DEVOPS-AN OPEN ACCESS JOURNAL
Open Journal

Review

DevOps

Christof Ebert*; Gorka Gallardo; Josune Hernantes; Nicolas Serrano
University of Navarra, Pamplona, Spain

**Corresponding author*
Christof Ebert
University of Navarra, Pamplona, Spain; E-mail: christof.ebert@vector.com

Article information
Received: August 14th, 2022; **Revised:** October 24th, 2022; **Accepted:** November 1st, 2022; **Published:** November 13th, 2022

Cite this article
Ebert C, Gallardo G, Hernantes J, Serrano N. DevOps. 2022; 1(1). doi: https://doi.org/10.70705/ppp.doaj.2022.v01.i01.pp22-25

DevOps is about fast and flexible development and delivery business processes. It efficiently integrates development, delivery and operations and thus facilitates a lean and fluid connection of these traditionally separated silos. Authors Gorka Gallardo, Josune Hernantes, Nicolas Serrano, and myself present a brief overview on most recent DevOps technologies such as delivery tools and Microservices, and what it means for industry projects. I look forward to reading from both readers and prospective column authors.

— *Christof Ebert*

The DevOps Promise

DevOps is a software practice that integrates the two worlds of development and operations with automated development, deployment and infrastructure monitoring. It's an organizational shift where instead of distributed silo-like functions cross-functional teams work on continuous operational feature deliveries. This integrative approach helps teams deliver value in a faster and continuous way, reducing problems generated by miscommunication between team members and enhancing a faster resolution of problems [1].

DevOps means a culture shift towards collaboration between development, quality assurance and operations. The generic process is indicated in Fig. 1. Its promise and goal is to better integrate the development, production and operations business processes with adequate technology, thus not staying on highly artificial process concepts which will never fly, but rather set up a continuous delivery process with small upgrades. Companies such as Amazon and Google have lead this approach achieving cycle times of minutes. This obviously depends on the deployment model, whereas a single cloud service is easier to facilitate than actual software deliveries to real products.

DevOps applies to these very different delivery models, but must be tailored to the environment and product architecture. Not all products facilitate continuous deliveries, for instance in safety-critical systems. Nevertheless upgrades can be planned and delivered in a fast and reliable scheme, as recent evolution of automotive software over the air (OTA) upgrades show. Aside the highly secured cloud based delivery model, such delivery models also need dedicated architecture and hardware changes, for instance a hot swap controller

concept, where one half is operational and the other half builds the next updates which are swapped to active mode after in-depth security and verification approaches. DevOps for embedded systems is more challenging than cloud and IT services due to the dependence on legacy code and architecture, and trying to fit it into a continuous delivery approach.

DevOps Technologies

DevOps is not about adopting specific tools. Tools are mandatory in automating DevOps. Quality deliverables are only possible with a high degree of automation. From this perspective, choosing the right tools for your environment or project it is an important step when moving to a DevOps practice.

In the build phase the tools need to support fast workflows. From this perspective, build tools help to achieve fast iteration reducing manual time consuming tasks, and Continuous integration tools merge code from all developers and check for broken code, improving software quality.

During the deployment phase the most important shift is to treat Infrastructure as code. With this approach infrastructure can be shared, tested and version controlled. A homogenous infrastructure is shared between development and production, reducing problems and bugs because of difference in infrastructure configuration.

Finally, although a lot of attention has been focused on the build and deploy phases DevOps also impacts operation of the infrastructure. From this perspective we need tools to help to maintain the stability and performance of your infrastructure as shown in Figure 2 and detailed next.

Build automation

Build automation is crucial for DevOps environment – in the small and in the big. With agile and fluid development paradigms they have also became a tool for managing the lifecycle of software development and service, which involves compiling code, managing dependencies, generating documentation, running tests or deploying the application to different environments. Here are some relevant tools (see Table):

Apache Ant was released in 2000 and became the standard way to build applications, especially in open source when you need to build from the sources. The simple syntax used to define the tasks to be executed reflects its main pros and cons: easy to understand but verbose even for simple tasks due to the use of XML since it is not a procedural programming language

Maven was set up to solve some of Ant's problems. It also uses XML to define the tasks. In this case, the statements define how the project is instead of the tasks that build it. Maven uses the Project Object Model (POM), which defines a uniform way to build systems. This has the advantage of defining a standard at the beginning of a project, but sometimes this becomes a point of inflexibility when custom operations need to be defined in a project. Managing the dependencies also helps automate the build process, but it sometimes creates problems due to the external automatic dependencies.

Gradle was released in 2012 and Rake was released in 2003. Both tools try to solve the problems of Ant and Maven by defining the process of building the application with a tool based on a programming language. With Rake code lines are written in Ruby and with Gradle the configuration is done with a Groovy DSL.

