jamk.fi

An approach to Software Deployment

Continuous Integration Practices

Ivo Kostecký

Bachelor's thesis
February 2019
School of Technology, Communication and Transport
Degree Programme in Information and Communications Technology
Software Engineering

jamk.fi

Description

Author(s)	Type of publication	Date
Kostecký, Ivo	Bachelor's thesis	February 2019
		Language of publication: English
	Number of pages 55	Permission for web publication: yes

Title of publication
An Approach to Software Deployment: Continuous Integration Practices

Degree program
Information and Communications Technology, Software Engineering

Supervisor(s)
Esa Salmikangas

Assigned by
Solteq Oyj

Abstract

The thesis describes the adoption process of DevOps practices to support software development. The aim and learning objectives were applying the DevOps solutions according to the standards of industry, practically implementing them in a new environment.

The practical part consists of an implementation of the presented concepts to DevOps tool for usage in existing software product. The project was carried out in co-operation with Solteq, a company where the DevOps solution was developed from scratch. During the adoption process, there was a need to solve the architecture of DevOps solution for an application used in commerce written on Microsoft Asp.Net platform. The thesis provides a brief overview of commands or procedures used in each stage to clarify the whole process.

The historical background is briefly described; however, the text mostly focuses on the adoption of technical stages of software deployment and the implementation process itself. Firstly, the organizational changes of the work process are described and later the paper focuses on the technical aspects. The thesis introduces each construction block and their usage in every stage of DevOps process development.

Finally, the findings and benefits of the automatization process are presented and discussed. The biggest issues were with designing architecture for a given software product and its organization. However, the implementation turned out to be successful, the practical requirements and theoretical expectations were met, and the designed procedures are still in use. In future, it is possible to extend more features to the existing ones or develop them to completely new project.

Keywords/tags

DevOps, Continuous integration, continuous development, Jenkins, automatization

Miscellaneous

1 Introduction 6
2 Theoretical decomposition 7
2.1. Creation of software 7
2.2. DevOps 9
2.3. History 10
2.4. Bigger view 11
Core 11
Deliver 11
Track 11
Drive 12
2.5. Work organization 13
2.6. Processes 15
2.7. Design 16
CAMS model 16
DevOps evolutionary model 17
2.8. Philosophy 19
Kaizen 20
Kaikaku 20
2.9. Centralized user management 20
2.10. Version control 20
2.11. Work management 21
2.12. Continuous integration 23
2.13. Documentation management 24
2.14. Continuous inspection 26
2.15. Continuous delivery and deployment 26
2.16. Infrastructure automatization or operations 27
2.17. Metrics, monitoring and Analysis 28
2.18. DevSecOps 29
2.19. NoOps and GitOps 33
2.20. DevOps cycle 34
Planning 35
Programming 35
Build 36
Testing 37
Release 38
Deployment 38
Run 39

Monitoring 40
2.21. The future 40
3 Practical implementation 42
3.1. Company and product introduction 42
3.2. Current state of development 42
3.3. Target state 42
3.4. Development process 43
3.5. Architecture of automatization 44
3.6. Phases of creation 45
Preparation 45
Integration 45
Deployment 47
4 Conclusion 50
4.1 Limitations 51
4.2 Relevance and future research 51
References 53

Figures

Figure 1 Typical scenario of code deployment without using DevOps practices 8
Figure 2 Development cycle of software 9
Figure 3 Visualization of DevOps timeline level development 13
Figure 4 Joint and crossing areas of SW development 14
Figure 5 Schematic of interaction of DevOps blocks for processing 15
Figure 6 Cycle of CAMS model steps 17
Figure 7 DevOps evolutionary model scheme by Puppet Report (2018) 19
Figure 8 Conventional wisdom: "pick two" out of 3 criteria cannot be an option 21
Figure 9 Correctly balanced development triangle 21
Figure 10 Pyramid of impact when changes occur in technological stack 22
Figure 11 Various artifacts and platforms can be managed with Jenkins CI tool 24
Figure 12 Different documentation categories 25
Figure 13 Scheme of used containers and component dependencies overview 28
Figure 14 Development and security tools overview per DevOps stage 30
Figure 15 Diagram of security operations per development phrase 32
Figure 16 Visualization of ratio 100:10:1 32
Figure 17 DevOps and NoOps basic differences 33
Figure 18 Schematics of blocks linkage and sequences 35
Figure 19 Organization of upstream and downstream jobs 44
Figure 20 Jenkins CI interface with stages and run time values of smooth build. 49

Acronyms

API	Application Programming Interface
CAMS	Culture, Automation, Measurement, Sharing
CD	Continuous Delivery
CI	Continuous Integration
DevOps	Development operations
DevSecOps	Development of security and operations
IaC	Infrastructure as Code
IDE	Integrated Development Environment
IT	Information Technologies
ITIL	Information Technology Infrastructure Library
OS	Operation System
PaaS	Platform as a Service
SW	Software
URL	Uniform resource locator

1 Introduction

The process of software development and deployment is a crucial activity of software houses and independent developers, helping them to satisfy their customers' needs and requests. The whole technical segment keeps going through a never-ending evolution and improvements. However, automatic integration and deployment have been the leading topic of changes in the field for the past years, and still, they continue altering the field.

