DevOps Toolchain

UpGuard

Table of Contents

Introduction

DevOps may need little introduction these days, but many are still at a loss to explain precisely what the movement entails. Some emphasize the portmanteau of the two terms, stating that the heart of DevOps is the collaboration between developers and operations staff. Others choose to focus on the tools and the problems they solve, singing the praises of DevOps for fixing their respective infrastructure woes. Tools-- though crucial enablers of the movement-- only form part of the equation. DevOps encompasses cultural innovation, a breaking down of walls and silos between software development, operations, and QA/ testing-- in addition to the tools and methodologies enabling this transformation.

Ultimately, the definition of DevOps varies per organization. Since its meaning depends heavily on the audience and context in question, general discussions around the true definition of DevOps are for the most part inconsequential. If you is specifically concerned about what is/not DeVops, check out our ebook "DevOps for Cynics" and our blog post "Defining DevOps." If you want to know about tools that can make your life easier, what makes each one unique, and how they fit together, read on.

Hybridization of Roles

A discussion regarding tools and DevOps should therefore begin by considering the individuals who will be utilizing the tools. The rise of so-called "polyglot programmers" and systems administrators with coding proficiency reflects a general trend in IT towards despecialization. Developers these days are adept in a number of languages and approaches, applying each accordingly based on the problem at hand. Similarly, most systems administrators possess competent programming abilities for traversing the stack-- on top of the requisite skills for managing IT operations. The industry has been quick in attaching new labels to these emerging hybrid roles: DevOps Engineer and DevOps Specialist being the most common. Notwithstanding, the key takeaway is that no single IT skill is more important or valuable than another; subsequently, many different tools are required to do the job effectively. So as DevOps is comprised of a group of concepts clustered around the premise of continuous software delivery, these concepts in turn encompass a range of associated tools for fulfilling particular functions.

All in all, these complementary tools fill out the DevOps toolchain, unifying the best elements from development and operations. Keep in mind that both tools and cultural innovation are required for DevOps; adopting a popular solution on its own as a magic on-ramp to DevOps is a quick path to disillusioned, as there are no "DevOps" tools, per se. The combination of cultural changes, information de-siloing, and tooling implemented along way is what enables an organization to recognize ROI from DevOps. In a sentence, it's not just about the tools, but the people as well.

Agile Roots

At first glance, DevOps may seem like an evolution or extension of Agile and Lean methodologies that have gained prominence in the last decade. While this is certainly true in many respects, an important distinction lies in scope: while Agile deals primarily on the development side of affairs, DevOps stresses a unified approach that covers the entire scope of software delivery. So as Agile stresses cross-functional collaboration to aid incremental, continuous development of quality software, DevOps expands this ideal to include development, IT operations, and QA/Testing teams as interdependent cogs of the same software delivery mechanism. Indeed, many of the Agile tools and methodologies find their way into the DevOps toolchain and workflow, as the two promote the same style of collaboration. Furthermore, as software development ultimately depends on operations for deployment, a closer integration of the two groups will naturally boost quality and efficiency.

Infrastructure-as-Code

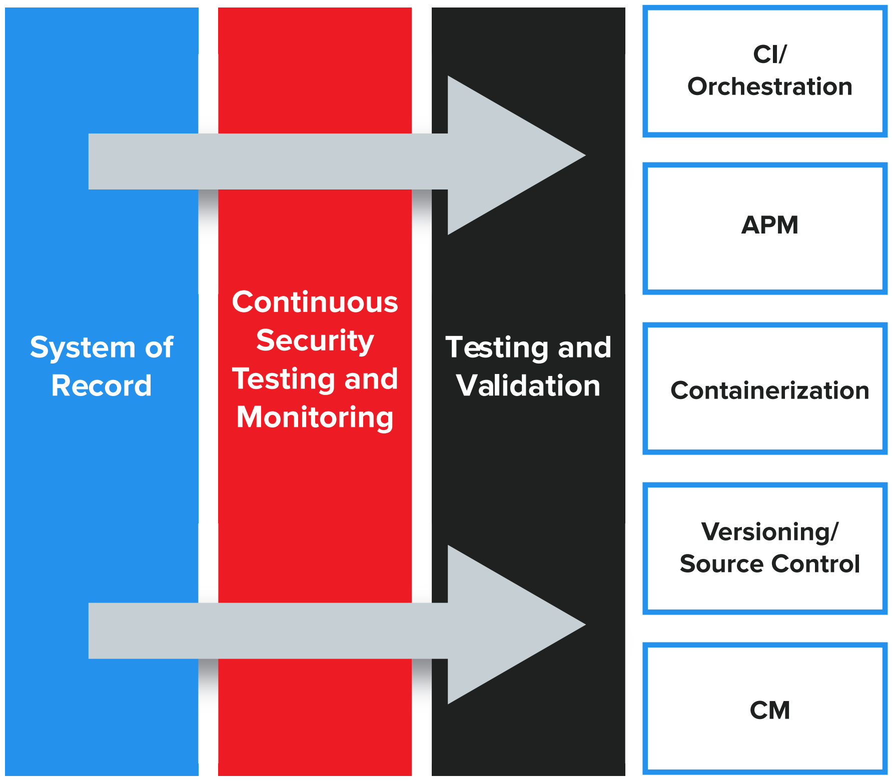
With Agile software teams becoming commonplace, IT operations needs a way to keep up infrastructure with this rapid pace of development. Furthermore, as virtualized environments and cloud infrastructures become more commonplace, the operations side needs a more dynamic, flexible approach to managing systems. Borrowing from their software development counterparts, systems administrators can now manage their infrastructures as code-- automating and tracking configurations like source code. This enables the ability for version control, rolling back of changes, as well as integrated testing and deployment to production of necessary software and server components. This unification of all sides of the software delivery puzzle is also referred to as "programmable infrastructure," and is central to practicing DevOps.

