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

8th May 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
8th May 2020

Contents

Chapter 1

Introduction

The Agile Manifesto drafted by Kent Beck etc. in 2001 created the Agile software development methods[19]. Since then, this new software development methods have draw attention of the industry and more and more companies started to apply Agile in the production. The Agile method advocates the shorter development iteration, continuous development of software and continuous delivery of the software to the customer. The goal of Agile is to satisfies customer with early and continuous delivery of the software.[19]

The Agile, which aims at the improvement of the process within the software development team and the communication between the development team and costumers [34] do makes the software development faster. However, it doesn't emphasis the cooperation and communication between the development team and other teams. In real life, the conflict and lack of communication between the development team and operation teams usually become the barrier for shortening the delivery time of the software project.

Thus, the concept of DevOps 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.[29] While Agile fills the gap between software development and business requirement from the customer, the DevOps eliminates the gap between the development team and the operation team. [13] By eliminates the barrier we mentioned in the last paragraph, DevOps further fasten software delivery. In conclusion, DevOps means a 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, maximizing delivered without risking high software quality. [5][22]

Helping the customer do the DevOps transformation is one of the main business activities of Eficode, the company which I'm writing my thesis. The DevOps toolchain is important for the company to help it's costumers to employ DevOps practices. For Eficode, It's important to help costumers select the toolchain tha has low cost, short implement time and good performance. The repaid development of

the cloud and container technologies brings new ways for us to develop and deploy the tools and software. This new changes brought by cloud may further improving the performance, lower the cost of DevOps toolchain development – both in money and time. As part of thesis work at Eficode, I investigated how could cloud technologies helps improve the DevOps toolchain.

1.1 Problem Statement

As the title says, the paper focuses on the toolchain in the DevOps. 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][11] On the other hand, DevOps is strongly rely on tools. There are specialised tools exist for helping teams adopt different DevOps practices[37].

At the same period that the tools for DevOps emerged and developed, the cloud technologies also developed rapidly. Compared with traditional on premise services, the new cloud services brings brand new ways for the implementation and deployment of software. Here are few examples: Software as a Service(SaaS) allow vendors provides the DevOps toolchain under a single application. 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. Serverless computing such as AWS Lambda and Google Cloud Function allows application to be divided by functions and designed under event-driving paradigm. The Serverless computing cloud could be used to deploy certain component of a DevOps toolchain to ease the implementation difficulties and reduce the cost. The managed scalable container services in cloud could help the toolchain become more scalable. Compared with traditional virtual machine, the container allow applications to be faster deployed, patched and scaled.[12]

As we can see from examples, the DevOps toolchain could leverage the capability of cloud services and benefit from it. So as part of the research questions we will investigate how does cloud services benefits the DevOps toolchain.

In introduction above we mentioned the SaaS DevOps toolchain. Or according some vendors(for example GitLab), this kind of toolchain is being called ”DevOps Platform”. This new emerged type of toolchain leave a question to the development team: which kind of toolchain should they select. The SaaS toolchain provides a out-of-box integrated solution for the whole DevOps lifecycle, which is tempting, but apart from the advertisement from the vendors of these ”DevOps” platforms, It is still unclear if it is real better than the traditional self-built toolchain. So

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

the another part of our research is to compare these two kinds of toolchains on different DevOps metrics. This could give the software development team a better knowledge on which kind of toolchain suites them the best.

Based on above, the research questions could be summarised as below:

1. **RQ1:** How DevOps toolchain make use of current cloud services and how these services improves the DevOps toolchain.

2. **RQ2:** How does the newly emerged SaaS single platform toolchain compared with the toolchain we build?

1.2 Research Method

To answer the RQ 1 we will first build a DevOps toolchain with the popular tools used in the industry. We will also try to justify why do we select a certain tool for each component. The toolchain will be deployed on Amazon Web Services (AWS) which is the cloud services used by Eficode. AWS contains many new cloud services for example AWS Lambda for serverless computing and AWS EKS for managed Kubernetes services. So In the process of develop and deploy the toolchain, we will answer the RQ by research how could cloud services in AWS benefit our toolchain. We will tye to investigate from different perspectives includes cost, performance and development difficulties. We will also have the demo implementations which shows the answer to this research question.

