These services also often provide monitoring and alerts for conditions related to message types, frequency, and infrastructure characteristics.

Good logging enables visibility by giving a picture of system activity over time. Visibility enables identification of key events and metrics. Knowing what to watch enables automation of monitoring. Good logging and good monitoring go hand in hand. It's for this reason that log aggregation tools also often support alerting based on monitors.

Virtual Infrastructure

The challenge of provisioning capacity to support workloads is great. Too little capacity is a problem for obvious reasons, but too much is also wasteful and cost-ineffective. The virtualization of infrastructure provides answers for this by creating flexibility for organizations in handling their workloads. It does this by using physical hardware to provide services to multiple instances of operating systems, rather than dedicating the hardware to only one.

By using virtual machines, teams also provision faster and can get up and running on projects in days rather than months. Such teams also have greater options for moving virtual machines between hosts to achieve greater fault tolerance. This means greater availability.

The Cloud Revolution

Virtualization opened the door to what is perhaps the largest and most impactful change in large-scale computing in a generation. Cloud computing is enabled by the utility of virtual machines. Major players like Amazon, Microsoft, Google, and IBM provide mind-boggling amounts of computing capacity and share it with the world via many strategies for running workloads in virtualized environments.

Cloud computing gives options to organizations to be able to scale in ways that were accessible only to the largest businesses before. It lowers the barrier to starting on projects and increases both speed and agility.

Containers

Beyond virtual machines, containers are the next step in virtualization. Containers move higher up the stack and virtualize not only the hardware but the operating system kernel as well. This yields a virtualization vector with the ability to create smaller and more concentrated payloads to deploy on top of infrastructure prepared for them. Docker, a leading container platform, doesn't make containers possible, but it makes working with containers straightforward. Docker is a set of tools that make it easy to build and run containerized processes that come with all the dependencies they need. Working with containers enables the true spirit of DevOps in that it's the most complete way yet devised for developers to package everything an application needs into an image that just needs to execute. Rather than documenting how an operations team should run an application, teams can now write a Dockerfile that scripts out the creation of the runtime environment. This is powerful in progressing DevOps.

Container Orchestration

With containerized workloads running processes with everything they need to thrive, there's still a need for coordination and command of those containers to make for a cohesive whole system. This is the realm of container orchestration. The leading tool for container orchestration is Kubernetes. It's a mature platform that emphasizes that teams specify the desired state for their containerized infrastructure, with the right amount of redundancy and ways of verifying health, and Kubernetes turns that desire into reality. After achieving this desired state, it then endeavors to keep it there.

Orchestration engines are complete platforms (at least, in the case of Kubernetes) that can run an entire operation. They do this while providing service abstractions, load balancing, fault tolerance, monitoring, and management capabilities.

Serverless

Yet another alternative model of virtual infrastructure is that of serverless applications.

The name is misleading. The software does run on servers. Serverless is named as it is, though, because teams deploying to serverless infrastructure don't have to think about servers.

They simply provide functionality for a cloud provider to run, and the infrastructure handles what it needs. For many workloads, it's an option worthy of consideration.

Test Automation

Automating tests is not always straightforward. It can be a messy business, and it takes time to learn how to do it effectively.

Despite the difficulty and investment required in getting up to speed on automated testing, it's well worth it. Teams that know their complete suite of automatically executed tests have advantages over those without. Such teams know their tests will notify them of problems. They feel comfortable making changes. They proceed with confidence.

Why Automate Tests?

Teams without such automation are often left holding their collective breath as they deploy. Work is a series of stressful days, waiting anxiously for the next fire to put out. In such a stressful environment, deployment is painful. When deploying is painful, increasing deployment frequency isn't feasible.

In addition to the confidence that comes from well-designed and well-written tests, such a test suite is a prerequisite for continuous deployment. If the automated tests can't be trusted to inform the team of regressions, deploying continuously isn't practically possible. Good test automation is therefore crucial for teams practicing DevOps.

Barriers to Effective Test Automation

There are many reasons testing is hard. Developers have often not practiced the skill of writing good unit tests. Tests often do too much or test things that don't need to be tested. Frequently, tests require too much setup and are therefore too slow.

