DEVOPS FOR THE ENTERPRISE: ALIGNING AND STRUCTURING PROJECTS, PART II

Patrick McClory July 21, 2016 DevOps 8,325 Views

In my last blog, I started to talk through best practices when it comes to enabling the DevOps workflow for enterprises. This can also be thought of as how to organize work around the teams in such a way that they are manageable and sufficiently discreet enough to enable end-to-end ownership. One blog was not enough for this complicated topic, so today I want to continue this conversation and focus on four additional components in the suite of important DevOps tools.

Automation Orchestration System

Building automation around your application management lifecycle starts with building your automation toolset around your applications. Many existing solutions offer the ability to build for multiple platforms, languages, and types of applications. This level of flexibility is key for large-scale codebases that might incorporate multiple technologies within the architecture. There are a number of open source and commercial products that are designed to manage the process of building and packaging applications. A specific variety of these, which are positioned as 'Continuous Integration' tools, have built-in features to

monitor a source control repository and trigger events from check-ins or other actions being performed by the team. In looking for a tool that will facilitate the rest of the automation discussed, it is important to ensure that the automation tool is extensible and can execute tasks written in the team's language of choice. Tools like Chef, Puppet and Ansible (configuration management) and Terraform (infrastructure as code) have, as a core tenant of their design an extensibility and plugin model that exists to allow for this future growth and customization without forcing you to take full ownership of the toolset overall. This flexibility is critical to leveraging open source and other toolsets in ways that both alleviate you of 'undifferentiated heavy lifting,' but also allow you the ability to save time and achieve your overall goals with more consistency and quality.