The DevOps Toolchain

Many of the tools that are essential to DevOps practitioners are also familiar Agile tools. Similarly, as configuration management (CM) is a central premise to DevOps, many of the tools mentioned below are popular CM and automation tools. Again, they fulfill the specific area of DevOps that they are good at.

Versioning and Source Control

Tracking code level changes is a common and necessary activity of today's software developers. Doing so enables concurrent development, merging, and rollback capabilities for applications/ software. Source Control Management (SCM) tools are popular options for keeping track of software code; many DevOps practitioners also track versions of their systems configuration with these tools, essentially managing their infrastructure "as code." For example, it's a common practice for systems administrators to store and manage their Puppet Manifests or Chef Cookbooks in GitHub.



Version/Source Control Tools: Git/ GitHub, Mercurial/BitBucket, Subversion

Continuous Integration and Orchestration

Continuous integration (CI) and orchestration tools enable the integration of development code into the overall software product frequently and early in order to mitigate potential conflicts down the line. Typically, these tools are employed to automate software builds and testing, and are crucial for applying quality control on a continual basis (as opposed to after the software has been developed and released). These tools can also be used to track and manage changes for CM-- for example, Chef Cookbooks can also be stored in version control with Github. The appropriate CI tool can then be used to test cookbooks for bugs and errors, and set up to automatically to do so every time infrastructure changes are committed and merged.

Continuous Integration/Orchestration Tools: Travis, TeamCity, CircleCI, Drone.io

Testing and Validation

Tools and frameworks for testing and validation are important for ensuring quality at all phases of development. In many cases, unique solutions are applied to a specific aspect of testing-- for example, one tool may be used for unit testing while another is used for integration testing. Solutions like UpGuard provide crucial functionality for testing/validating environments, and are indispensable for troubleshooting and debugging software applications. The platform allows one to anticipate changes and pre-validate every environment before deployment; by generating tests directly from development and running them against the target environment, DevOps practitioners can confidently release quality, error-free software. Combined, these testing and validation solutions provide a consistent mechanism and format for testing application features and behavior on both a micro and macro-level.

Testing/Validation Tools: Cucumber, Rspec, Selenium, Capybara, Coverity, UpGuard

Configuration Management (CM)

CM tools allow one to define the desired state of a system and/or environment in regards to configuration files, software installed, users, groups and many other resource types. They also provide functionality to automatically push changes onto specific machines, also known as automation and orchestration. Tools like UpGuard can provide initial discovery and visibility into an infrastructure, create "golden images" for automation tools like Puppet and Chef, and validate that results are in line with expectations, post-automation.

CM Tools: Puppet, Chef, Ansible, SaltStack, UpGuard

Containerization

Containerization essentially allows one to package up or "containerize" an application in its own environment, making software easier to deploy and move. Such tools may often take radically different approaches to meet this end, but achieve similar results: developers are free from infrastructure-related concerns, and operations staff similarly need not worry about the applications being run inside the containers-- they just work. As a lightweight alternative to virtual machines, containerization tools have gained immense popularity as of late for testing software applications.

For example, Vagrant is a popular tool for automatically creating and configuring lightweight, reproducible, and portable development environments. Vagrant can coordinate with a configuration management (CM) solution to continue the process of installation where the operating system's installer finishes, otherwise known as provisioning. Technologies such as Docker accomplish similar functionality as Vagrant, but uses a different approach to containerization. Tools like UpGuard can natively output to Vagrant and Docker for easy provisioning of containers.

Containerization Tools: Docker, Vagrant, Rocker

Application Performance Management (APM)

In contrast to testing and validation on the code level, APM solutions allow one to test and troubleshoot a software application's performance under various conditions. For example, SaaS applications are commonly tested and monitored with APM tools to ensure high availability, low response time, and quality of service. By gauging how efficiently an application is utilizing system resources, developers can more easily identify and resolve performance bottlenecks-- the net result being superior service delivery of one's software applications.