To answer RQ2, 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 on both toolchains. From this part of the study, we could make a full comparison between traditional toolchain and the new emerged SaaS single application toolchain. For software development teams, it could provide better insights on how to select the DevOps toolchains.

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 focusing on a literature survey on cloud services which the DevOps toolchain could make use of. Chapter 4 focuses on the designed and the implementation of the experiment. The experiments shows how does the cloud services introduced in CH3 cloud services could benefit DevOps toolchain. Chapter 5 will introduce you the experimentation that comparing 2 kinds of toolchains we mentioned earlier. We will finally summarise our research in Chapter 6.

The main contributions of this paper are:

- We provide a study on how could the DevOps tools leverage the cloud services to reduced development/deployment difficulties, lower the cost and improving the performance. 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 of the tools needed for implementing the practices(Chapter 4).

- We give the overview of 2 different types of DevOps toolchain. We also implement demo prototypes for each type of toolchain and conduct experiments with these prototypes. The experiment result shows a comparison between different toolchains. It could help the team understand which toolchain cloud be selected based on 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.[14] 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.[21] In contrast, the agile software development aims to satisfy the customer with early and continuous delivery of the software.[19]. 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[18]. In the ideology of agile, the best design the software product comes from the iterative development, rather than the tedious planning.[19]

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[20] 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. [19]

According to the Manifesto for Agile Software Development, compared with traditional software development, the agile software development value these aspects: [19]

- 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.[26]. 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. [9]
- *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]

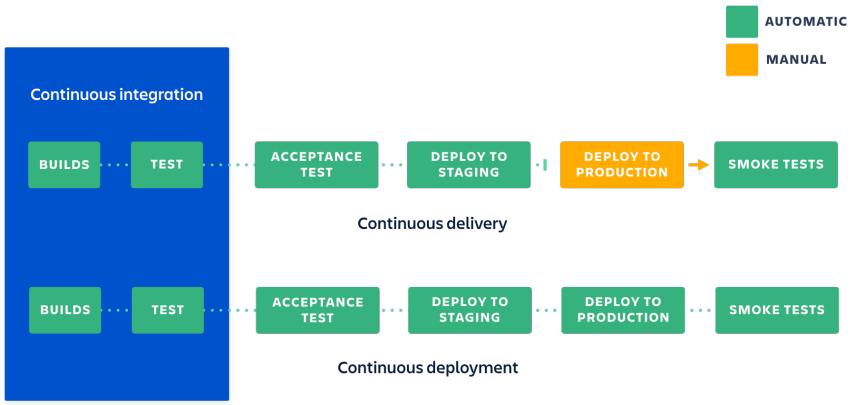

Figure 2.1: The relationship between continuous integration, continuous delivery and continuous deployment

With the help of these 2 practices, for each developer in the team, the workflow in continuous interaction as follows:[26] 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.[25]This means the software always maintains a high quality and in a deployable state.[15] 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:[15][25]

- 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.[31] 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 respond to change with minimal effort and deliver the 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 to speed up software delivery without risking high-quality.[17]

DevOps is the extension and evolution[33][30] of Agile. DevOps and Agile both driven by the collaboration ideology and the adoption of DevOps needs Agile as the key factor.[33] DevOps has a different focus on agile. DevOps focus on the delivery while agile is focused on the development with the requirement and customer. Figure 2.2[6] shows the workflow and practices of a team working under DevOps.

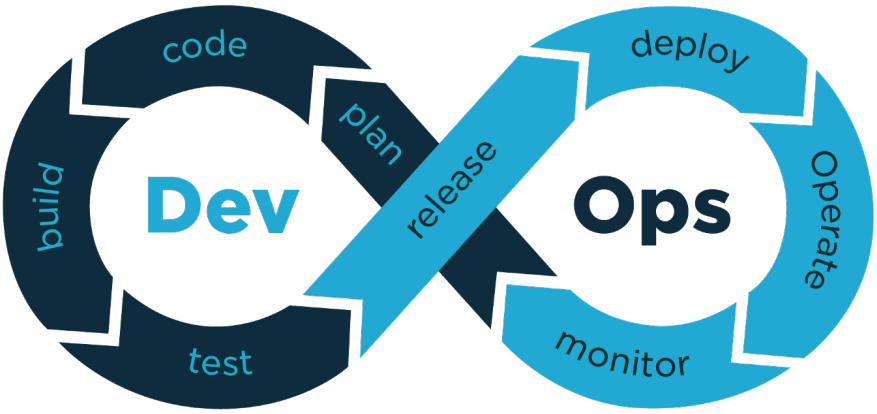

Figure 2.2: DevOps Practices and Workflow

2.3.2 Elements

In this section, we will introduce the necessary elements that an organization need to includes when employing DevOps. 4 necessary elements need to be considered.

