



DevOps

Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano

DevOps is about fast, flexible development and provisioning business processes. It efficiently integrates development, delivery, and operations, thus facilitating a lean, fluid connection of these traditionally separated silos. In this instalment of Software Technology, Gorka Gallardo, Josune Hernantes, Nicolas Serrano, and I present a brief overview of most recent DevOps technologies such as delivery tools and microservices and discuss what they mean for industry projects. I look forward to hearing from both readers and prospective column authors. —*Christof Ebert*

Quality deliveries with short cycle time need a high degree of automation.

DEVOPS INTEGRATES the two worlds of development and operations, using automated development, deployment, and infrastructure monitoring. It's an organizational shift in which, instead of distributed siloed groups performing functions separately, cross-functional teams work on continuous operational feature deliveries. This approach helps deliver value faster and continuously, reducing problems due to miscommunication between team members and accelerating problem resolution.¹

DevOps means a culture shift toward collaboration between development, quality assurance, and operations. Figure 1 shows the generic process, which aims to better integrate the development, production, and operations business processes with adequate technology. Instead of staying with highly artificial process concepts that will never fly, organizations set up continuous delivery with small upgrades. Companies such as Amazon and Google have led this approach, achieving cy-

cle times of minutes. Obviously, the achievable cycle time depends on the environmental constraints and deployment model; a single cloud service is easier to facilitate than actual software deliveries of real products.

DevOps can be applied to very different delivery models but must be tailored to the environment and product architecture. Not all products facilitate **continuous delivery**—for instance, in safety-critical systems. Nevertheless, even in such constrained environments, upgrades can be planned and delivered quickly and reliably, as the recent evolution of automotive software over-the-air shows. Besides highly secured cloud-based delivery, such delivery models need dedicated architecture and hardware changes. One example is a hot-swap controller in which one half is operational and the other half builds the next updates, which

are swapped to active mode after in-depth security and verification procedures. DevOps for embedded systems is more challenging than cloud and IT services because it attempts to combine legacy code and architecture with continuous delivery.

DevOps Tools

Tools are mandatory in automating DevOps. Quality deliveries with short cycle time need a high degree of automation. So, choosing the right tools for your environment or project is important when you move to DevOps. Table 1 lists the automation tools we discuss in this article.

Build

In this DevOps phase, the tools must support fast workflows. Here we discuss two types of tools. Build tools help achieve fast iteration, reducing manual time-consuming tasks. Continuous-integration (CI) tools merge code from all the developers and check for broken code, improving software quality.

Build tools. Build automation is crucial for DevOps environment—in the small and in the big. With agile and fluid development, build tools have also become tools for managing the software development and service life cycle, which involves compiling code, managing dependencies, generating documentation, running tests, or deploying an application to different environments.

Apache Ant has become the standard tool for building applications, especially in open source when you need to build from the sources. The simple syntax used to define the tasks to execute reflects Ant's main advantage and disadvantage. It's easy to understand but verbose even for simple tasks because it uses

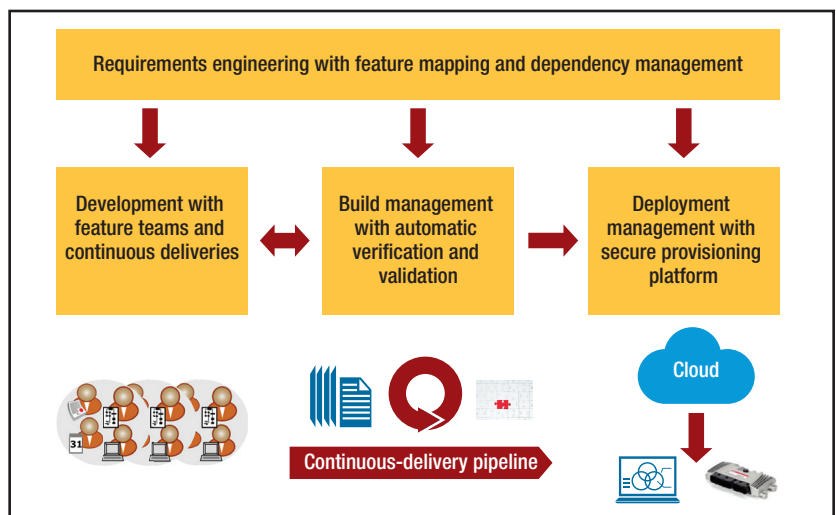
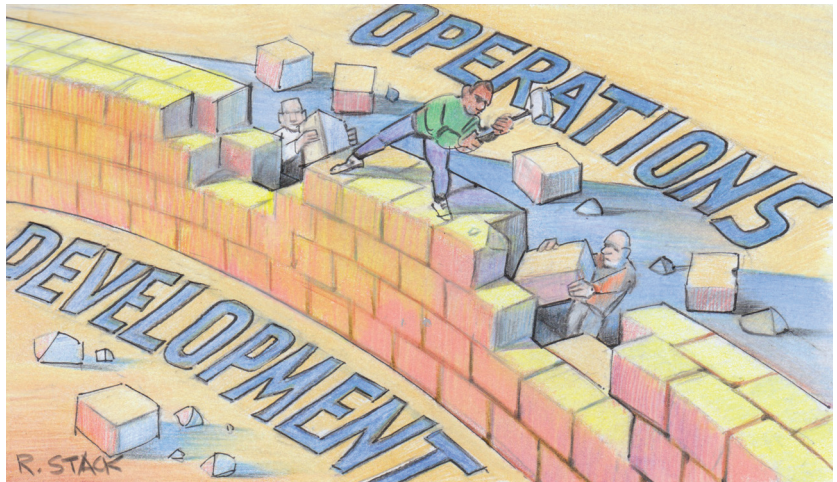