For many software developers, the process of developing software ends with pushing the source code to the repository. Their primary and sole concern is the maintenance of repository branches after committing new features before developing new ones.

However, before the code is turned into a product to be operated by users, there is a long process and a list of necessary activities to be done. This thesis introduces modern approaches to those procedures, explaining how to view all the related steps of building a software product and finally, presenting experience from practical deployment.

2 Theoretical decomposition

2.1. Creation of software

Delivered software brings value to a customer – making them happy and bringing financial reward to its supplier, which is the main target of the whole development process from software suppliers' point of view. The trend is obvious; every great company is a software company regardless of their original business field, starting from banks, apparel firms, industrial manufacturing, and several others. Even though they may not sell anything that is called a software, their products rely on software, technologies and IT mindset.

There are various techniques and features which simplify the development process. Their main purpose is to make the development easier, faster, clearer, safer and provide feedback to the author of a code. The purpose of those tools is to provide comfort for developers by solving the given problem itself by putting a great deal of attention there, therefore bringing primary business value to the final user or customer. The situation usually tends to focus on problem solving or bugs, however, the process of delivering it to a customer is slow, manual and painful, and the delivery is as important as writing the code. Because of that, it is good to focus on those steps needed for a fast software delivery with high quality, without an impact of running systems and users as presented on the Figure 1 in a suboptimal case. It describes the need of DevOps for minimalizing of lead time by cutting out time when operation specialists are trying to understand a developed product. Presence of DevOps specialist during the maturing process can support cutting that time lost.

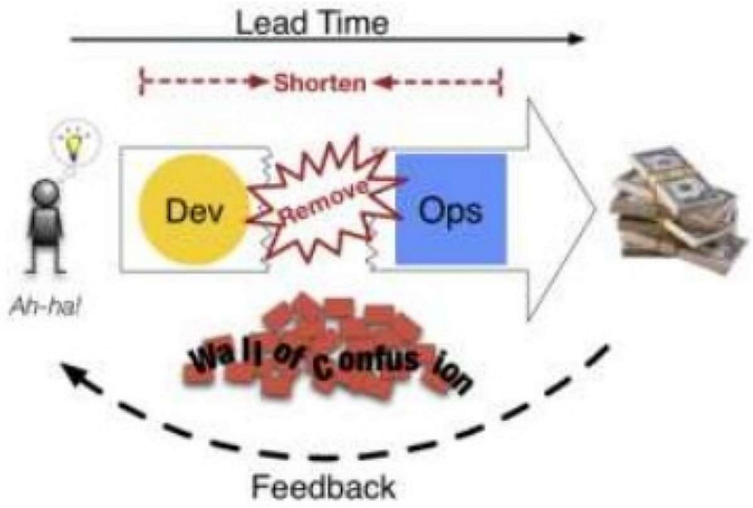

Figure 1 Typical scenario of code deployment without using DevOps practices

The technical questions to be presented in the following sections are as follows: How to deliver a production build to a customer? How to monitor the success rate and specify various factor metrics to make sure that nothing is forgotten, and the quality of the product remains high? These concerns could be significant, especially when these issues touch the versioning of builds and multi-customer environments.

The above-mentioned examples of the issues demonstrate the complexity of software delivery field. Never ending process of development is shown in the figure below as closed loop of developing, deploying, testing and restarting the process. The following chapters will introduce some of the solutions to given challenges and patterns that are

well accepted in the community of DevOps.

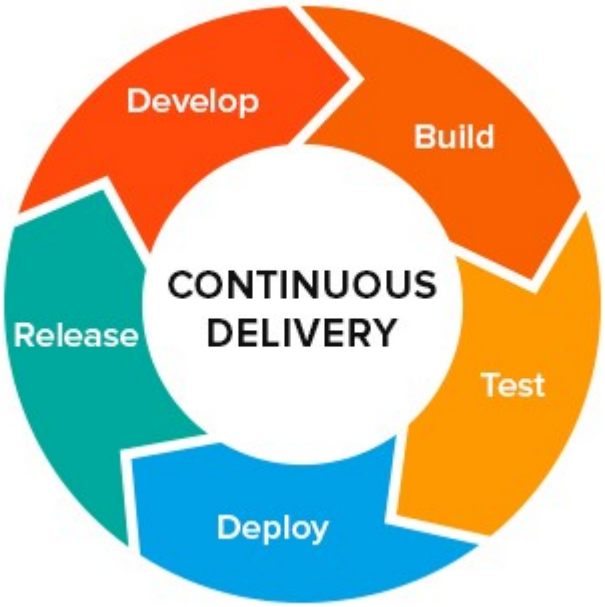

Figure 2 Development cycle of software

2.2. DevOps

The term DevOps is an acronym consisting of two words:

