DevOps best practices, Part 1: Develop reliable software with DevOps

Bob Aiello
Leslie Sachs

June 01, 2016
(First published May 11, 2013)

Learn the best practices to use in order to create reliable software systems and avoid the high profile incidents that have recently impacted so many other companies.

View more content in this series

Software systems impact just about every aspect of our lives. From the web-based portals that we use for online shopping to the large-scale enterprise systems that run our businesses, we expect computer systems to provide a wide array of functions, to scale to meet peak usage demands, and to learn our preferences so that they can anticipate our every need. When we work with one well-written software system, we become accustomed to its power and flexibility and we're surprised when other software systems don't meet or exceed the same expectations.

Computer glitches with disastrous results

Because computers play a role in many of our daily tasks, we are particularly impacted when the systems we depend upon are less than one hundred percent reliable. This article focuses on the best practices to adopt to create reliable software systems and to avoid the high-profile incidents that have recently affected so many other companies.

An example from Knight Capital Group

In August of 2012, Knight Capital Group, Inc. suffered a disastrous computer glitch that proved fatal to the company's very existence as an independent organization. Published reports indicated that a failed attempt to upgrade software related to the New York Stock Exchange (NYSE) Euronext resulted in the company inadvertently purchasing what was reported to be seven billion dollars of stocks that they did not want.

Knight had to liquidate the stock immediately, an action that resulted in a loss of over 440 million dollars. The loss grew as some customers withdrew their business in what some industry experts claim was a loss of confidence in the company's capabilities. Knight Capital was acquired by another firm. It went out of business primarily because the company lacked adequate procedures for managing a software upgrade. This well-publicized and dramatic incident is only one of many
recent software glitches affecting banking systems, trading firms, stock exchanges as well as critical government systems such as the New York City 911 emergency system.

**Similar examples from the stock market**

A number of other recent software glitches have had similar significant consequences. In May 2013, the Chicago Board Options Exchange (CBOE) trading system was shut down for business because of a software glitch that was reported to be related to systems configuration changes required for extending the trading day. This incident has brought into question whether or not the CBOE should really be the single source of options based on the S&P 500 index and the VIX gauge of equity volatility. This incident occurred just as other firms had been challenging the CBOE’s exclusive right to manage these trading indexes and served to demonstrate that trading exchanges might not be able to continue to be a single source for these options.

Some industry experts have questioned whether large-scale computer systems have become so complex that it is impossible for any company or organization to ensure that these enterprise-wide systems are reliable and free from service outages. This article answers that question. We know exactly how to develop large, mission-critical systems that are completely reliable. In fact, many industry standards and frameworks that establish industry best practices, aligned with DevOps, can help ensure a high degree of reliability in computer systems.

**Industry standards and frameworks**

The IEEE, ISO, and several other well-respected standards organizations provide detailed guidance on how to develop reliable and safe systems. Using these industry best practices, you can demonstrate that you have effective IT controls to help ensure that your systems are reliable and meet most federal regulatory requirements. In this series, we will discuss the practical implementation of DevOps best practices in the context of relevant industry standards and frameworks and explain how to implement the necessary automated procedures in a practical and realistic way that naturally aligns with the practices and principles associated with DevOps.

For example, the Information Technology Infrastructure Library (ITIL) is a set of practices that support IT service management (ITSM) by focusing the alignment of IT services with the needs of the business. ITIL V3, one of the most popular frameworks, provides guidance on how to ensure that IT services can be maintained and upgraded without the risk of service interruption. ITIL describes the Configuration Management System (CMS) that is used to track changes to all configuration items (CIs). It also describes the Configuration Management Database (CMDB) that supports the CMS by providing updated and accurate information on the status of CIs on runtime systems.

**Configuration Management Database requirements support**

The CMDB is an essential component of the CMS, providing updated and accurate information through automated discovery procedures. DevOps helps to keep the CMDB up to date by enabling mature build procedures. In practice, application code can only be discovered and accurately identified if it has been built using embedded and immutable version IDs. Therefore, your build
engineering effort must include a procedure to embed the version ID in the configuration item itself and to embed the version ID into the manifest of the container, such as a jar, war, or ear file in which it is packaged.