Culture

In the pre-DevOps era, the Development and Operation are two different teams with a different 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 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[30]. The lack of communication and conflict between developers and the operation team slow down the software delivery process and also make it harder for the teams to be real Agile. Therefore the concept ”DevOps” is being proposed at 2008, for eliminating of the boundary between developers (Dev) and operation team (Ops). According to Walls (2013), this is being done by promoting the culture with 4 characterises: open communication, incentive and responsibility alignment, respect and trust.[36]

The open communication means openly discussion and debate. As mentioned above, the traditional communication method is through a very formal and regularized ticket system. In the DevOps, the communication is not limited within the formal ticket system. instead, the team will keep in the whole lifecycle of a product, from the requirement, schedule, and anything else. [36] The information sharing is also important.[32] The metrics and the project status is available for

everyone in the team , so each member could have a clear scope about what the team is doing.

The incentive and responsibility alignment mean the whole teams (combines Dev and Ops) shares the same goals and also takes the same responsibility. The shift from ”Dev” and ”Ops” to DevOps requires people who used charges in only development and operation starting sharing the responsibility from both side.[32] This means individuals or a certain part of the team will be not solely blamed if the product is failed. This ”no blame” culture could help each engineer be willing to take the development responsibility for the whole system.[24]

Respect means all employees should respect and recognize the contribution of other teams members. A DevOps team is not a single team without any division of jobs, there is still an operation part within a team. [10] However the people operation team will take development responsibility, and the developers will also put their hands-on operation and management.[35] To make people with different roles works in a team, trust and respect each other is critically important.

Organisation

In the organizational level, the DevOps emphasizes the collaboration between different part of an organization. This is strongly correlated with the ”culture” part of this section. Inside a team, each member should be a generalist who could understand all aspect of a project. There will not be a dedicate QA, operation or security team within a team. Instead, these are the job that belongs to everyone[24][29] The organization should provide the team member with opportunities to learn all skill needed for building the whole system.

The team size should be small. A small team could help to reduce the inter-team communication. The small team means the scope of the project is small. And it also means less bureaucracy in team management. There are four benefits to have a small team:[29]

- The smaller team allows each team member to easily understand the whole project.
- The smaller team could reduce the amount of communication needed. It could also limit the growth rate that the product could have.
- The smaller team could decentralize power. In DevOps, each team lead could define the metrics which become the overall criteria of the whole team’s performance.[29]
- In a smaller team, failure doesn’t mean a disaster for the company. This allows the team to fail. Thus each employee could train their headship skill in the team without too much pressure.

Furthermore, another important organizational aspect for DevOps is to have a loosely-coupled architecture. The first benefit of this is the better safety. In the organisation with a tightly-coupled architecture, small changes could result in large

failure.[29] The second benefit is productive. In a traditional organisation, the result of each team will be merged, tested together and deploy together. This means it is time-costly to configure and manage the test environment requires dependencies. A loose organisation enable each team to finish the development of lifecycle (from planing to deployment) independently. Each team could update their products independently, which gives the team more flexibly to align the product with the change in the customer requirement. This means the update of each team's product won't affect other teams as well.

Automation

In the DevOps, automation means automation within the whole development and operation process. The organisations which employing DevOps aims for a high degree of automation.[23] With automation, people could be free from the repetitive work and reduce human error. It could help build the DevOps culture of collaboration, and it is seen as the cornerstone of the DevOps.[8] The main practices regarding Automation are the automated testing, continuous delivery and automated operation. Automated testing could be achieved by test automation. We already mentioned the benefit of this at 2.2.1.

The continuous delivery pipeline is the core of the DevOps.[27] As we discussed at 2.2.2. The continuous delivery will ultimately automate all steps between the developer to commit the code to the product in the production.