- Dev – **dev**elopers of software
- Ops – software **op**eration**s**

The meaning of the term has been aligned within last years and detailed specification continues. Originally it started as a job position of a person interconnecting team of software developers with a team in charge of software operations. The person had a deep background about development and product, was able to prepare an automated build process, and at the same time, understand the infrastructure and tools related to the deployment. Lately, it was explored that software deployment is not only about provisioning, but it is a long list of steps that also ensure quality and security, as well as monitor and update various environments where the products run. Because of those dependencies that heavily cross several traditional departments or that seriously interact with workers inside different teams, business strategy was not anymore, a challenge in the terms of used tools but in communication between differently based people inside an organization. Therefore, the definition of the term DevOps moved to

express a company culture related to releasing a new version, or a mindset of used approach to handle the changes.

The field or movement itself started to exist, when collaboration tools and versioning systems got popular along developers as software delivery model kept evolving. One way to describe it would be to say that the software delivery process was massively changed in the end-user sector, when the popularity of non-desktop applications on pocket computers (smartphones) found their way to users. The existence of auto-updating applications enabling access to new versions changed thinking about the traditional box software delivery and created the current phenomenon. During years this new approach has reached the conservative business sector and industries where failure was not allowed, therefore putting more pressure onto the process itself. That background opened space for new specialization of the operations sector which is now called DevOps.

DevOps could be summarized with a definition by Chef (Embrace DevOps 2018, p. 3), who observes “Rather than being a static entity, with a single definition, DevOps may be closer to practice, with some underlying principles that remain constant, and with forms and applications that vary according to the experiences of the practitioners.“

2.3. History

The term DevOps was used the first time in 2009 on a *DevOpsDays* conference held in Ghent, Belgium. However, the problematics of development and operations were introduced already in the year 2008 by Patrick Debois in a lecture *Agile 2008* in Toronto, Canada. Although the issues related to software release were known before the mentioned years, they were not considered as a single topic.

The actual subject matter opener was the Agile management approach instead of a waterfall model and identification of the fact that software developers tend to be more flexible in their work than people from other departments, e.g. infrastructure IT-PROs, testers or even those from business department. It was suggested as an outcome to use agile methods not only in development but also for other fields as operations, management, and business.

2.4. Bigger view

Now when DevOps as a term is defined, there is a need to put it into a context of whole software development business instead of presenting it only as a technique dealing with deployment. The four levels of modular scalable DevOps platform can be defined according to the vision of software house Eficode (DevOps Platform Guide, 2018).

- Core
- Deliver
- Track
- Drive

Core

The stage when the software development is stabilized and harmonized between all projects is called core because contains solution for given problem. Simultaneously, the documentation and work tasks are unified to ensure the understanding of progress: what has been accomplished and what will be done in the future. An important part of this stage is also the integration of all development and supporting tools in such a way that it is possible to access all of them using same user account, typically administrated by a company. This stage expects usage of version control system for the software.

Deliver

When the core of software business is ready for shipping, it is necessary to develop and maintain automatic delivery to different environments. This requirement is supported by the existence of continuous integration. It is completely used, as it is no longer enough to just standardize the build and testing phrases of development. The automation must be extended to storing the completed software packages, provisioning the environments and the production delivery. The shortening of development cycles forces the involved persons to carry out automated testing, building and delivery when possible.

Track

This level is built on the previous one, providing expanded ability to communicate and to monitor the state of software development. Based on the metrics and analytics, it is

now possible to compare different projects with each other, which is called development analytics. These outputs are actively used in aligning the right direction of development leadership. Another important metrics are collected through operational analytics to support a constantly growing number of running instances, as it is essential to know which services run and have no immediate problems discovered.

Drive

The level which is also called business enlightenment in some sources it is called drive because reflects conscious controlling of whole development and delivery chain. It is about the usage of properly scaled technologies allowing dynamic environments, where maintenance is carried out by continuous delivery. The core of drive skill is in having the whole process optimized with artifacts and mastered tools tuned. It was already mentioned that the cloud services are used there, however, they are not the only components. Another expected component could also be a service desk, where the staff react to incoming tickets immediately, where they monitor SLAs and provide direct communication to the users.

Other technologies are handled properly as well, as virtualization and containerization are used in isolated instances. Self-service portals for handling on-demand environments are used to reduce time and the needed communication for testing purposes. New approaches such as deep analysis are used or ready for implementation, so when a business needs to decide, it is possible to use a source from for example BigData or Machine learning with Artificial intelligence according to Eficode (DevOps Platform Guide, 2018). If full DevOps solution is the target, then

Figure 3 illustrates introduced steps which leads to having the full experience.