FIGURE 1. The generic DevOps production and delivery process. It aims to better integrate the development, production, and operations business processes with adequate technology.

XML, which isn't a procedural programming language.

Maven aims to solve some of Ant's problems. It also uses XML to define the build process. In this case, the statements define the nature of the project instead of the tasks that build it. Maven uses the **Project Object Model**, which defines a uniform way to build systems. This has the advantage of de-

fining a standard at the beginning of a project but sometimes causes inflexibility when custom operations must be defined. Managing the dependencies also helps automate the build process but sometimes creates problems due to the external automatic dependencies.

Rake and Gradle try to solve the problems of Ant and Maven by offering a programming-language-

TABLE 1

Automation tools for DevOps.

Tool	DevOps phase	Tool type	Configuration format	Language	License
Ant	Build	Build	XML	Java	Apache
Maven	Build	Build	XML	Java	Apache
Rake	Build	Build	Ruby	Ruby	MIT
Gradle	Build	Build	Based on Groovy	Java and a Groovy-based domain-specific language (DSL)	Apache
Jenkins	Build	Continuous integration	UI	Java	MIT
TeamCity	Build	Continuous integration	UI	Java	Commercial
Bamboo	Build	Continuous integration	UI	Java	Commercial
Puppet	Deployment	Configuration management	DSL similar to JSON (JavaScript Object Notation)	Ruby	Apache
Chef	Deployment	Configuration management	Ruby-based DSL	Ruby	Apache
Ansible	Deployment	Configuration management	YAML (YAML Ain't Markup Language)	Python	GPL (GNU General Public License)
Loggly	Operations	Logging	—	Cloud based	Commercial
Graylog	Operations	Logging	—	Java	Open source
Nagios	Operations	Monitoring	—	C	Open source and GPL
New Relic	Operations	Monitoring	—	—	Commercial
Cacti	Operations	Monitoring	—	PHP	GPL

based tool to build applications. With Rake, code lines are written in Ruby; with Gradle, configuration employs a **Groovy-based domain-specific language** (DSL).

Continuous-integration tools. CI is a key agile concept. CI tools integrate developers' work as early and as often as possible. In this way, the system is constantly tested. Embracing CI isn't always easy and straight-

forward. For example, in embedded-system development, testing is challenging because building and testing on the target hardware aren't always possible and thus must be simulated. Moreover, hardware limitations affect test speed and therefore cycle time.

Jenkins is an open source, Java-based system and one of the most implemented tools, so the plug-in options are many. Because of its

popularity, finding support from its large user base is easy. However, its UI is outdated and less attractive than the other tools' UIs.

TeamCity, developed by ITBrains, is based on Java and thus has good Java support, like Jenkins. Unlike Jenkins, it also offers good .NET support. Although it's licensed, it has a free edition for small projects.

Bamboo is from Atlassian and integrates well with other Atlassian

tools. So, it offers advantages if you're already using Atlassian tools.

Deployment

During this phase, the most important shift is to treat infrastructure as code. With this approach, infrastructure can be shared, tested, and version controlled. **Development and production share a homogenous infrastructure, reducing problems and bugs due to different infrastructure configurations.**

DevOps pushes automation from the application to the infrastructure. Compared with manual infrastructure provisioning, configuration management tools can reduce production provisioning and configuration maintenance complexity while enabling recreation of the production system on the development machines. Such tools are a major DevOps enabler, especially as architecture moves from a monolithic block of software to a microservices approach.² (We discuss microservices in more detail later.)