The automation of the operation part is usually done by using the concept of "Infrastructure as Code"[32]. The Infrastructure as Code (IasC) means to define everything in the software infrastructure level as code.[16] Because it is code, we could use the automation methodology used in the software development to manages and deploy these codes. According to Christof et. (2016), under IasC, infrastructure can be shared, tested, and version controlled. [22] This could help emphasizes the automation within the operation scope. With the automation in operation, the team could be free from the tedious environment configuration and shorten the product development lifecycle. Automating server configuration means the developers and operation staff can equally know the server configuration [8] which help build the culture of shared responsibility and trust.

Monitoring

Monitoring is to continuously collect the matrices from the running system for helping the team find the problems in the system. In the DevOps way of development, the testing is the key to maintain the quality of the software continuously. However, when the product enters the production, we cannot test the software any more. So, we need monitoring to keep track the status of the product.[28] Thus, Monitoring is an important component of DevOps.

With monitoring, the software team could keep tracking the status, and maintain the quality of deployed production. The monitoring has also enabled the team to

collect the data from costumers' usage behaviour. This helps the agile development team to make an improvement in the next iteration of the product. [32]

For develop a high-quality monitoring system, the development of monitoring could be in parallel with the main product, and the monitoring system can be already be used against the "staging deployment" (see Figure 2.1) at the early stage of the iteration. By this, the development team can improve the monitoring system continuously together with the main software system. The parallel development of the monitoring system and the main system helps the team to find the gap in the monitoring earlier.[28]

On the other hand, As we mentioned in the "Culture" section, the collaboration is an important part of the DevOps culture. collaboration needs the communication and information sharing between the development(Dev) and operation(Ops) team. The monitoring could be one of the channels between the Dev and Ops since it can expose the information of the whole system which helps team members to understand the system as a whole. This helps the team achieving the point we mentioned at 2.3.2 (Culture) that the project status and matrices should available to every team members.

2.3.3 DevOps Practices

Chapter 3

Literature Analysis of the Cloud Services

Chapter 4

Cloud Services That Could Help DevOps Toolchain

In the chapter we introduce the implementation of our DevOps toolchain which act as the basic environment of our experiments. Then we introduce the experiment that for answering RQ1: **How could DevOps toolchain make use of current cloud services and how these services improves the DevOps toolchain.** In section 4.1 we introduce the implementation of our testing DevOps toolchain which is according to the DevOps definitions and DevOps practices we introduced at Chapter 2. In section 4.2 we discuss the cloud services selection consideration for the experiment. In section 4.3 we introduce the implementation of our experiments. Section 4.4 focuses on the experiment result.

4.1 DevOps Toolchain Implementation

Our DevOps toolchain is used to conduct experiment that could answering 2 research questions. As we mentions above, it include DevOps elements we introduced at CH2. In this section we will first introduce the composition of our DevOps toolchain, and secondly, which elements of DevOps does each components belongs to.

// Introduce our initial design of Devops toolchain

// Point out the problem, and point out the improvement can be done in the toolchain.

4.2 Cloud Services

In this section, we will introduce several could services from CH3 that could be helpful to the DevOps toolchain. // Using services in AWS as example, Introduces how cloud services could improve. describe on services in one section

4.2.1 Managed Container Services for Distributed Builds

// Describe how AWS Fargate could Help

4.2.2 Serverless computing

// Describe how AWS lambda could Help and why do we chose it

4.2.3 ...

4.3 Setup for the Experiments

4.3.1 Managed container services

Test task and System Description

// This is just a draft

In this experiment we simulate the continues delivery process of a Spring Boot web application. From the experiments, we could verify our assumption in 4.2. The continues delivery pipeline includes following steps:

1. *Pull from version control*: Pull the most recent change from Github repository
2. *Build*: Build the application with Gradle
3. *Test*: Automate testing with JUnit integrated in Gradle
4. *Artefact store*: Push the build artifacts to Artifactory