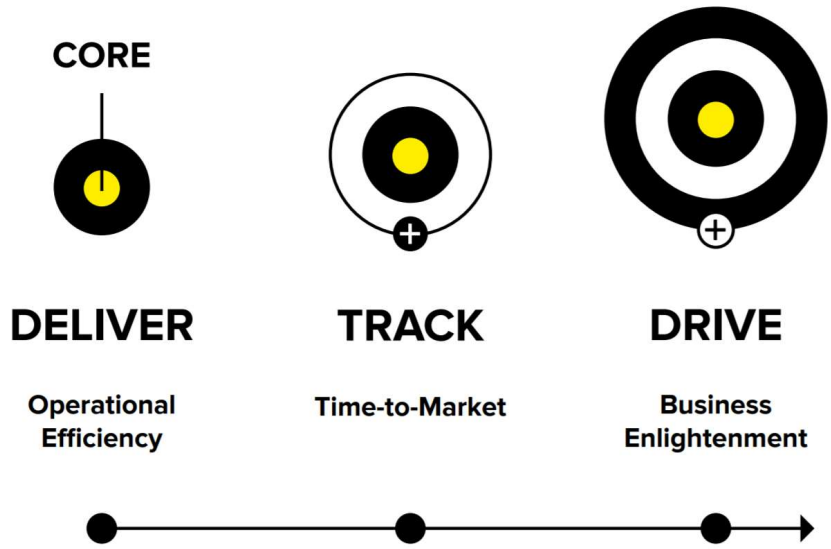

Figure 3 Visualization of DevOps timeline level development

2.5. Work organization

Technical specialists say that repeated activities need to be automatized when a number of executions is higher than one. In practical life, there is usually a need to find a reliable routine during the manual process of evolution, then implement automatization with regard to human resources utilization. However, continuing manual deployment in practice is quite common for smaller teams, however, it turns out to be very ineffective as it is time and resource consuming.

Some articles recommend developing the first automatized deployment solution already in the testing period before the software product is finished and transferred into a customer's environment. The following figure 4 suggests the interconnection between factors affecting the DevOps solution and shows the best result when all the inputs are united.

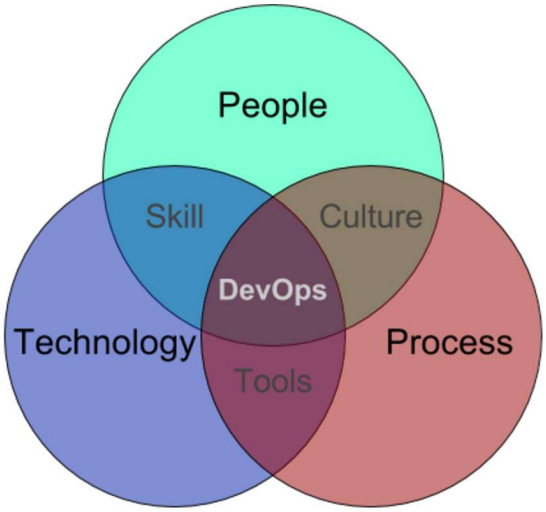

Figure 4 Joint and crossing areas of SW development

The field of DevOps cannot be understood as a clearly specified job position, but rather as a set of best practices combined with prepared processes and specified tools leading to a smoother cooperation between workers, assuring the product quality and a satisfied customer. A developer-operational worker is a person who is setting up, developing and maintaining that process. The value of that person lies in their ability to understand the code, while they would have the necessary knowledge about infrastructure as well.

Integrating or opening a DevOps position into currently operating team is not a painless process. The relations between different teams could be cold, unfriendly or closed however they still need to be interconnected. That kind of concurrency between employees makes the work of an integration specialist harder because there is a need for fixing broken relations to improve common knowledge and shared process. This highlights the importance of leadership, as the management should be able to ensure a good working environment and progress in the project. According to the report of Puppet (2018), the biggest obstacle (48%) in developing DevOps practices in the year 2013 was the problem with understanding the needs of DevOps and unwillingness in changing the usual workflow.

Practically, the software deployment is in contradiction with software development when it comes to multiple changes. Developers produce modifications almost daily, but the operation department dealing with customers do not want to risk and affect the production too often. The deal breaker could be within the quality assurance

specialists since they need to be able to decide if a new version of software product is without bugs or side effects. Before new ideas of a software delivery becomes industry, standard partial troubles between team members and in their communication can be expected.

2.6. Processes

Since the idea of DevOps crosses many team borders and routines, the field of processes is not an exception. Business or management deployment processes usually define the necessary steps before deployment, and their existence can simplify the deployment of DevOps process as they do not need to be invented from scratch. On the other hand, a strictly defined set of steps with a huge amount of byrocracy could limit the ideas of not just DevOps but also of all other teams. As conclusion, an enormous amount of unoptimized processes is contraproductive in case of DevOps. The following figure 5 presents the linkage between blocks strives on the result.