DevOps teams need tests that execute quickly and give rapid and reliable feedback on the correctness of builds.

Tests that test the entire stack from end to end are also important. They need to be fewer in number to support the idea of rapid feedback because of their relative slowness. Still, they complete the picture of the needs for automated testing. Such tests, or at least a subset of them, can also be used to verify that production is operating as expected.

Test Automation Tools

The primary testing tools DevOps teams need to know are unit test frameworks, browser-driver automation tools, and behavior-driven development (BDD) frameworks.

Developers use unit test frameworks most frequently to write small tests with limited scope that execute extremely quickly. This is primarily to drive fast feedback during the development process. Such tests are adept at catching regressions too. These tools are beneficial for other types of tests as well and have useful features for selecting which tests run in which contexts. Unit test frameworks exist for close to every language known.
Browser automation tools are useful for testing

as a user would, from end to end in the system via the user interface. Such tests are usually the slowest of all automated tests. However, they have the greatest scope and are the most complete in the problems they can identify. They are also more susceptible to false positives and are relatively hard to maintain. For these reasons, they're necessary but should be a minority of the automated tests written by a team. Selenium is the most popular browser automation tool.

Behavior-driven development tools qualify as testing tools. The primary aim of BDD tools is collaboration on understanding the needs of users rather than testing. Although they aren't strictly testing tools, they're sometimes used as though they were. Doing this can result in very readable tests and very useful test results reports. For these reasons, BDD tools can stand in the stead of unit test frameworks in addition to serving purposes beyond testing.

DevOps Security

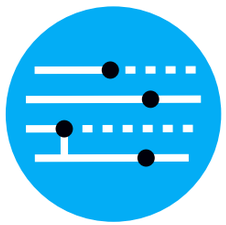
DevOps causes a shift in perspective concerning organizational security. The idea of one team that encompasses both development and operations and a blending of those roles might imply a few things about security specialists. That security specialists are left out of this new clique, yielding a fragmented organization, is a possibility. That team members take responsibility for security is another.

DevSecOps is the term that applies the DevOps idea of whole team responsibility to security. In the same way that DevOps blends responsibility for development and operations into the whole team, DevSecOps includes security in DevOps. This means applying a security mindset to DevOps and applying a DevOps mindset to security. It requires team members to understand security concerns and take them seriously.

Fast-paced DevOps environments present new security challenges. Cloud computing infrastructure provides different security facilities than traditional on-premise deployments. Clouds are not necessarily less secure than on-premise devices, but the security concerns are different. Containers present additional problems. Docker, for example, has had vulnerabilities exposed. They tend to resolve them quickly, but these many moving pieces present a diverse set of attack vectors. Defaults in building container images often yield running with root access. Secure containers result from careful attention from thoughtful team members. This requires either constant vigilance or tools to resist vulnerability. Further, the proliferation of automated deployments and dynamically created infrastructure creates new requirements for secrets management that can be accessed in multiple instances and still secured.

Dealing with these security challenges requires sophisticated governance. Cloud platforms provide secret management, as does Kubernetes. Platforms provide facilities for policies to restrict access, including private, segmentable networks. Kubernetes, for instance, is configurable such that it will only run containers executing as specific non-root users.

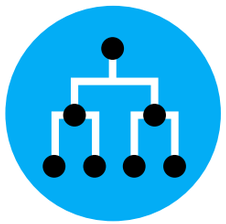
Penetration testing and security audits are also essential security tools for DevOps.



Pipeline Orchestration

Only the smallest and simplest of organizations have a single pipeline to worry about. In large-scale, modern businesses, multiple teams working on multiple subdomains compose the larger operation. Ideally, boundaries are constructed such that deployments can happen independently and teams move forward delivering with dependencies on when other teams deploy. This isn't always a reality. Even in cases where it is, coordination of multiple builds and deployments makes sense to track features from concept to cash.

Most of the tools discussed in this paper so far are concerned with a piece of the overall flow of value through a system. Pipeline orchestration enables the composition of these disparate pipelines into a more cohesive whole.



