DevOps Toolchain Required by Efficient Software Development Teams

Ruiyang Ding

Delft University of Technology

DevOps Toolchain Required by Efficient Software Development Teams

Master's Thesis in Computer Science

Parallel and Distributed Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Ruiyang Ding

25th March 2020

Author
Ruiyang Ding

Title
DevOps Tool-chain Required by Efficient Software Development Teams

MSc presentation
TODO GRADUATION DATE

Graduation Committee
TODO GRADUATION COMMITTEE Delft University of Technology

Abstract

TODO

Preface

TODO MOTIVATION FOR RESEARCH TOPIC

TODO ACKNOWLEDGEMENTS

Ruiyang Ding

Delft, The Netherlands
25th March 2020

Contents

Chapter 1

Introduction

1.1 Problem Statement

DevOps is a concept which emerged in recent years. The term ”DevOps” is created by Patrick Debois in 2009, after he saw the presentation ”10 deployments per day” by John Allspaw and Paul Hammond.[21] DevOps is the combination of practices, and culture which aims to combine separate departments(software development, quality assurance and the operation and others) in the same team, in order fasten the software delivery without risking high software quality.[5][17]

For a software team which will employ DevOps, the team must first understand what are the DevOps practices needed. This problem leads to the RQ1 of this paper. Usually, the practices that the team need to adopt help to employ DevOps includes: automated testing and deployment, monitoring, team-working and continues integration etc.[20][25].

DevOps strongly rely on tools. There are specialised tools exist for helping teams adopt different DevOps practices[25]. There are different categories of tools used for different parts of the DevOps practice. For example, Jenkins for continuous integration, Ansible for process automation, sonarqube for code analysis, Slack for team communications etc. It could be interesting if we could investigate what kind of tools does it exist, and how those tools could help teams in DevOps different practice. The research could include how these tools could combine as well. This leads to RQ2.

In software engineering, the toolchain is a set of tools which combined for performing a specific objective. Thus DevOps toolchain is the integration between tools that specialised in different aspect of the ecosystem, which support and coordinate the DevOps practices. The DevOps toolchain could assistant business in creating and maintain an efficient software delivery pipeline, simplify the task and further achieve DevOps.[7][9]

The deployment of toolchain could be on-premise, which means the software development team does the deployment and integration between the tools by themselves. This could: first, provides the maximum freedom for the team to choose the

tools, and secondly, allow team customise the toolchain according to their needs. However, the downside is that the need for extra time for deployment and maintenance. Besides, the cost is hard to calculate and control.

With the development of cloud technologies, now some vendors starting provides the DevOps toolchain under a single application with the concept of Software as a Service(SaaS). A good example is GitLab CI [1]. GitLab CI provides a complete set of tools which covers the whole lifecycle. The toolchain is delivered as a single platform that allows development teams to start using DevOps toolchain without the pain of having to choose, integrate, learn, and maintain a multitude of tools. [6]

So the RQ3 of this thesis project focuses on the comparison between these 2 kinds of toolchains. We will build a DevOps toolchain with the popular tools used in the industry, the deployment and integration of the toolchain will be on-promise. We will also try to justify why do we select a certain tool for each section. This self-built toolchain will be used to compare with commercial singe application DevOps toolchain. We will pick the most popular singe application DevOps toolchain for comparison. In the comparison, we will simulate the same DevOps lifecycle of a demo Spring Boot web app in these 2 toolchains. The perspective of comparison between these toolchains will include:

- Development time: The time spend for implements the toolchain and set up the whole DevOps pipeline.
- Cost structure: The total cost for using the toolchain. For self-built toolchain, it will also include the cost decomposition (for different tools).
- Flexibility: How much freedom can you add/change tools in the toolchain.
- Scalability: How easy to scale the toolchain for the larger project.
- Performance: The performance of the Continues Delivery pipeline, for the same task, how long will it take for the whole process?

From this part of the study, we could make a full comparison between on-premise toolchain and the single application toolchain. For software development teams, it could provide better insights into how to select the DevOps toolchains.