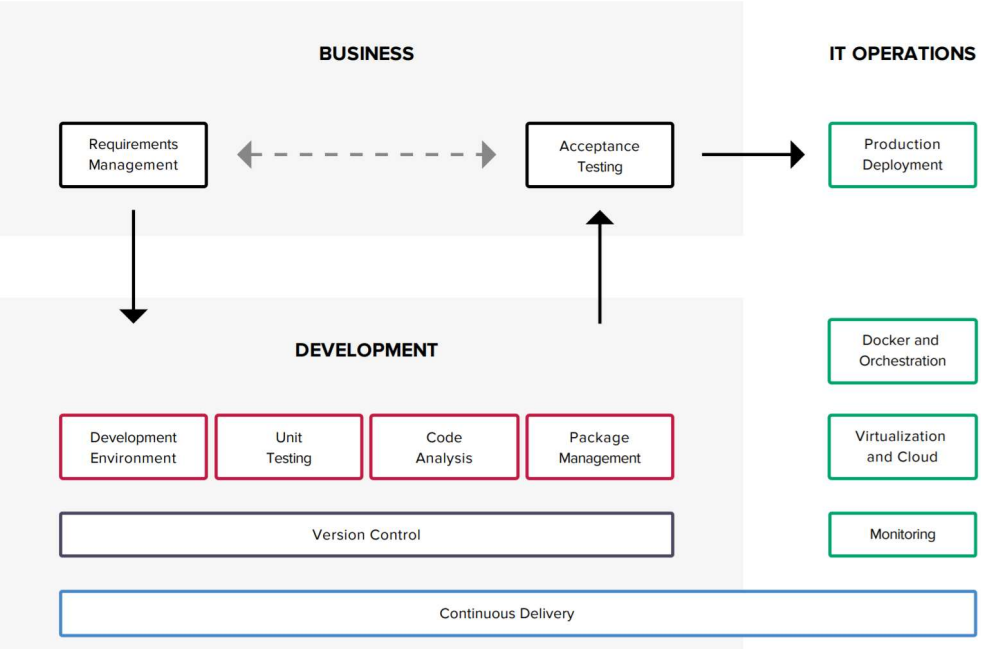

Figure 5 Schematic of interaction of DevOps blocks for processing

The well-known set of rules in IT management are ITIL processes (Information Technology Infrastructure Library) which are in direct contradiction with the relaxed thoughts of DevOps. However, if there are applied ideas of cooperation and communication between teams, then ITIL can harmonically join their work and

support it by providing the process of transition from development to the operational state.

2.7. Design

CAMS model

The processes supporting software building are not the only ones that exist, as there are also processes for developing DevOps tools and methods. One of these is called the CAMS model approach, which defines the steps of continuous integration loop, meaning that it is an iteration model of development for all the solutions to get more mature. CAMS itself is abbreviation of Culture, Automatization, Measurement and Sharing. All those nouns stand for some part of DevOps developer process. The CAMS model was presented in State of DevOps Report Puppet by Puppet in 2018 and this chapter is referencing to that report.

Culture here means that the used methods and tools need to reflect the existing environment and processes of a team. Right understanding and using of culture brings the success closer.

Automation is the core activity in DevOps process as a productivity gainer, as automation also prevents defects, creates consistency and brings self-service. On the other hand, some researchers claim that automatization is only 25 % of the workload of DevOps specialists.

Measurement is about monitoring, gathering statistics and tracking the performance of solution, which is crucial to get feedback. It is an important step to be able to evaluate the progress of the key collected metrics. This process allows to learn from mistakes and avoid unnecessary work, since that is the most expensive part of development.

Sharing is about providing feedback freely without blaming team members, ensuring that everyone can learn from mistakes or from the gained knowledge. It is a key component not just of DevOps but of the whole society. On the other hand, sharing is about consuming resources such as people’s time during meetings or writing and reading posts. In general, however, it brings an important preselected knowledge to appropriate recipients.

The following figure describes that after all the steps, at the end of finalizing a sort of functionality, the developer returns to the start of the development process adding there more features. All of these circles in a loop until a project is finished.

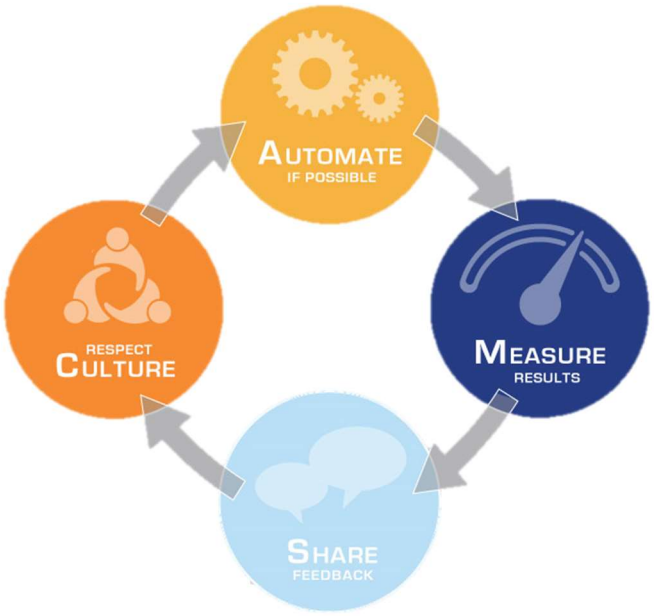

Figure 6 Cycle of CAMS model steps.