Continuous Integration

Continuous integration (CI) is a key agile development concept. The main goal of these tools is to integrate developers work as often and early as possible. In this way, the system is constantly tested. Embracing a continuous integration practice is not always easy and straightforward. For example, in the case of an embedded system development, testing is challenging as building and testing in the target hardware it is not always possible, and it has to be simulated. Moreover, hardware limitations will affect importantly speed test and therefore cycle time. Here are some typical tools:

Jenkins is an open source Java based CI system and one of the most implemented tools, so the plugin options are wide. Because of its popularity is easy to find support in its large user base. However, the UI is outdated and less attractive than other alternatives.

TeamCity is a CI system developed by ITBrains based on Java, and therefore it has a good Java support like Jenkins, however it also offers a good .NET support. Although licensed, it has a free edition for small projects.

Bamboo is a CI software from Attlasian that integrates well with the rest of Attlasian tools, so being an advantage if we are already using other Attlasian tools.

Configuration Management

DevOps push automation from application to infrastructure. Compared with manual infrastructure provisioning, configuration management tools can reduce production provisioning and configuration maintenance complexity and at the same time they will enable to recreate the production system in the development machines. Configuration management tools are a major DevOps enabler especially as architecture is moving from a monolithic block of software to a microservices approach [2].

In order to achieve continuous delivery for embedded systems devices must be connected, so machine to machine communications and an IoT (Internet of things) architecture should be implemented. Continuous delivery will impact time to market and allows agile practices with a rapid consumer feedback. However, in critical applications where failure is not an option, for example medical or aerospace, a more conservative approach should be considered. The following tools will have an important positive impact on the application lifecycle:

Puppet approach for infrastructure provisioning is about describing the desired end state of the system. Puppet infrastructure generally consists of a master server with agents installed on each client. Puppet is based on Ruby, but uses a JSON like DSL to represent machine configurations.

Chef approach is more about writing a recipe on a pure Ruby DSL that details the required steps to provision the system. Chef can be used either as a client-server tool or as an isolated installation in "chef-solo" mode.

Ansible is the easiest to implement as it does not require agent installation in the client machines, as SSH is used to push configurations. Ansible is an open source tool based on Python, while configuration is coded in YAML (Yet Another Markup Language) files, reducing the learning curve.

Logging

Logging is sometimes considered to be a replacement for debugging, and it is recommended that programmers not log and debug instead in order to not leave traces in the application. But from a DevOps point of view, traces are a necessary tool for managing the application. The rule is to log everything and determine the level of the log (e.g.: Fatal, Error, Warn, Info, Trace, Debug). For applications that are not very complex the standard tools can be used (in Java it will be the java.util.logging package), while for other more complex projects frameworks like log4j or their family of different languages can be used although the most well-known tools for these tasks are the log management systems. Here are the respective tools:

Loggly offers a cloud-based service to collect log data from applications, platforms and servers in real time using open standards such HTTP or syslog. The installation process is simple, and creating custom performance and devops dashboards is also an easy process in addition to provide a powerful search feature. Loggly can be integrated with New Relic, making it an effective tool for identifying and isolating problems and finding solutions.

Graylog2 is an open-source log analyzer that allows storing and searching through log errors. This tool has an easy to use web interface, where ad-hoc dashboards can be built in addition to strong alerting features. It can handle a large range of data formats and can be customized with a considerable number of plugins.

Monitoring

Monitoring tools enable organizations to identify and resolve IT infrastructure problems before they affect critical business processes. They monitor the system including CPU load, RAM allocation, network traffic statistics, memory consumption, and the availability of free disk space, among other things. These monitoring tools continuously track the health of the system and alert administrators when problems are detected so that they 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.. Nagios provides an updated, 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 plugins solve some of the tool's limitations, such as the lack of automatic device discovery and the fact that it is difficult to configure.

New Relic is a SaaS-based monitoring solution that provides a powerful interface for web applications. It also monitors native mobile

applications for iOS and Android. New Relic allows apps that are running in hybrid, cloud or on-premise environments to be monitored. New Relic dashboards can be customized allowing the integration of additional monitoring features. Custom reports can be generated and threshold alarms can be set to trigger notification for specific events.

Cacti is a web-based monitoring system with an intuitive and easy to use interface. It enables a high level of customization thanks to its template-based configuration. It allows building graphs, which can be displayed in tree view mode in addition to the standard list view and preview modes.