1.2 Research Objectives and Questions

- RQ1: What kind of practices a software development team need to employ DevOps.
- RQ2: What kind of DevOps tools could be included in the toolchain that would help the team implements the practices mentioned in RQ1.
- RQ3: How does the single platform toolchain compared with the on-premise toolchain we build?

[1] https://about.gitlab.com/stages-DevOps-lifecycle/

1.3 Thesis Structure and Main Contributions

In Chapter 2, we will introduce concepts within the scope of DevOps. We will also include the concepts in cloud computing which is related to our research. Chapter 3 is focuses on the literature analysis of DevOps practices. Chapter 4 is focuses on the tools which helping apply the DevOps practices. Chapter 5 focuses on the implementation of DevOps toolchains and the experimentation for comparison between 2 kinds of toolchains. We will summarize our research on the Chapter 6.

The main contributions of this paper are:

- We conduct a literature research on the practices that are necessary for a team to employ DevOps(Chapter 3). We provide a study on the DevOps tools and their mapping to different DevOps practices. This part of research could help the software team which is going to employ DevOps understand the practices needed. Besides, the research gives them a clearer scope on the tools needed for implementing the practices(Chapter 4).

- We give the overview on 2 different types of DevOps toolchain. We also implements demo prototypes for each type of the toolchain, and conduct experiments with these prototypes. The experiment result shows the comparison between different toolchains. It could help team understand which toolchain cloud be selected according to the needs(Chapter 5).

Chapter 2

Background and Concepts

In this chapter, we will introduce several main concepts related to our study.

2.1 Agile software development

The term ”Agile” represents the fast adaptation and response to the changes.[10] Agile software development is a new method of software development that implements the ideology of ”agile”. Agile software development advocates the continuous development of software teams. The software development under this methodology will have shorter planning/development time before it delivers to the costumers and could better adapt to changes in the environment and requirements.

Agile software development uses an iterative way in the development process. The traditional software development process, like the waterfall method, requires the long and complicated planning process, and a complicated document. Once one phase of the development is done, the teams shouldn't change the output (document and code) of this phase.[16] In contrast, the agile software development aims to satisfy the customer with early and continuous delivery of the software.[14]. Early means the shorter time before software delivery. Continuous means the development does not end with the delivery. Delivery means the end an iteration, together with a demonstration to stakeholders. After delivery, the team continue to next iteration according to the feedback it gets from stakeholders. In each iteration, the team not aims to add major features to the software, rather their goal is to have a working and deliverable release[13]. In the ideology of agile, the best design the software product comes from the iterative development, rather than the tedious planning.[14]

The rapid development doesn't mean low software development quality. On the contrast, the quality of software design is highly appreciated in the agile software development. The automatic testing is widely used in Agile. The test cases will be defined and implements from the beginning of the development process. The testing goes through the whole development iteration ensure the software has a high enough quality to be released or demonstrate to costumers at any point of an

iteration[1].

The agile software development processes include collaboration across different groups, ie. bushiness development team, software development team, test team, and costumers. It values more face to face communications[15] and feedbacks. The goal for these communications is, firstly to let everyone in the multifunctional agile team understand the whole project, secondly, to receive feedback that helps the software in the right development track that aligns with the requirement of the stakeholders. [14]

According to the Manifesto for Agile Software Development, compared with traditional software development, the agile software development value these aspects: [14]

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

2.2 CI/CD

In the software development, CI/CD refers to continuous integration, continuous delivery and continuous deployment.[4]. As we mentioned in 2.1, agile software development requires continuous software quality assurance and iterative development. Currently, CI/CD is one set of the necessary practices for the team to become agile by achieving the requirements above. Figure 2.1 shows the relationship between these 3 practices.