DevOps evolutionary model

The previous CAMS model describes iterating evolution of solution but what if the development of DevOps were to be seen as a classic waterfall model from the start to the end? The following evolutionary model introduced in *State of DevOps Report 2018* shows the DevOps journey from the beginning evolving to the final product. The nested iteration model can be even more nested in each bullet, however, in general, this model goes within time. The main idea is non-returnability as a classic waterfall with the sequential design of the process in software engineering.

The model itself consists of several phases, refers to the name of the model as evolution model. It is about get the process done from the begin to the end. The key idea of the list is that the development of DevOps techniques follows the development of software product, so none of projects can be ahead of others.

Stage 0: Build the foundation

This phase is about to adopt the concept of DevOps in the born product, to think about the future impact and to assign the roles in a team, or even to find new team members.

Moreover, this phase accepts the idea and starts with basic integration, which works as a base for successful usage of DevOps methods.

Stage 1: Normalize the technology stack

When a basic model is released, it is time to decide which technologies are going to be used for the development of a product itself and a decision for DevOps tools is also needed. That is the starting point, when the complexity grows rapidly. Refactorization could probably be needed later; however, dramatical changes of technology stack after this step are very painful and expensive. At this point, version control (introduced later) have to be fully adopted in the project to be able to expand according to authors of report State of DevOps 2018.

Stage 2: Standardize and reduce variability

"This stage is where both dev and ops teams concentrate on reducing variance" and bring some universal solution of deployment across several related projects. Since technologies are selected in previous phase, it is then about how to use same construction blocks or how to synchronize the used design patterns. The main reason for this optimization is to reduce complexity, which supports teams to share expertise. Additional value of this step is easier management and reduced documentation needs, the report describes.

Stage 3: Expand DevOps practices

When proof of concept is ready for production (if not already), it is time to add more functionality to the current solution. The idea of an existing minimal viable product does not meet with the needs of real production environment. DevOps specialist need to put more effort to create actually a production styled release procedure and do not keep it for later, because many factors can be missed during incremental continuation in development. The drawback is that it consumes lot of resources, especially time. The best thing for fast solution is a situation where these steps are documented, in suboptimal case there is a need to make research of used practices and implement them into DevOps process.

Stage 4: Automate infrastructure delivery

When processes are implemented, and they can do what is requested, it is time to let them do what is needed without an interaction of a developer. If all previous steps were considered seriously, then the current step is just about the refactoring of scripts

and to pipeline them together. Automated infrastructure delivery resolves the issue of developer's utilization and releases hands for putting effort to another case.

Stage 5: Provide self-service capabilities

The last step of build process brings the highest value out of the whole automatization. The cumulative effect is in satisfied participants of deployment chain who get fast feedback for the current release when self-service portals are deployed. The key feature is that IT teams do not need to respond to the requests every single time, because they are handled by the self-service option which is highly beneficial for both parties.

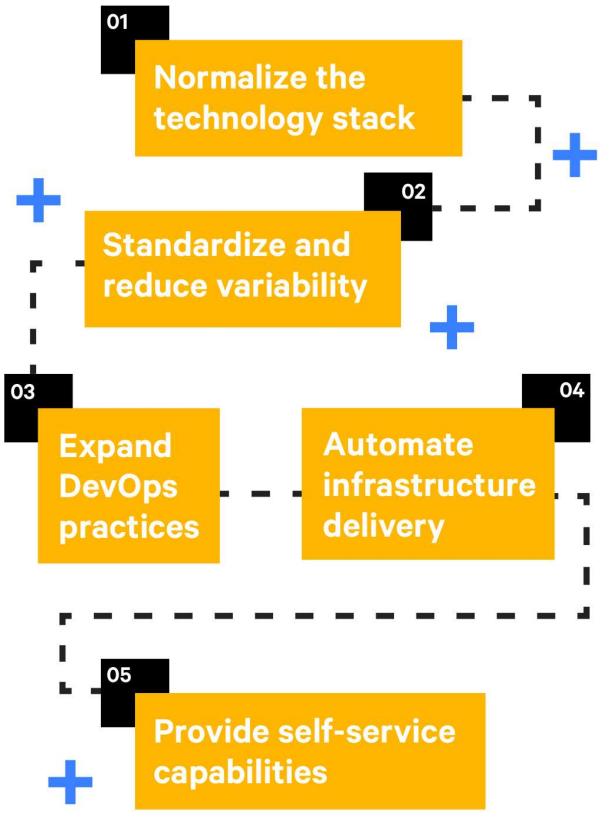

Figure 7 DevOps evolutionary model scheme by Puppet Report (2018)