APM Tools: New Relic, Ruxit, AppDynamics, Stackify

Continuous Security Testing and Monitoring

The importance of continually testing and monitoring one's infrastructure for vulnerabilities, configuration changes, and drift cannot be stressed enough. Developers may be savvy enough to avoid code-level security issues in an application, but ultimately the software is as vulnerable as its underlying systems and infrastructure. Detecting and remediating security flaws at all levels of the application and technology stack is therefore crucial to bolstering a software application against security threats and potential compromise. Implemented as part of the continuous integration process in ongoing software iterations, continuous security testing and monitoring help to maintain a strong security posture throughout all phases of development. UpGuard provides comprehensive vulnerability scanning and monitoring to ensure that one's infrastructure and systems are optimally poised against an evolving threat landscape.

Continuous Security Testing/Monitoring Tools: UpGuard

System of Record

The cross-functional collaboration and information "un-siloing" promoted by DevOps is deceptively straightforward in theory but can be quite challenging in practice. Much of this is due to the sheer volume of disparate moving parts required to make the DevOps machinery operate: developers checking in/out and merging application code, operations staff bringing up/down and patching systems, and any number of continuous integration activities. These factors-- along with the natural tendency for system configurations to drift over time-- make a single system of record for DevOps crucial for a myriad of critical functions. This mechanism ensures the validity and consistency of environment-wide configuration information, and provides a common datasource for CM activities, automation, and continuous security monitoring, among others.

Consider activities instrumental to CM and automation like baselining or "golden image" creation: to attain a specified desired state, one must have a correct reference model to work from. This can be for any number of purposes: to harden one's infrastructure security posture, replicate environments for testing, or to confidently automate provisioning; referencing a common datasource for systems information is necessary in these and many other scenarios. Having a single system of record enables proper visibility and validation for consistent delivery of quality software and services.

As mentioned previously, UpGuard performs a critical role in capturing desired system and environment states. In this capacity, it serves as the single source of record for CM, testing, and other constituent components of the DevOps toolchain. UpGuard closes the feedback loop to ensure that developers begin from the same state as production, post-automation states are in line with expectations, and infrastructures are monitored against an up-to-date, secure "golden image."

Tools for Establishing a System of Record : UpGuard

Project Management	Infrastructure changes are tracked as tickets in UpGuard and sent to your project management tool of choice.	Examples: Jira, Asana, Pivotal
Requirements Gathering	The requirements of current applications are available in UpGuard's system state documentation.	Examples: Word, Wikis, Spreadsheets
Versioning	Artifacts Code and UpGuard policies are versioned and checked in to be used in the build and deployment process.	Examples: Git, SVN
Continuous Integration	Continuous integration and deployment tools use UpGuard policies to validate environments before and after deployment.	Examples: Jenkins, Team City, Travis, CircleCI, Drone.io
Configuration Management	UpGuard generates manifests forconfiguration management tools like Puppet, Chef, Powershell DSC, Ansible, Salt, and more.	Examples: Puppet, Chef, Ansible
Monitoring	Configuration state is continuously checked for deviations from baseline, much like you would with performance monitoring.	Examples: Shell Scripts
Discovery	Complete configuration state is documented and accessible for anomaly analysis and troubleshooting.	Examples: CMDBs
New Baseline	After confirming the system state's health, UpGuard documents the new baseline for development.	Examples: None
Standardize	UpGuard provides the feedback mechanism from the end of one development cycle to the beginning of the next.	Examples: None

Conclusion

A typical DevOps toolchain might consist of the following: UpGuard to discover and track what you have and to determine what your environment should look like. The platform can then output to a tool like Chef, Puppet, or Ansible for provisioning and automation--or directly to Docker for creating containers or Vagrant for creating development and test environments. Once systems changes and applications have been deployed to production, UpGuard can validate that the changes have indeed been rolled out successfully, as well as provide further validation that any deployed applications and systems are free of vulnerabilities through comprehensive vulnerability scanning.

In the context of DevOps, the whole is truly greater than the sum of its parts. One must be equipped with the proper range of tools to address the unique, ongoing challenges of continuous integration and software delivery, and no one tool can do the job alone. DevOps is about delivering higher quality applications quicker and with less errors; this is accomplished by breaking down silos between development and operations and creating a smoother path towards software delivery. DevOps and its underlying concepts provide undisputed benefits to any forward-thinking organization, and the DevOps toolchain provides mechanisms to realize these benefits. ■

Project Management

Requirements Gathering

Versioning

Continuous Integration

Configuration Management

APM

Log Monitoring

Discovery

New Baseline

Standardize

Appendix

http://www.agilealliance.org/blog/2012/08/02/the-agile-root-of-devops

http://www.networkcomputing.com/networking/achieving-infrastructure-as-code/a/d-id/1318498

http://www.webopedia.com/TERM/C/containerization.html

http://www.centurylinklabs.com/what-is-docker-and-when-to-use-it/

http://www.drdobbs.com/architecture-and-design/containers-for-development/240168801

http://devops.com/blogs/automated-security-testing-continuous-delivery-pipeline/

http://devops.com/blogs/devops-critical-systems-record-systems-engagement/