DevOps in Practice

Reliable and automated software delivery

Code Crushing

DANILO SATO

Code Crushing
Books and programming
Rua Vergueiro, 3185 - 8º andar
04101-300 – Vila Mariana – São Paulo – SP – Brasil

Dedication

“To my dad, who introduced me to the world of computers and is my life-long role model."

Preface

Jez Humble

Shortly after I graduated from university in 1999, I got a job at a start-up in London. My boss, Jonny LeRoy, taught me the practice of continuous deployment: When we were finished with a new feature, we would do a quick manual smoke test on our workstation and then ftp the relevant ASP scripts directly onto the production server – not a practice I would recommend today, but it did have the advantage of enabling us to get new ideas to our users quickly.

In 2004 I joined ThoughtWorks where my job was to help enterprises deliver software, and I was appalled to discover that lead times of months or even years were common. Fortunately, I was lucky enough to work with a number of smart people in our industry who were exploring how to improve these outcomes while also increasing quality and improving our ability to serve our users. The practices we came up with also made life better for the people we were working with (for example, no more deployments outside of business hours) – an important indication that we were doing something right. In 2010, Dave Farley and I published " Continuous Delivery," in which we describe the principles and practices that make it possible to deliver small, incremental changes quickly, cheaply, and at low risk.

However, our book omits the nuts and bolts of how one actually gets started creating a deployment pipeline, put in place monitoring and infrastructure as code, and the other important, practical steps needed to implement continuous delivery. Thus I am delighted that Danilo has written the book that you have in front of you, which I think is an important and valuable contribution to the field. Danilo has been deeply involved in helping organizations implement the practices of continuous delivery for several years and has deep experience, and I am sure you will find his book practical and

informative.

I wish you all the best with your journey.

About the book

Delivering software in production is a process that has become increasingly difficult in IT department of various companies. Long testing cycles and a division between development and operations teams are some of the factors that contribute to this problem. Even Agile teams that produce releasable software at the end of each iteration are unable to get to production when they encounter these barriers.

DevOps is a cultural and professional movement that is trying to break down those barriers. Focusing on automation, collaboration, tools and knowledge sharing, DevOps is showing that developers and system engineers have much to learn from each other.

In this book, we show how to implement DevOps and Continuous Delivery practices to increase the deployment frequency in your company, while also increasing the production system's stability and reliability. You will learn how to automate the build and deployment process for a web application, how to automate infrastructure configuration, how to monitor the production system, as well as how to evolve the architecture and migrate it to the cloud, in addition to learning several tools that you can apply at work.

Acknowledgements

To my father, Marcos, for always being an example to follow and for going beyond by trying to follow the code examples without any knowledge of the subject. To my mother, Solange, and my sister, Carolina, for encouraging me and correcting several typos and grammar mistakes on preliminary versions of the book.

To my partner and best friend, Jenny, for her care and support during the many hours I spent working on the book.

To my editor, Paulo Silveira, for giving me the chance, and knowing how to encourage me at the right time in order for the book to become a reality. To my reviewer and friend, Vivian Matsui, for correcting all my grammar mistakes.

To my technical reviewers: Hugo Corbucci, Daniel Cordeiro and Carlos Vilella. Thanks for helping me find better ways to explain difficult concepts, for reviewing terminology, for questioning my technical decisions and helping me improve the overall contents of the book.

To my colleagues Prasanna Pendse, Emily Rosengren, Eldon Almeida and other members of the "Blogger's Bloc" group at ThoughtWorks, for encouraging me to write more, as well as as for providing feedback on the early chapters.

To my many other colleagues at ThoughtWorks, especially Rolf Russell, Brandon Byars and Jez Humble, who heard my thoughts on the book and helped me choose the best way to approach each subject, chapter by chapter.

Finally, to everyone who contributed directly or indirectly in writing this book.

Thank you so much!

About the author

Danilo Sato started to program as a child at a time when many people still did not have home computers. In 2000, he entered the bachelor's program in Computer Science at the University of São Paulo [USP], beginning his career as a Linux Network Administrator at IME-USP for 2 years. While at university he began working as a Java / J2EE developer and had his first contact with Agile in an Extreme Programming (XP) class.