2.8. Philosophy

Another important part of DevOps processes and design is the approach how to do it, a so called philosophy or mindset. A variety of used techniques can be used, many of them are often named with Japanese terms as Kaizen and Kaikaku, which complement each other. Originally, they were developed as a part of Toyota management systems as presents The Lean Management Systems Handbook. Both fit well for the purposes

of DevOps, because they maintain the systems, and make important small changes in specific areas.

Kaizen

Improvement and optimization methods are based on slow continuous improvements over the time. It is well fitting for everyday decisions to keep the “wheels spinning”.

Kaikaku

A complementary method to Kaizen is Kaikaku, which stands for radical and rapid changes, builds from scratch and reinvents already invented within limited period. Kaikaku is best to apply for very specific scope only.

2.9. Centralized user management

The important part is to have an united user management of involved members to drive permission handling and communication. Usually that part is handled by companies that are giving employees assigned emails which also work as single sign-in modules for other tools used for internal purposes. The main concern is to allow people to be inside when they are needed but also to remove them from resources when they should not access them. After all, the worst case is to give permission to users on personal accounts, according to Eficode (DevOps Platform Guide, 2018).

2.10. Version control

One of the basic construction blocks in the process of preparing DevOps solution is the usage of some sort of source code version management tool, which allows developers to contribute to the shared project and, on the other hand, allow software release tools access the source code chronologically with transparent tracking skills. The version control system also sometimes called revision or source control system allows reverting last code changes easily. There is no space to describe this complex system, but what is salient to know is that sharing the code inside version system during development later enabled the foundation of DevOps phenomena. The most known versioning system nowadays is Git.

The version controlling practices can be presented as a set of properties: **atomicity** stands for separation of processed changes, a commit should always change just one

thematic issue, even though it was to be processed at the same time. Each commit should also be **isolated** in the meaning of independency from other commits, so it can be applied separately to a repository. Another issue is the title and message following certain rules used for committing to support **traceability**. Version control should be also without generated and temporary files to maintain the **essence** of source code. Key request is also to keep **functionality** preserved with every commit, so incomplete changes are not published. These requirements were summarized by Michael Ernst in his article Version control concepts and best practices.

2.11. Work management

During the time and implementation of DevOps solution, questions usually arise outside the area of tools, thus starting interfering with the other departments and business needs. Three steps of DevOps levels can be recognized: culture, tools, and process. Nowadays, a business process of delivering the value is connected with the IT process. Following two figures present bad and good approaches to development. Statistically there is always pressure to put something to prefer over other criteria.

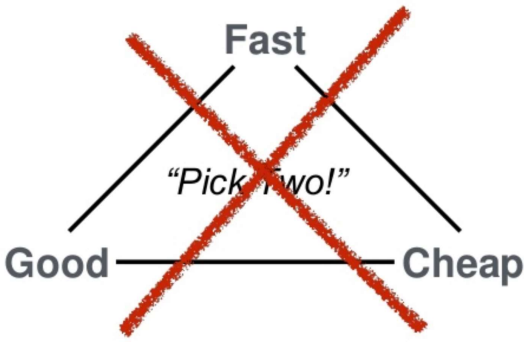

Figure 8 Conventional wisdom: "**pick two**" out of 3 criteria cannot be an option

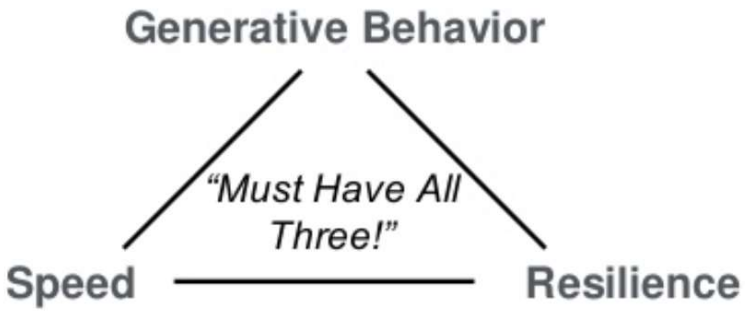

Figure 9 Correctly balanced development triangle

Adoption timeline recognizes the hierarchy of the DevOps needs, enterprise requirements and management and business needs. Agile methods defining the strategies of how to handle the work management and processes are widely used approaches for leadership in IT field.

Figure 10 below explains the relationships and dependencies between technical and cultural changes and their possible impact on users. There is an obvious indirect proportionality between the technological and cultural changes in development which are interdependent. It is not surprising that technical improvement itself has a minimal influenceability on people, as the change does not bring values to them. On the other hand, transformation in culture cannot be achieved if the technical background is not evolving.