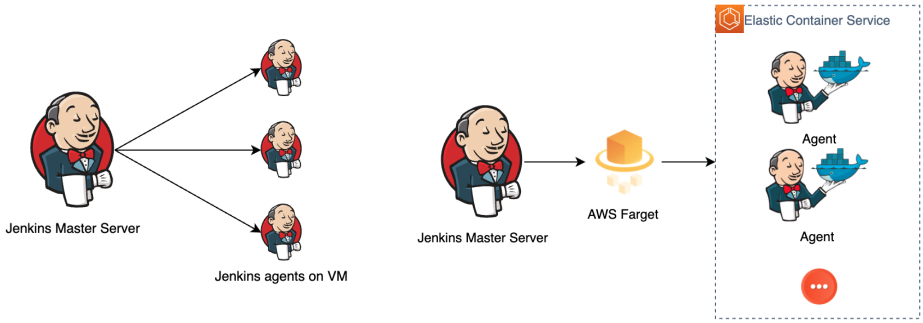

Figure 4.1: Architecture diagram of the test Jenkins cluster with agents running in traditional virtual machine (left) and on ECS with AWS Fargate (right)

To evaluate our assumption in 4.2, we have 2 different setups. The first setup Figure 4.1 (left) is a Jenkins server with traditional virtual machine as worker agents. In the second setup Figure 4.1 (right), we use the same Jenkins server but the worker nodes is dynamically provisioned in the AWS Fargate managed cloud service.

// here write some more detail about the setup which includes IAM and hardware configuration, and also include the graph which shows the topological structure of 2 setups,

Performance Properties and Evaluation

We run the pipeline through 2 different setups, we will get the result of following properties:

- *Runtime* describes the total time for finishing all the tasks.
- *Cost Structure* describes the daily cost of 2 setups under the same workload
- *Resource Utilization* describes the average CPU/RAM usage for each instance during a single run of the pipeline.

To shows how does the 2 setups performance within the teams with different sizes, we run by run different number of tasks parallel through the pipeline. This simulates the different team size, besides, it could also shows the scalability when comes to the need of task parallelization in bigger organizations.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

TODO CONCLUSIONS

5.2 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] Devops in a scaling environment - tajawal - medium. https://medium.com/tech-tajawal/devops-in-a-scaling-environment-9d5416ecb928. (Accessed on 03/27/2020).

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

[8] Devopsculture. https://martinfowler.com/bliki/DevOpsCulture.html. (Accessed on 03/27/2020).

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

[10] There's no such thing as a "devops team" - continuous delivery. https://continuousdelivery.com/2012/10/theres-no-such-thing-as-a-devops-team/. (Accessed on 03/26/2020).

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

[12] What are containers? – benefits and use cases — netapp. https://www.netapp.com/us/info/what-are-containers.aspx. (Accessed on 04/09/2020).

[13] What is a devops toolchain? – bmc blogs. https://www.bmc.com/blogs/devops-toolchain/. (Accessed on 03/13/2020).

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

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

[16] Matej Artac, Tadej Borovssak, Elisabetta Di Nitto, Michele Guerriero, and Damian Andrew Tamburri. Devops: introducing infrastructure-as-code. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 497–498. IEEE, 2017.

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

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

[19] 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.

[20] 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.

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

[22] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. Devops. *Ieee Software*, 33(3):94–100, 2016.

[23] FMA Erich, Chintan Amrit, and Maya Daneva. A qualitative study of devops usage in practice. *Journal of software: Evolution and Process*, 29(6):e1885, 2017.

[24] Dror G Feitelson, Eitan Frachtenberg, and Kent L Beck. Development and deployment at facebook. *IEEE Internet Computing*, 17(4):8–17, 2013.

[25] M Fowler. Continuous delivery. may 30, 2013, 2013.

[26] Martin Fowler and Matthew Foemmel. Continuous integration, 2006.

[27] Asif Qumer Gill, Abhishek Loumish, Isha Riyat, and Sungyoup Han. Devops for information management systems. *VINE Journal of Information and Knowledge Management Systems*, 2018.

[28] Michael Hüttermann. *DevOps for developers*. Apress, 2012.

[29] 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.

[30] 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.

[31] 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.

[32] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. Dimensions of devops. In *International conference on agile software development*, pages 212–217. Springer, 2015.

[33] 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.

[34] Marco Miglierina. Application deployment and management in the cloud. In *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 422–428. IEEE, 2014.

[35] Jordan Shropshire, Philip Menard, and Bob Sweeney. Uncertainty, personality, and attitudes toward devops. 2017.

[36] Mandi Walls. *Building a DevOps culture*. " O'Reilly Media, Inc.", 2013.

[37] Liming Zhu, Len Bass, and George Champlin-Scharff. Devops and its practices. *IEEE Software*, 33(3):32–34, 2016.