2.2.1 Continuous Integration

Continuous interaction is the base practice of all practices within CI/CD, and continuous delivery/deployment is based on the continuous interaction.[4] The continuous integration means the team integrate each team member's work into main codebase frequently(multiple times per day). "Integrate" means merge the code to the main codebase.[19]. The continuous interaction rely on 2 practices: *Build Automation* and *Test Automation*. The definition of these 2 practices are:

- *Test Automation:* Test automation means using separate software to execute the software automated, without human intervention. It could help the team to test fast and test early. [8]
- *Build Automation:* Automate the process of creating software build. This means to automate the dependency configuration, source code compiling, packaging and testing. It is viewed as the first step to continuous integration[2]

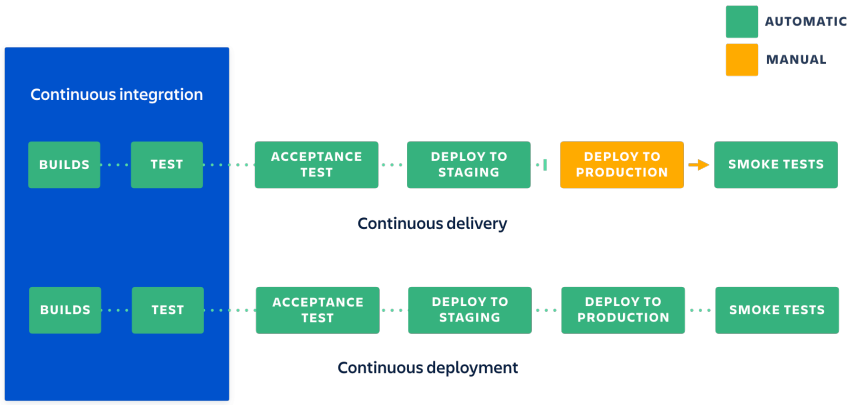

Figure 2.1: The relationship between continuous integration, continuous delivery and continuous deployment

With the help of these 2 practices, for each developer in the team, the workflow in continuous interaction as follows:[19] In the development of each feature, the developer first pull the code from the main codebase. During the development, new test cases could also be added to the automated test. After the development is done, automated testing also runs on the code to maintain the code quality and minimize the number of bugs from the beginning. The build automation compiled the code locally in the development machine.

After the step above, the developer already has the executable and the high quality (passed the automated test) code in the development machine before submitting the change to the code base. This represents the principle of quality and automation in agile software development. In the next step, the developer commits changes to the repository, which is the main codebase, and the system check the conflict and do the test/build again, to make sure that there are not any bugs missed in the test on the development machine. If the code passes this build and test, it will be merged to the main codebase and the integration is done.

2.2.2 Continuous Delivery and Continuous Deployment

Continuous delivery is practices that software development team build a software that can be released at any time of the lifecycle.[18]This means the software always maintains a high quality and in a deployable state.[11] It is a subset of agile, which focuses on the software delivery.[3] From the last section, we introduce the concept of continuous interaction. The continuous delivery is based on continuous interaction but further automate the software deployment pipeline. In the software deployment pipeline, the team divide build into several stages, first build the

product and then push the product into the production-like environment for further testing. This ensures that the software could be pushed to production at any time. However, in continuous delivery, the deployment of software into production is done manually. The benefit of continuous delivery includes:[11][18]

- High code quality: The automate and continuous testing ensure the quality of the software.
- Low risk: The software could be related at any time, and it's easier to release and harder to make the mistake
- Short time before going to the market: The iteration of software development is much shorter. The automation in testing, deployment, environment confirmation included in the process, and the always read-to-deploy status shorten the time from development to market.

The continuous deployment is based on continuous delivery. The only difference is continuous deployment automates the deployment process. In continuous delivery, the software is deployable but not deploy without manual approval. In the continuous deployment, each change that passed automated build and testing will be deployed directly. The continuous deployment is a relatively new concept that most company not yet put the practice into production.[23] While continuous delivery is the required practice for the company to be DevOps and it is already being widely used.