He started his Masters at USP soon after graduation, and supervised by Professor Alfredo Goldman, he presented his dissertation in August 2007 about "Effective Use of Metrics in Agile Software Development" [15].

During his career, Danilo has been a consultant, developer, systems administrator, analyst, systems engineer, teacher, architect and coach, becoming a Lead Consultant at ThoughtWorks in 2008, where he worked in Ruby, Python and Java projects in Brazil, USA and United Kingdom. Currently, Danilo has been helping customers adopt DevOps and Continuous Delivery practices to reduce the time between having an idea, implementing it and running it in through production.

Danilo also has experience as a speaker at international conferences, presenting talks and workshops in: 2007/2009/2010 XP, Agile 2008/2009, Ágiles 2008, Java Connection 2007, Falando em Agile 2008, Rio On Rails 2007, PyCon Brazil 2007, SP RejectConf 2007, Rails Summit Latin America 2008, Agile Brazil 2011/2012/2013, QCon SP 2011/2013, QCon Rio 2014, RubyConf Brazil 2012/2013. He was also the founder of the Coding Dojo @ São Paulo and an organizer for Agile Brazil 2010, 2011 and 2012.

Contents

Chapter 1

Introduction

With the advancement of technology, software has become an essential part of everyday life for most companies. When planning a family vacation — scheduling hotel rooms, buying airplane tickets, shopping, sending an SMS or sharing photos of a trip — people interact with a variety of software systems. When these systems are down, it creates a problem not only for the company that is losing business, but also for the users who fail to accomplish their goals. For this reason, it is important to invest in quality software and stability from the moment that the first line of code is written until the moment it starts running.

1.1 Traditional approach

Software development methodologies have evolved, but the process of transforming ideas into code still involves several activities such as requirements

gathering, design, architecture, implementation and testing. Agile software development methods have emerged in the late 90s, proposing a new approach to organize such activities. Rather than performing them in distinct phases - the process known as waterfall - they happen at the same time, in short iterations. At the end of each iteration, the software becomes more and more useful, with new features and less bugs, and the team decides with the customer what should be the next slice that is developed.

As soon as the customer decides that the software is ready to go live and the code is released to production, the real users can start using the system. At this time, several other concerns become relevant: support, monitoring, security, availability, performance, usability, among others. When the software is in production, the priority is to keep it running stably. In cases of failure or disaster, the team needs to be prepared to react quickly to solve the problem.

Due to the nature of these activities, many IT departments have a clear separation of responsibilities between the **development team** and the **operations team.** The development team is responsible for creating new products and applications, adding features or fixing bugs, while the operations team is responsible for taking care of these products and applications in production. The development team is encouraged to introduce changes, while the operations team is responsible for keeping things stable.

At first glance, this division of responsibilities seems to make sense. Each team has different goals and different ways of working. While the development team works in iterations, the operations team needs to react instantly when something goes wrong. Furthermore, the tools and knowledge necessary to work in these teams are different. The development team evolves the system by introducing changes. On the other hand, the operations team avoids changes because they bring a certain risk to the stability of the system. This creates a conflict of interest between these two teams.

Once the conflict exists, the most common way to manage this relationship is by creating processes that define the method of working as well as the responsibilities of each team. From time to time, the development team packages the software that needs to go to production, writes some documentation explaining how to configure the system, how to install it in production, and then transfers the responsibility to the operations team. It is common to use

ticket tracking systems to manage the communication between the teams and defining service-level agreements (SLAs) to ensure that the tickets are processed and closed in a timely fashion. This hand-off often creates a bottleneck in the process of taking code from development and testing to production. It is common to call this process a **deployment** to production, or simply a production **deploy**.

Over time, the process tends to become more and more bureaucratic, decreasing the frequency of deploys. With that, the number of changes introduced in each deploy tends to accumulate, also increasing the risk of each deploy and creating the vicious cycle shown in figure 1.1.