To achieve continuous delivery for embedded systems, devices must be connected, so machine-to-machine communication and an Internet of Things architecture are necessary. Continuous delivery will improve time to market and allows agile practices with rapid consumer feedback. However, for critical applications in which failure isn't an option—for example, medical or aerospace applications—a more conservative approach is preferable. The following tools will improve the application life cycle.

Puppet's approach to infrastructure provisioning is about describing the system's desired end state. A Puppet-based infrastructure generally consists of a master server with agents on each client. Puppet is based on Ruby but uses a DSL similar to JSON (JavaScript Object Notation)



CONTAINERIZATION OR VIRTUALIZATION?

Because of differences between development and production machines, to achieve consistency, developers use virtualization technologies to recreate production environments on development machines. But virtualization technologies focus on security and multitenancy, whereas in application development, environmental consistency is the main goal.

In this scenario, containerization can be a lightweight alternative to virtualization. Moreover, virtualization lacks compatibility between different providers, whereas containerization enhances migration between cloud providers. So, traditional hypervisor vendors, such as VMware with Project Bonneville and Microsoft with Windows Server 2016, are integrating containerization support on their platforms.

to represent machine configurations.

Chef's approach is more about writing a recipe on a purely Ruby-based DSL that details the required steps to provision the system. You can use Chef as a client-server tool or an isolated installation in "chef-solo" mode.

Ansible is the easiest to implement because it doesn't require installing agents on the client machines, because it uses SSH (Secure Shell) to push configurations. It's an open source tool based on Python, but configuration is coded in YAML (YAML Ain't Markup Language) files, reducing the learning curve.

Finally, when selecting your orchestration technology, you need to take into account whether your production infrastructure is based on virtualization or you want to move to containerization technologies such as Docker. (For more on this, see the "Containerization or Virtualization?" sidebar.)

Operations

Although researchers have focused considerably on build and deployment, DevOps also impacts infra-

structure operations. So, you need tools to maintain your infrastructure's stability and performance.

Logging tools. Logging is sometimes considered a replacement for debugging, and programmers have been advised to debug instead of log so that they don't leave traces in an application. But from a DevOps viewpoint, traces are necessary for managing applications. The rule is to log everything and determine the log's level (for example, fatal, error, warn, info, trace, or debug). For applications that aren't very complex, you can use the standard tools (in Java, that's the `java.util.logging` package). For more complex projects, you can use frameworks such as Log4j or the Log4j tools' family for different languages, although the best-known tools for these tasks are log management systems.

The Loggly tool offers a cloud-based service to collect log data from applications, platforms, and servers in real time using open standards such as HTTP or syslog. Installation is simple, and creating custom performance and DevOps dashboards

is also easy. Loggly also provides a powerful search feature. Loggly can be integrated with New Relic, making it effective for identifying and isolating problems and finding solutions.

Graylog2 is an open source log analyzer that allows storing and searching through log errors. In its easy-to-use Web interface, you can build ad hoc dashboards and strong alerting features. It handles a large range of data formats and can be customized with a considerable number of plug-ins.

Monitoring tools. Monitoring tools let organizations identify and resolve IT infrastructure problems before they affect critical business processes. These tools monitor system aspects such as the CPU load, RAM allocation, network traffic statistics, memory consumption, and availability of free disk space. They continuously track the system's health and alert administrators when they detect problems so that the administrators can take corrective actions.³

Nagios is a well-known open source tool for monitoring IT infrastructures such as end-user stations, IT services, and active network components. It provides an easy-to-navigate Web interface with an interactive dashboard that includes a high-level overview of hosts, services, and network devices. It provides trending and capacity-planning graphs that let organizations plan infrastructure upgrades. The plug-ins solve some of the tool's limitations, such as the lack of automatic device discovery and the difficulty of configuring the tool.

New Relic, based on software as a service, provides a powerful interface for Web apps and monitors native mobile applications for iOS and Android. You can monitor apps running in hybrid, cloud, or on-premises

environments. Customizable dashboards let you integrate additional monitoring features. You can generate custom reports and set threshold alarms to trigger notifications of specific events.

Cacti is a Web-based system with an intuitive, easy-to-use interface. Its template-based configuration enables a high level of customization. You can build graphs, which can be displayed in tree view mode in addition to the standard list view and preview modes.

