



DevOps and Its Practices

Liming Zhu, Data61 | CSIRO

Len Bass, Professional Education Consortium

George Champlin-Scharff, IBM



NOT VERY FAR away and not very long ago, an engineering staff member would hand out CDs containing the nightly build to each developer every workday morning. Each developer would load that disk onto his or her computer and, 40 minutes later, begin his or her daily activities. At night, the reverse process would happen. Each developer spent about two hours daily in operational overhead dealing with builds.

Today, the situation is much different. Developers initiate builds any time during the day, and the results are quickly available. Tests and deployments are automatic. Furthermore, deployment isn't the only difference between then and now. Postdeployment processes detect and resolve errors and encourage developers to write code that's more error resistant.

These revolutionary changes are all a portion of what's meant by DevOps. "DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality."¹ As with all technological revolutions, DevOps practices impact processes, products, associated technologies, organizational structures, and business practices and opportunities. In addition, adoption of DevOps practices isn't always smooth. The revolutionary nature of the changes introduces organizational and business stresses.

Many of the organizational stresses are standard for new technologies. For example, much of the writing about DevOps deals with cultural issues. Cultural issues in technology adoption have long been a discussion topic.² We've chosen to focus on aspects of product changes and how those changes affect the way developers think about their products.

The Effects of DevOps

DevOps practices affect developers throughout the software development life cycle:

- Developers must verify a system's provenance upon initialization. This verification determines whether the system has gone through the requisite gates with the requisite approvals.
- One practice is continuous deployment. A developer can place code into production without coordinating with members of other development teams. This affects the design choices and the overarching architectural style.
- Systems move through various environments on their way to production. This affects the use and management of configuration parameters.
- Systems are monitored after deployment, and changes might be rolled back. This affects the architectural style, the information that's exposed, and how that information is exposed.

In addition, DevOps practices rely heavily on tools of various kinds, including tools for container management, continuous integration, orchestration, monitoring, deployment, and testing. Increasingly, software engineers are the ones who maintain and configure such tools. In some organizations, such as Netflix, Google, and Amazon, they also develop those tools, whereas most organizations use existing tools.

Microservices

The microservices architectural style³ is fast becoming the standard for building continuously deployed systems. This style is a restriction of a service-oriented architecture. The

restrictions are that each service is small (hence the "micro") and that all service developers understand they're working on the same overall system.

The size restriction means that large systems comprise many smaller systems. With microservices, a single development team develops and maintains responsibility for a microservice, and coordination among the teams is minimized. This gives a system composed of microservices some characteristics of a system of systems.⁴ In particular, the question exists of determining the overall system's health and attributing changes in that health to changes in individual services. Furthermore, there's the challenge of encouraging individual developers to ensure that their services are good citizens within the overall system in terms of reliability and reporting performance. In this theme issue, "Chaos Engineering," by Ali Basiri and his colleagues, addresses these challenges and places them in a broader context.

Migrating a system to microservices involves rearchitecting it. When the system is currently being used in production, changes should be incremental. Having an example of a sequence of changes provides some guidance on how to proceed. Armin Balalaie and his colleagues offer this in "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture." They describe how they used open source tools and incremental changes to migrate a system providing services for mobile developers to a microservices architecture. They've abstracted a collection of migration patterns that provide guidance independently of any particular system.

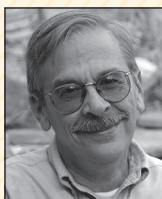
Adopting DevOps

As we said, migrating an organization to a microservices architecture

ABOUT THE AUTHORS



LIMING ZHU is the research director of Data61's Software and Computational Systems Research Program. His research interests include software architecture, dependable systems, and data analytics infrastructure. Zhu received a PHD in software engineering from the University of New South Wales. Contact him at liming.zhu@data61.csiro.au.



LEN BASS is a member of the Professional Education Consortium. His research interests are software architecture, DevOps, and software engineering education. Bass received a PhD in computer science from Purdue University. Contact him at lbass@professionaleducationconsortium.com.



GEORGE CHAMPLIN-SCHARFF is the agile practice lead at IBM Watson Engineering. He has 20 years' experience at IBM helping teams grow engineering skills, embrace automation, and adopt continuous improvement. Contact him at georgecs@us.ibm.com.

with the associated introduction of DevOps practices involves both technical and cultural transformations. In "DevOps: Making It Easy to Do the Right Thing," Matt Callanan and Alexandra Spillane focus on both the technical and cultural issues associated with introducing a continuous-delivery pipeline.

DevOps is in its infancy in terms of its adoption curve. Two of the three articles in this theme issue deal with adoption issues. Some future DevOps issues are foreseeable. One clear question is, "Which practices are best for which kinds of systems in which kinds of organizations?" DevOps practices grew up in organizations providing services over the Internet with, essentially, one very complex and large system. Although Amazon, Netflix, and Google have evolved their systems since introducing them, the system elements

are basically extensions of the same family. This isn't true for the systems used in a financial institution such as a bank. The mind-set involved in evolving such systems differs from the mind-set involved in integrating two similar systems or performing many enterprise-engineering roles.

Another question is, which domains might benefit from DevOps practices? One such domain is big data systems. Many big data systems rely on rapid deployment to support their data pipeline; thus, big data systems will rely more and more on DevOps practices.

DevOps Tools


Tool-related DevOps practices will also evolve. Currently, specialized tools exist for each portion of a pipeline. However, the overall pipeline flow must be hand-tailored using an orchestration engine or specialized

plug-ins for existing tools. Tools or tool families will emerge that are aware of the whole pipeline and that manage the orchestration of and configuration parameters for each pipeline stage. One step in that direction is ThoughtWorks' GoCD tool (www.thoughtworks.com/go). One analogy to tool evolution is programming-language evolution. Although it's possible to do everything in assembly language or C, domain-specific languages provide the abstractions that make specifying applications in the target domain easier.

This theme issue can only touch the surface of all the issues associated with DevOps. However, if you are migrating to microservices or have implemented them and are now dealing with postdeployment monitoring and reliability challenges, you should find this issue's articles helpful. 🍷

References

1. L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*, Addison-Wesley Professional, 2015.
2. D.R. Conner and R.W. Patterson, "Building Commitment to Organizational Change," *Training and Development J.*, April 1982, pp. 18–30.
3. S. Newman, *Building Microservices*, O'Reilly Media, 2015.
4. M. Maier, *The Art of System Architecting*, 3rd ed., CRC Press, 2009.



See www.computer.org/software-multimedia for multimedia content related to this article.