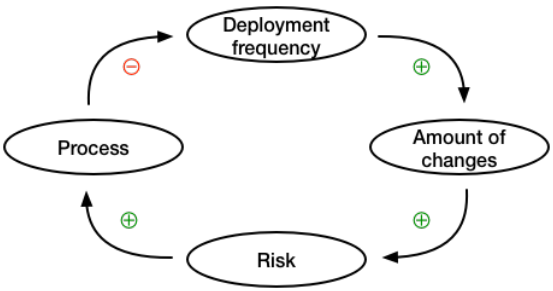

Figure 1.1: Vicious cycle between development and operations

This vicious cycle not only decreases the ability of the company to respond quickly to changes in business, but also impacts earlier stages of the development process. The separation between development and operation teams, the hand-off of code between them and the ceremony involved in the deployment process, end up creating a problem known as **"the Last Mile"** [17].

The last mile refers to the final stage of the development process that takes place after the software meet all its functional requirements but before being deployed into production. It involves several activities to verify whether the software that will be delivered is stable or not, such as: integration testing, system testing, performance testing, security testing, user acceptance testing (UAT), usability testing, smoke testing, data migration, etc.

It is easy to ignore the last mile when the team is producing and showcasing new features every one or two weeks. However, there are few teams

that are actually deploying to production at the end of each iteration. From the business point of view, the company will only have a return of investment when the software is actually running in production. The last mile problem is only visible when taking a holistic view of the process. To solve it we must look past the barriers between the different teams involved (the business team, the development team or the operations team).

1.2 An alternative approach: DevOps and Continuous Delivery

Many successful internet businesses — such as Google, Amazon, Netflix, Flickr, Facebook and GitHub — realized that technology can be used in their favor and that delaying a production deploy means delaying their ability to compete and adapt to changes in the market. It is common for them to perform dozens or even hundreds of deploys per day!

The line of thinking that attempts to decrease the time between the creation of an idea and its implementation in production is also known as **"Continuous Delivery"** [11], and is revolutionizing the process of developing and delivering software.

When the deployment process ceases to be a ceremony and starts becoming commonplace, the vicious cycle of figure 1.1 gets completely reversed. Increasing deployment frequency causes the amount of change in each deploy to decrease, also reducing the risk associated with that deploy. This benefit is not something intuitive, but when something goes wrong it is much easier to find out what happened because the amount of changes that may have caused the problem is smaller.

However, reducing the risk does not imply a complete removal of the processes between development and operation teams. The key factor that allows the reversal of the cycle is process automation, as shown in figure 1.2. Automating the deployment process allows it to run reliably at any time, removing the risk of problems caused by human error.

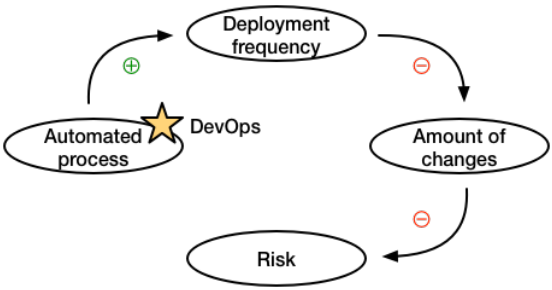

Figure 1.2: DevOps practices help to break the vicious cycle through process automation

Investing in automation is not a new idea; many teams already write automated tests as part of their software development process. Practices such as test-driven development (TDD) [4] or continuous integration [13] — which will be discussed in more detail in chapter 6 — are common and widely accepted in the development community. This focus on test automation, along with the creation of multidisciplinary teams, helped to break the barrier between developers, testers and business analysts, creating a culture of collaboration between people with complimentary skills working as part of the same team.

Inspired by the success of Agile methods, a new movement emerged to take the same line of reasoning to the next level: the **DevOps** movement. Its goal is to create a culture of collaboration between development and operation teams that can increase the flow of completed work — higher frequency of deploys — while increasing the stability and reliability of the production environment.