DevOps satisfies this requirement by providing the build, release, and deployment automation that enables the CMDB to provide accurate information about application code to the CMS.

The incident at Knight Capital Group might have been prevented by a CMDB that could discover the versions of the code on their servers and verify against the expected version in the CMS. These techniques are possible only if the development and operations organizations work together to build and automate them early in the software development lifecycle.

**DevOps builds quality in with release engineering**

Release engineering provides the most effective approach to building and packaging code that can be verified on the target servers. This approach prevents the very glitches that have become so commonplace in the financial services industry today. Successful implementation depends on the effective collaboration of the development and operations teams to:

- Build in version identification
- Create discovery tools and processes, run by operations, that:
  - Verify that the correct code has been deployed
  - Verify that no unauthorized changes have occurred due to either malicious intent or human error.

Each team has a particular purpose:

- **Operations team focus**: To maintain reliable service.
- **Development team focus**: To develop new functions.
- **DevOps focus**: To ensure collaboration between the operations team and the development team to get these automated procedures in place to prevent software glitches due to the wrong version of a piece of code being deployed on a production server.

Using a DevOps approach to build, package, and deploy applications enables organizations to build quality in from the beginning, as advocated by quality guru W. Edwards Deming. If we know how to prevent the sorts of issues that have occurred at Knight Capital, why aren't more organizations embracing these industry best practices?

**DevOps actually saves money**

The most common excuse for failing to establish required IT controls is that it just costs too much and takes too much time. In many organizations, challenging deadlines and the pressure to deliver new functionality results in cutting corners, a decision that frequently leads directly to software defects that range from missing requirements to introducing errors into the codebase. Quality does cost something, but the cost of delivering code with defects is also great and can include monetary loss and, more significantly, the loss of confidence in the organization itself.
DevOps places the primary focus on building in quality from the beginning. This focus is essential for the company to deliver software that works and that supports the business.

**DevOps minimizes the impact of complexity on software development**

Another common excuse for poor quality is that the software is just getting too complex.

Software systems are indeed providing more and more complex functions. Most large software systems cannot be completely understood by any one technology professional. We all work with software frameworks that enable us to write code faster and deliver more functionality, but these advantages often come at the cost of using components (written by others) that we do not completely understand.

It is possible, however, to manage each piece of the software solution if automated procedures are developed to build, package, and deploy the application. These procedures can be created to verify the interfaces to runtime dependencies and to ensure that the environment is correctly configured to support all of the components that are required, including the build and deployment of the components themselves. By developing automated build, package, and deployment procedures for each component, the overall complexity of the system can be tamed and managed effectively.

**DevOps manages build, package, and deployment dependencies**

Implementing automated build, package, and deployment processes is an essential focus of any DevOps effort. Many software developers are focused entirely on working within their Integrated Development Environments (IDE), such as Eclipse and Visual Studio.

The problem is that they might not actually know and understand all of their build dependencies. When these developers move on to their next project, or an accident causes a laptop to crash, the organization might find it does not have the required knowledge to build, package, and deploy their code. It is quite common for developers to have only a partial understanding of their build and runtime dependencies. This is the precise situation in which a build engineer, often required by industry regulations for a separation of duties, can enhance reliability by capturing the required knowledge and automating the build and deployment pipeline.

**DevOps improves the reliability and the deployment pipeline**

Scripting and automating the build ensures that the essential knowledge of compile and runtime dependencies is discovered and documented. The developer may have long forgotten all of the environment settings that were configured in the IDE, but fortunately, the build scripts written in Ant, Maven, or Make provide a clear and accurate view of the essential configuration required to build, package and deploy the code.