Example: AWS

Amazon Web Services (AWS) offers three alternatives in order to implement continuous delivery to a given infrastructure. First of all, AWS Elastic Beanstalk supports continuous deployment with an easy approach, and therefore a shallower learning curve but with less configurability. AWS OpsWorks offers a middle point where infrastructure can be coded offering integration with Chef Configuration management software. An AWS CloudFormation template can be created, written in a JSON format, to provision infrastructure in a repeatable way, and control all components of the cloud infrastructure. For applications, AWS CodeDeploy service can be used to deploy applications across multiple VMs (EC2 instances) with minimum downtime. Alternatively, AWS CodePipeline can be considered. This service launched in 2015, integrates build, test and implementation. Complementarily, other AWS components can be used to support a continuous delivery infrastructure. AWS CodeCommit component is a managed source control service that hosts private repositories, while AWS Cloudwatch offers a monitoring and alerting infrastructure that can help the whole team to work on the reliability and performance deployed application.

Beyond DevOps: Microservices

The software and IT industries are about speed and efficiency. DevOps has emerged as a paradigm to bringing innovative products and features faster to the markets. Many technologies recently pop up to smoothen 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, do not apply for embedded software. But we can certainly learn from those approaches and map them to other domains. The emergence of "over the air" technologies for fast automotive software updates shows that principles can be transferred, while considering their stringent needs of continuous functional safety and security.

For a while microservices have been discussed to better manage complex legacy architectures. Microservices software breaks systems and applications down to a more granular, modular level. It has been created as a SOA follow-up some ten years ago. It is about fragmenting complex applications to small pieces and a fluid delivery model where services are delivered 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 does not need it. Its major advantage is fast delivery cycles by integrating the before silo-style business processes of development and operation.

DevOps and Microservices should be as much as possible cloud-based, especially if it is only about service delivery. However, microservices as well as complex features by nature are distributed. Each delivery has dependency impacts which must be analyzed, validated and considered for packaging. When operated across networks they will cause significant performance penalties that must be accommodated with additional architectural tweaks, like caching layers. Obviously more complex and critical applications with availability and security constraints should not be addressed entirely by a volatile cloud delivery model. Since microservices need DevOps, we recommend starting with a tailored DevOps strategy. It will have immediate value due to better integration across the life-cycle and can gradually evolve to a microservices delivery model – if appropriate.

The DevOps Culture Shift

At Vector we have supported a number of companies on improving efficiency with DevOps and continuous delivery. A key learning for all companies is that the culture shift should not be underestimated. There are four major challenges which we face in all DevOps projects, namely:

- Break complex architectures and feature sets towards small chunks that can be produced and deployed independently.
- Maintain a configuration and build environment which provides visibility at all times about what is currently deployed with which versions and dependencies.
- Introduce a purpose-built development and production environment from legacy ALM/PLM environments.
- Bridge the traditional silo-type cultures of development (perceived by operations in its thoroughness as cumbersome and expensive) and operations (perceived by developers as quick and dirty).

Transitioning towards a DevOps paradigm is obviously more challenging with legacy software, both in IT and embedded domains. Most case studies in literature show application development and web environments which are rather easily migrated to DevOps as there is a single source and thus orchestrated rollback feasible. This is different in real-world deployed software.

When embracing a DevOps culture, developers take a full-stack developer approach and must take responsibility of the testing and release environment. Developers need to master an extended skillset beyond code knowledge, including DBA and testing. Furthermore, as boundaries of functional silos blur, a more intense collaboration with other team members is required.

When embracing a DevOps approach test is a critical part of the development process, and TDD (test-driven-development) and CI (Continuous Integration) are testing practice performed by the development team. In this scenario testers can be paired with developers, and both can gain technical knowledge. Quality assurance responsibility is to assure that all test cases are automated and that full code coverage is achieved.

From an operations perspective, a DevOps practice has great impact on culture and discipline. With DevOps an operations team must continuously connect to other functions without losing control. Therefore, a close collaboration between IT operations and development is needed to achieve the continuous process promoted by the DevOps team. Moreover, infrastructure monitoring and application performance management is more important in this approach and a

fluid communication with other team members is needed.

DevOps is a paradigm shift impacting the entire software and IT industry. Building upon 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 will benefit from better connecting the previously isolated silos of development and operations. Mutual understanding from requirements onwards to maintenance, service and product evolution will yield typically a cycle time improvement of 10-30% and cost reduction of up to 20%. Major drivers are less requirements changes, focused testing and quality assurance, and much faster delivery cycle with feature-driven teams. As products and life-cycle

processes vary, each company needs its own approach towards a DevOps environment, from architecture to tools and culture.

REFERENCES

[1] Virmani, M. (2015), "Understanding DevOps & Bridging the gap from Continuous Integration to Continuous Delivery", Proceedings of Fifth International conference on Innovative Computing Technology (INTECH 2015), pp.78-82.

[2] Spinellis, D.(2012), "Don't install software by hand", IEEE Software, vol.29 (4), pp. 86 – 87.

[3] Hernantes, J. Gallardo, G.; Serrano, N. (2015), "IT Infrastructure-Monitoring Tools", IEEE Software, vol.32 (4), pp. 88 – 93.