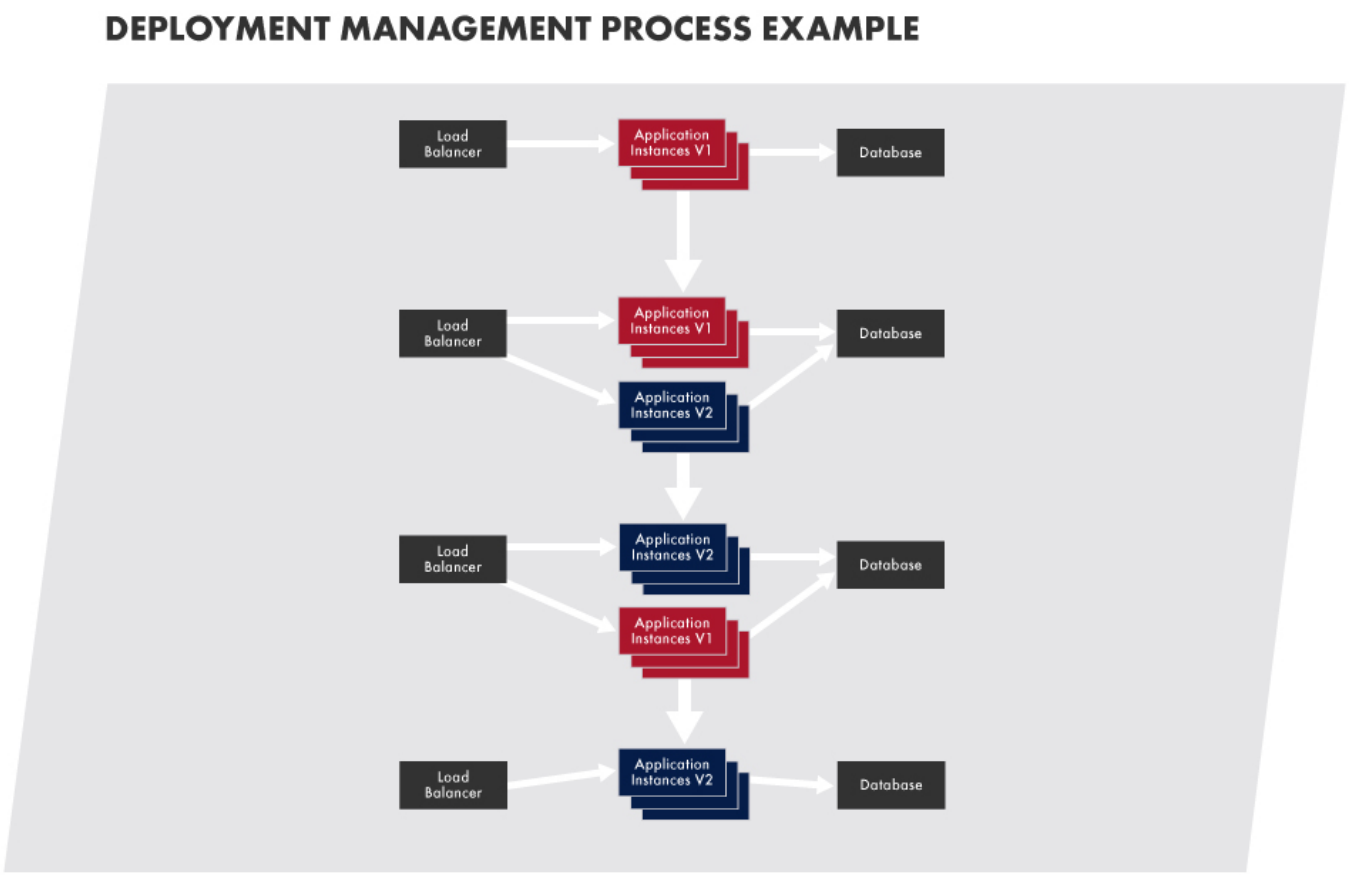
Functional Testing Tools

While there is definite value to human interaction with a system to ensure it is operating properly, there is certainly a high percentage of tasks that can be automated with a tool or suite of tools specifically designed for the type of application (web, desktop, service) being tested. What's most important is that your functional testing tool integrates with the automation orchestration system that's been implemented as the corporate standard. Ideally, it would do so with a reasonably similar set of technologies (language, database type, etc.) that individual teams would be familiar with. Along with the software engineers, software engineers in test (or QA engineers) should also be able to leverage the same source control management tools that the software development team uses in order to locate all of the project's assets in one place. Executing functional tests as a part of the build, package, and deploy process could then rely on one set of versioned tests. A simple and well-managed feedback cycle allows the team to move quickly and increase project velocity and cross-functional collaboration throughout the application lifecycle.

Deployment Management Process

Managing deployments for integration, test, and staging environments should leverage the same framework and be integrated tightly into the overall development lifecycle. While deploying a specific individual or set of applications to a target environment is important, handling the deployment rollback process is just as critical. While refining existing strategies and techniques is a decent approach to this problem, resource management in the cloud-focused infrastructure paradigm leads to an inflection point of innovation which leverages the programmability of cloud resources along with the economics of pay-for-use resource allocation. Resource allocation in the cloud is no longer a function of your physical data center footprint or your capacity to buy physical hardware fast enough. Concepts such as managing database backups via volume snapshots rather than database-specific extracts and file artifacts leads to the

possibility to version not only your application code and your server configuration, but also the database (and its data). With a strong configuration management toolset and a commodity perspective on compute resources, spinning up new pools of servers for a new version of an application and using auto-scaling policies to increase and decrease capacity on demand allows for true A-B production rollout configurations where failing back the application tier requires that you simply reinstate the old pool of servers.



Defining a Feedback Loop

The final component in the suite of DevOps tools is to implement a ticketing or issue management system as a means of providing a feedback loop – for feature requests, bugs, environmental, performance, and technical-level items to be addressed by the team overall. Issue management is not anything new. The key differentiator in this workflow is that environment ownership is played out by the team's collaborative effort to resolve issues across multiple disciplines. The mark of a well-run DevOps team is the collaborative effort to resolve problems as opposed to a continual handoff between disparate resources.

While an enterprise system might already contain an issue tracking system, commonly they are over-engineered. Creating a simple, easily understood, and open system is critical. Simplicity in managing feedback allows for greater transparency and better management of requirements, issue definition, and

enhancement requests. True to the DevOps form, this too should fall under a process of 'Automate and Simplify', wherein as much as possible is automated and the rest is simple as possible. Automating this process could include features within an issue tracking system including auto-populating fields such as reporting user, system affected, and even setting an issue template based on the type of issue reported for a specific application or system.