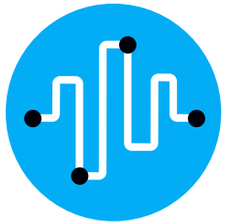
Management

Related to pipeline orchestration, organizational leadership needs visibility into quality and understanding of how every system state came to be. Governance is the term used to relay the importance of this knowledge. DevOps tools need to surface the information necessary to prove compliance to standards for auditing.

Changing the culture and automating pipelines are only steps in a true DevOps transformation. Visibility into pieces of what's happening in the overall operation is necessary but leaves a lot to be desired when it's fragmentary.

With unified visibility, leadership better understands the flow of value through the stages of delivery. This enhances decision-making at the top level, which feeds back into the implementation. Friction points requiring attention are identified and can then be addressed.

DevOps tools offer control levers and valuable information at all levels of the organization.



Value Stream Mapping

When DevOps takes hold, visibility into pieces of an overall operation is straightforward. Visibility into the overall health and, more importantly, the value of the whole is quite another matter. Disparate log streams and pipeline statuses tell a story, but it isn't the whole story of the value delivered by the business and the technology teams serving it.

The original mapping of the value creation process toward realizing product features is typically done on whiteboards. Features are born in a creative collaboration of minds designing how pieces of systems can interact to create something useful.

Agile planning brings this utility into user stories that can be implemented by teams. DevOps yields quick delivery of these stories. Management of this value stream helps the organization respond to customer needs and deliver features quickly.

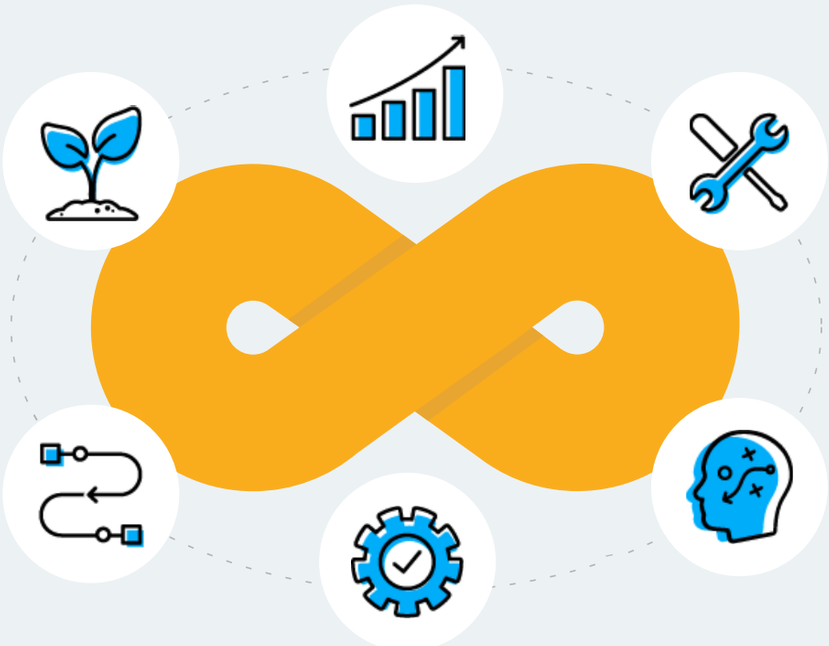
Ultimately, teams want to be focused on creating value. In the same way that user stories shift focus from requirements to real user needs, an effectively mapped value stream facilitates the delivery of real value, rather than potentially meaningless features.

Plutora offers a value management platform that enables the enlistment of a vast array of DevOps tools into a converged toolchain. The converged toolchain offers a unified and cohesive view and control plane for understanding customer value and the process of delivering. This means not only DevOps teams, but broader DevOps organizations.

Tying It All Together

Doing DevOps right means uniting teams in a common purpose and aligning that purpose with value to end users. This and speedy agility with rapid feedback are enabled by automation. DevOps tools form the crucial foundation of an automation platform. While these tools make operations proceed without human intervention, they also provide views into the operation for human intelligence to make sense of what's happening automatically. Finally, at the top-level, a converged toolchain like the Plutora value stream management platform completes the link between needs and implementation.

DevOps isn't about tools, but tools make complete DevOps possible. Tools enhance cultural improvement and are indispensable.



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