Besides being a cultural change, the DevOps movement focuses a lot more on practical automation of the various activities necessary to tackle the last mile and deliver quality code to production, such as: code compilation, automated testing, packaging, creating environments for testing or production, infrastructure configuration, data migration, monitoring, log and metrics aggregation, auditing, security, performance, deployment, among others.

Companies that have implemented these DevOps practices successfully no longer see the IT department as a bottleneck but as an enabler to the busi-

ness. They can adapt to market changes quickly and perform several deploys per day safely. Some of them even make a new developer conduct a deploy to production on their first day of work!

This book will present, through actual examples, the main practices of DevOps and Continuous Delivery to allow you to replicate the same success in your company. The main objective of the book is to bring together the development and operations communities. Developers will learn about the concerns and practices involved in operating and maintaining stable systems in production, while system engineers and administrators will learn how to introduce changes in a safe and incremental way by leveraging automation.

1.3 About the book

The main objective of the book is to show how to apply DevOps and Continuous Delivery concepts and techniques in practice. For this reason, we had to choose which technologies and tools to use. Our preference was to use open source languages and tools, and to prioritize those that are used widely in industry.

You will need to use the chosen tools to follow the code examples. However, whenever a new tool is introduced, we will briefly discuss other alternatives, so that you can find the option that makes the most sense in your context.

You do not need any specific prior knowledge to follow the examples. If you have experience in the Java or Ruby ecosystem, that will be a bonus. Our production environment will run on UNIX (Linux, to be more specific), so a little experience using the command line can help but is not mandatory [12]. Either way, you will be able to run all the examples on your own machine, regardless if you are running Linux, Mac or Windows.

Target audience

This book is written for developers, system engineers, system administrators, architects, project managers and anyone with technical knowledge who has an interest in learning more about DevOps and Continuous Delivery practices.

Chapter structure

The book was written to be read from beginning to end sequentially. Depending on your experience with the topic of each chapter, you may prefer to skip a chapter or follow a different order.

Chapter 2 presents the sample application that will be used through the rest of the book and its technology stack. As the book's focus is not on the development of the application itself, we use a nontrivial application written in Java built on top of common libraries in the Java ecosystem. At the end of the chapter, the application will be running in production.

With the production environment running, in **chapter 3** we wear the operations team's hat and configure a monitoring server to detect failures and send notifications whenever a problem is encountered. At the end of the chapter, we will be notified that one of the servers crashed.

In **chapter 4** we will rebuild the problematic server, this time using automation and treating infrastructure as code. **Chapter 5** is a continuation of the subject, covering more advanced topics and refactoring the code to make it more modular, readable and extensible.

After wearing the operations team's hat, we will turn our attention to the software development side. **Chapter 6** discusses the Agile engineering practices that help writing quality code. You will learn about the various types of automated tests and launch a new server dedicated to perform continuous integration of our application code.

Chapter 7 introduces the concept of a deployment pipeline. We setup continuous integration for the infrastructure code, and implement an automated process to deploy the newest version of the application in production with the click of a button.

In **chapter 8**, we migrate the production environment to the cloud. Finally, we discuss more advanced topics, including resources for you to research and learn more about the topics that were left out of the scope of this book.

Code conventions

In the code examples, we will use ellipsis `...` to omit the unimportant parts. When making changes to already existing code, we will repeat the

lines around the area that needs to be changed to give more context about the change.

When the line is too long and does not fit on the page, we will use a backslash \ to indicate that the next line in the book is a continuation of the previous line. You can simply ignore the backslash and continue typing in the same line.

In the command line examples, in addition to the backslash, we use a greater than signal > in the next line to indicate that it is still part of the same command. This is to distinguish between the command that is being entered and the output produced when the command executes. When you run those commands, you can simply type it all in one line, ignoring the \ and the >.

More resources

We have created a discussion forum on Google Groups where you can send questions, suggestions, or feedback directly to the author. To subscribe, visit the URL:

https://groups.google.com/d/forum/devops-in-practice-book

All code examples from the book are also available on the author's GitHub projets, in these URLs:

https://github.com/dtsato/loja-virtual-devops

https://github.com/dtsato/loja-virtual-devops-puppet