Being able to reliably build, package, and deploy in a consistent way is essential to ensure that the system can be supported and modified without unintended and serious consequences. Aside from being able to reliably build the code, we also need to ensure that we can verify that the correct
code has been deployed and more importantly, that any unauthorized changes from malicious intent or human error are immediately identified.

**Correct deployment depends on use of cryptography and baselining**

After the application code has been compiled and packaged, it is important for the deployment engineer to verify that the code has been correctly deployed. Problems can be introduced here for many reasons. Sometimes, the code does not get to the target machine as intended, either because of permission issues or because of simple human error.

Although we test the application code that is written and verify that it meets the original requirements and design, many deployment engineers forget to verify that the code that has been built is actually copied successfully to the target machine. The correct way to handle this is to use techniques such as cryptography to verify that the exact code that was built is actually deployed to the target machines. Cryptography can also be used to identify and detect any unauthorized changes that could potentially result in a systems outage.

**Move the build, package, and deployment functions upstream**

Each of the techniques described in this article require some effort and technical expertise. Many companies try to implement these controls after the code is deployed to production at a point that is simply too late in the process.

DevOps puts the proper focus on implementing the automated deployment pipeline very early in the application development lifecycle. The decision to write code that can be verified, builds quality into the process from the beginning, a fundamental principle of effective DevOps.

Organizations usually have build engineering teams who are responsible for automating the application build, package, and deployment from the very beginning of the software development lifecycle. Code, such as the code that embeds immutable version IDs described earlier in this article, should be written to facilitate the automation effort. Unfortunately, some organizations err by leaving developers to handle their application builds in the beginning of the software development lifecycle. This is a mistake. If the build team automates the build, package, and deploy of the code starting in the beginning with the development test environment, developers can enjoy these practices and write code faster. The quality assurance and testing group also benefits from automating the application build, package, and deploy tasks because these automated techniques ensure that the code that is tested, matches the code that is deployed. In addition, automating deployment tasks helps ensure that the application works and is free from defects that could potentially impact systems reliability. DevOps considers the provisioning task to be a job for code.

DevOps correctly identifies the task of provisioning the infrastructure (and provisioning the servers in a cloud-based environment) as a code and development effort. In addition, the task of configuring servers in a secure and reliable way is also a code effort.

On a similar topic, the task of managing the deployment pipeline is a software and systems development effort, which must include its own lifecycle. DevOps puts the right focus on
deployment automation as its own essential development effort. This approach requires that DevOps engineers verify the DevOps process itself.

**DevOps verifies the DevOps process itself**

Whether provisioning a server or deploying an application, the DevOps effort must be treated to be as a development lifecycle, with the goal of creating the automated deployment pipeline. Many DevOps practitioners approach this task using agile practices, improving the deployment process itself in an iterative way.

In fact, many early DevOps enthusiasts referred to these practices as agile systems administration, a phrase that is both illustrative and appropriate, although many of us have used these methodologies to support waterfall development efforts, as well. Regardless of whether your organization is using agile, waterfall or a hybrid agile-waterfall approach, software methodology is fundamental.

**Software methodology in practice**

Application Lifecycle Management (ALM) defines the tasks and processes employed by all of the stakeholders involved with successfully implementing any software or systems development effort. The well-defined ALM, automated by use of a workflow tool, helps to provide the essential clarity necessary so that each stakeholder understands the tasks for which they are responsible and to provide transparency by facilitating communication.

**Conclusion**

Developers focus on creating new functions. Operations teams focus on providing reliable systems. DevOps engineers provide the principles, practices and hands-on procedures to develop software that has quality built in from the very beginning of the software and systems and delivery lifecycle. These practices align well with well-respected best practices as described in industry standards and frameworks. Creating reliable systems requires the very practices and principles that are emerging as part of the DevOps revolution.
Related topics

- Agile Manifesto
- The Software Project Manager's Bridge to Agility. Michell Sliger and Stacia Broderick. (Addison-Wesley, 2008)
- User Stories Applied for Agile Software Development. Mike Cohn. (Addison-Wesley, 2004)