2.3 DevOps

> The fundamental goal of DevOps is to minimize the service overhead so that it can response to change with minimal effort and deliver maximum amount of value during its lifetime.
>
> – Markus Suonto, Senior DevOps Consultant, Eficode

2.3.1 Definition

DevOps is a set of practices that aims to combine different, traditionally separated disciplines (eg. software development, operations, QA, and others) in cross-functional teams with the help of automation of work in order to speed up software delivery without risking high quality.[12]

DevOps is the extension and evolution[24][22] of Agile. DevOps and Agile both driven by the collaboration ideology and the adoption of DevOps needs Agile as the key factor.[24] In the pre-DevOps era, the Development and Operation are two different teams with difference goal. The interface between them is based on the ticket system which the operation team do the ticket management. As we mentioned at 2.1, the goal of Agile is to shorten the deliver life cycle and delivery

software quickly to the costumers. So when practice agile development method under this scenario, the development try to to deliver the code they develop earlier but the operation team usually will delay the process for quality control or other reasons. In practice, this causes the delay between the code change and the software delivery to the costumers[22].

The lack of communication and conflict between developers and operation team slow down the software delivery process and also make it harder for the teams to be real Agile. Thus the concept ”DevOps” is being proposed at 2008, for eliminating of the boundary between developers (Dev) and operation team (Ops) and enhance the collaboration within an organization.

2.3.2 DevOps Practices

Chapter 3

Conclusions and Future Work

3.1 Conclusions

TODO CONCLUSIONS

3.2 Future Work

TODO FUTURE WORK

Bibliography

[1] Agile software development - wikipedia. https://en.wikipedia.org/wiki/Agile_software_development#Iterative,_incremental_and_evolutionary. (Accessed on 03/18/2020).

[2] Build automation - wikipedia. https://en.wikipedia.org/wiki/Build_automation. (Accessed on 03/19/2020).

[3] Continuous delivery vs. traditional agile - dzone devops. https://dzone.com/articles/continuous-delivery-vs. (Accessed on 03/24/2020).

[4] Continuous integration vs. continuous delivery vs. continuous deployment. https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment. (Accessed on 03/19/2020).

[5] Devops - wikipedia. https://en.wikipedia.org/wiki/DevOps. (Accessed on 02/24/2020).

[6] The devops lifecycle with gitlab — gitlab. https://about.gitlab.com/stages-devops-lifecycle/. (Accessed on 03/11/2020).

[7] Devops toolchain - wikipedia. https://en.wikipedia.org/wiki/DevOps_toolchain. (Accessed on 03/11/2020).

[8] Test automation in a ci/cd pipeline — spritecloud. https://www.spritecloud.com/test-automation-with-ci-cd-pipeline/. (Accessed on 03/19/2020).

[9] Toolchain - wikipedia. https://en.wikipedia.org/wiki/Toolchain. (Accessed on 03/11/2020).

[10] What is agile software development? — agile alliance. https://www.agilealliance.org/agile101/. (Accessed on 03/18/2020).

[11] What is continuous delivery? - continuous delivery. https://continuousdelivery.com/. (Accessed on 03/23/2020).

[12] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: A software architect's perspective*. Addison-Wesley Professional, 2015.

[13] Kent Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999.

[14] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001.

[15] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Principles behind the agile manifesto. *Agile Alliance*, pages 1–2, 2001.

[16] Michael A Cusumano and Stanley A Smith. Beyond the waterfall: Software development at microsoft. 1995.

[17] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. Devops. *Ieee Software*, 33(3):94–100, 2016.
[18] M Fowler. Continuous delivery. may 30, 2013, 2013.
[19] Martin Fowler and Matthew Foemmel. Continuous integration, 2006.
[20] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. What is devops? a systematic mapping study on definitions and practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016*, pages 1–11, 2016.
[21] Gene Kim, Jez Humble, Patrick Debois, and John Willis. *The DevOps Handbook:: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, 2016.
[22] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A survey of devops concepts and challenges. *ACM Computing Surveys (CSUR)*, 52(6):1–35, 2019.
[23] Marko Leppänen, Simo Mäkinen, Max Pagels, Veli-Pekka Eloranta, Juha Itkonen, Mika V Mäntylä, and Tomi Männistö. The highways and country roads to continuous deployment. *Ieee software*, 32(2):64–72, 2015.
[24] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. Relationship of devops to agile, lean and continuous deployment. In *International conference on product-focused software process improvement*, pages 399–415. Springer, 2016.
[25] Liming Zhu, Len Bass, and George Champlin-Scharff. Devops and its practices. *IEEE Software*, 33(3):32–34, 2016.