DevOps in the Real World

The software and IT industries are about speed and efficiency. DevOps has emerged as a paradigm to bring innovative products and features faster to the market. Many technologies have recently popped up to smooth the transition between development and operations. They have in common fluid delivery practices, thus managing complexity. Technologies for fast application delivery, such as for a search or selling platform, don't apply to embedded software. But we can certainly learn from those approaches and map them to other domains. As we mentioned before, the emergence of over-the-air technologies for fast automotive software updates shows that the principles are transferable, while taking into account the stringent needs of continuous functional safety and security.

Cloud and Web development have been early adopters of DevOps practices and can serve as guide for other domains. For example, Amazon Web Services (AWS) offers several tools for implementing continuous delivery. One such tool is AWS Elastic Beanstalk, which supports continuous deployment with an easy approach and therefore a shallower learning curve, but with less configurability.

AWS OpsWorks offers a middle way, in which you can code infrastructure; it offers integration with Chef. You can also create an AWS CloudFormation template, written in a JSON format, to provision infrastructure in a repeatable way and to control all the cloud infrastructure components. You can use the AWS CodeDeploy service to deploy applications across multiple virtual machines (Elastic Compute Cloud instances) with minimal downtime.

Alternatively, you can use AWS CodePipeline. This service, launched in 2015, integrates build, testing, and implementation.

Complementarily, you can use other AWS components to support continuous delivery. AWS CodeCommit is a managed source control service that hosts private repositories. AWS CloudWatch offers a monitoring and alerting infrastructure that can help the whole team work on the deployed application's reliability and performance.

Figure 2 shows a DevOps architecture that uses AWS tools.

Beyond DevOps: Microservices

Researchers have been discussing microservices to better manage complex legacy architectures. **Microservices software breaks systems and applications down to a more granular, modular level.** Such software emerged approximately 10 years ago as a follow-up to service-oriented architecture. This approach is about fragmenting complex applications and delivering services on demand, thus improving performance.

DevOps provides the framework for developing, deploying, and managing the microservices container ecosystem. DevOps benefits from a refactored architecture but doesn't

need it. Its major advantage is fast delivery cycles by integrating development and operations.

DevOps and microservices should be cloud-based as much as possible, especially if the focus is on efficient service delivery. Unlike a centrally governed cloud solution, microservices are inherently distributed. Each delivery has dependency impacts that must be analyzed, validated, and considered for packaging. When microservices are operated across networks, the deliveries will incur significant performance penalties that must be accommodated with additional architectural tweaks, such as caching layers. More complex and critical applications with availability and security constraints shouldn't be addressed entirely by a volatile cloud delivery model.

Because microservices need DevOps, we recommend starting with a tailored DevOps strategy. It will have immediate value owing to better integration across the life cycle and can gradually evolve to a microservices delivery model, if that's appropriate.

The DevOps Culture Shift

Vector Consulting Services has helped several companies improve efficiency with DevOps and continuous delivery. (For an example, see the sidebar "A DevOps Case Study.") A key lesson for all these companies has been not to underestimate the culture shift. All DevOps projects face four major related challenges:

- breaking complex architectures and feature sets into small chunks that can be produced and deployed independently;
- maintaining a configuration and build environment that provides

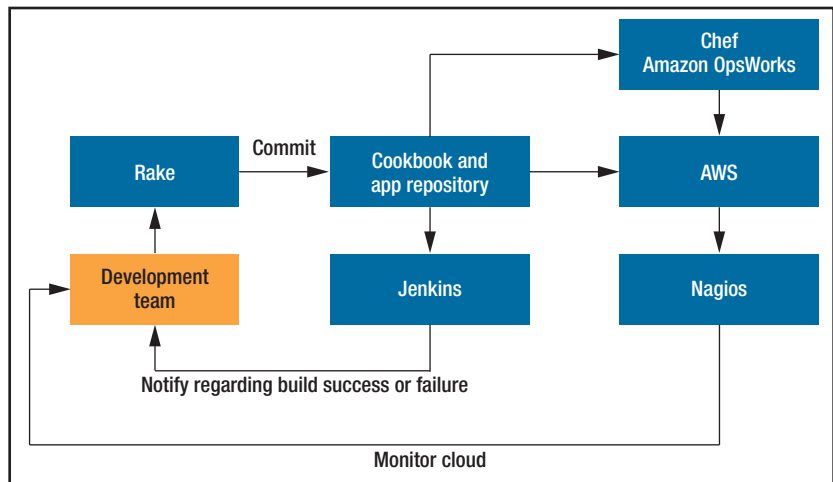


FIGURE 2. A DevOps architecture that uses Amazon Web Services (AWS) tools to implement continuous delivery.

constant visibility of what's deployed, with which versions and dependencies;

- introducing a purpose-built development and production environment derived from legacy application life-cycle management or product life-cycle management environments; and
- bridging the traditionally siloed cultures of development (which operations folk perceive as cumbersome and expensive because of its thoroughness) and operations (which developers perceive as quick and dirty).

Transitioning toward DevOps is obviously more challenging with legacy software, both in the IT and embedded domains. Most case studies have involved application development and Web environments that migrated rather easily to DevOps because there was a single source and thus orchestrated rollback was feasible. Real-world deployed software isn't like that.

When embracing a DevOps culture, developers must take a full-

stack developer approach, in which they take responsibility for the testing and release environment. They must master an extended skill set beyond code knowledge, including database administration and testing. Furthermore, as the functional silos' boundaries blur, more intense collaboration with other team members is required.

When developers embrace DevOps, testing is a critical part of development, and the development team performs test-driven development and CI. In this scenario, testers can be paired with developers, and both can gain technical knowledge. The quality assurance team must ensure automation of all test cases and full code coverage.

From an operations perspective, DevOps greatly affects culture and discipline. Operations teams must continuously connect to other functions without losing control. So, the IT operations and development teams must closely collaborate to achieve the continuous process promoted by the DevOps team. Moreover, infrastructure monitoring and application performance manage-

A DEVOPS CASE STUDY

A global supplier of critical infrastructure solutions faced an overly long cycle time and high rework of delivered upgrades. The overall delivery process, from development to deployment in the field, took 18 months for new products and up to three months for upgrades. This was far too long, even in this domain.

Vector Consulting Services introduced a DevOps model tailored to these environmental constraints. Figure A shows the eight focus areas mapped to the V-shaped life-cycle abstraction. The key change was the enhanced requirements-engineering and delivery model (numbers 1 and 2 in the figure).

By running the automated tests and static and runtime analysis with every check of automatic build management, our client could discover defects early during development. Fewer changes during feature development and less rework due to quality issues directly improved the return on investment. Software releases became more consistent and less painful because tests were run early and often. Owing to better quality and fewer changes, the overall end-to-end cycle time decreased by almost 12 months for products and a few days for small upgrades.

1. Requirements engineering with feature mapping and a delivery model
2. Architecture and delivery management
3. Modeling and tool support
4. Feature teams with ownership and Scrum
5. Incremental development and continuous delivery
6. Automatic build and release processes
7. Code quality with static analysis
8. Efficient feedback from the field

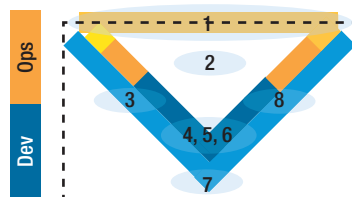



FIGURE A. A DevOps model tailored to the environmental constraints of a global supplier of critical infrastructure solutions. The eight focus areas are mapped to the V-shaped life-cycle abstraction.

understanding from requirements onward to maintenance, service, and product evolution will improve cycle time by 10 to 30 percent and reduce costs up to 20 percent. The major drivers are fewer requirements changes, focused testing and quality assurance, and a much faster delivery cycle with feature-driven teams. However, because products and life-cycle processes vary, each company needs its own approach to achieve DevOps, from architecture to tools to culture. 

References

1. M. Virmani, "Understanding DevOps & Bridging the Gap from Continuous Integration to Continuous Delivery," *Proc. 5th Int'l Conf. Innovative Computing Technology (INTECH 15)*, 2015, pp. 78–82.
2. D. Spinellis, "Don't Install Software by Hand," *IEEE Software*, vol. 29, no. 4, 2012, pp. 86–87.
3. J. Hernantes, G. Gallardo, and N. Serrano, "IT Infrastructure-Monitoring Tools," *IEEE Software*, vol. 32, no. 4, 2015, pp. 88–93.

CHRISTOF EBERT is the managing director of Vector Consulting Services. He is on the *IEEE Software* editorial board and teaches at the University of Stuttgart and the Sorbonne in Paris. Contact him at christof.ebert@vector.com.

GORKA GALLARDO is a professor of information systems at the University of Navarra. Contact him at ggallardo@tecnun.es.

JOSUNE HERNANTES is a professor of computer science and software engineering at the University of Navarra. Contact her at jhernantes@tecnun.es.

NICOLAS SERRANO is a professor of computer science and software engineering at the University of Navarra. Contact him at nserrano@tecnun.es.

ment are more important, and communication among team members must be fluid.

DevOps is impacting the entire software and IT industry. Building on lean

and agile practices, **DevOps means end-to-end automation in software development and delivery.** Hardly anybody will be able to approach it with a cookbook-style approach, but most developers will benefit from better connecting development and operations. Typically, mutual