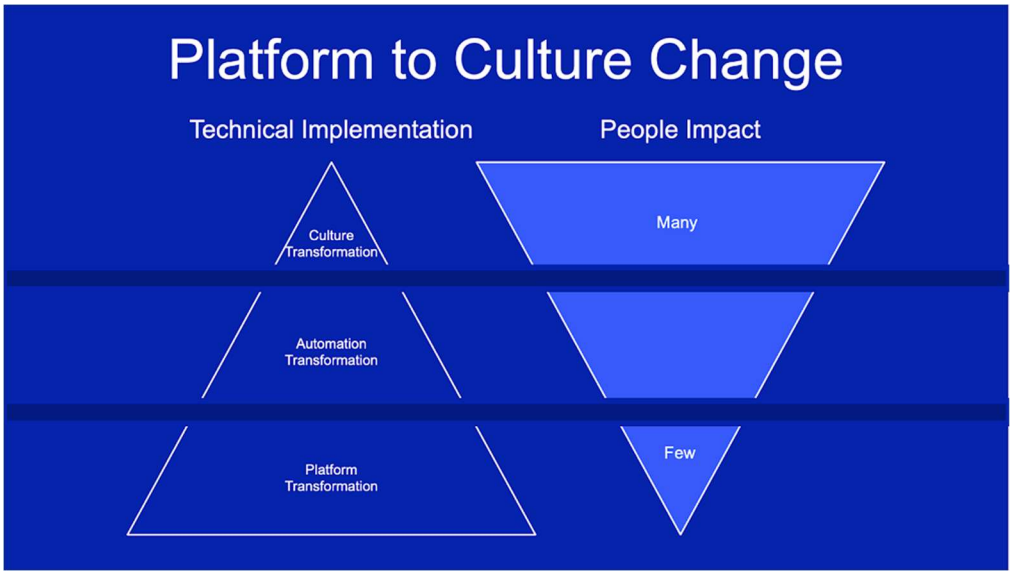

Figure 10 Pyramid of impact when changes occur in technological stack

The described process is not simple, and it usually needs an experienced leader to be able to establish a successful change in the required direction. Specialized community compounded of management and DevOps specialists have a saying “There are many paths in a DevOps evolution to success, but even more ways that lead to failure.” This one comes from Puppet - State of DevOps Report 2018 as same as statement that “It is not unusual that companies hire transformation consultants who help them to move their workflow and culture standards to meet the modern targets.”

One significant warning should be given concerning the topics that are being solved between the consultant and the employees. A company called Eficode in their

presentation ‘Sauna IOT Solution’ on conference DevOps2018 in Helsinki presented research where was calculated that the average cost of experienced DevOps consultants in Finland is for a person 110 € per hour. Usually there is a need for one consultant per each developer team, and the main goal of consultants is to improve the DevOps environment. Even if the specific consultant spent only 10 % of their work time on aligning tools instead of tuning the cultural and technical issues, the monthly cost would still reach 1800 € per month. This could be considered as a high price, thus having an effect how far and widely the change would spread. Eficode researched by a questionnaire that 80 % of DevOps transformation consultants spending more than 50 % of their time on tuning the tools, making the cost to be 9000 € per month for one external worker.

2.12. Continuous integration

The idea of continuous integration merges changes from all participating developers into the actual working product as often as possible. This process is executed usually several times per day to avoid the big impacts and unwanted surprises when collisions in developing occur. Another big benefit is the rapid feedback and availability of delivering a minimum viable product at any moment of request. This chapter summarized thoughts of Jennifer Davis from the book Effective DevOps, 2016.

The term Continuous Integration (CI) was introduced already in year 1991 in *Object-Oriented Analysis and Design with Applications* (Booch); however, the technique got its current meaning only when agile development started to be massively used around the year 2009. The keystone of CI is in the usage of version control system and in the automatized build process, as these features of software development allow bringing the new approaches to collaboration.

A practical application of continuous integration process could be as follows. After getting an assignment, the task of a developer is to download the local copy of code from version system and to start making changes. After the developer has finished editing, the local changes of code are committed back to the version control. Potential conflicts, with other developers working on nearby code related features, are resolved. The automatic request launches the continuous integration tool. The code is validated and tested, making sure that the changes did not affect the solution and operability of

the product. If no troubles are caught, then the change is correctly integrated into the system; otherwise, a message notification or warning is generated to the developer by a specified channel and fix on the specific part is requested. The idea persists when big changes are made for a long time without feedback, then the process takes longer to isolate the problem, investigate it and mainly remove to full functionality.

There is a variety of products which can stand in the role of a continuous integration tool, e.g. JenkinsCI, Azure DevOps, Ansible, TravisCI and others. Mentioned tools are having wide technology platform straddling as illustrated on the figure 11.

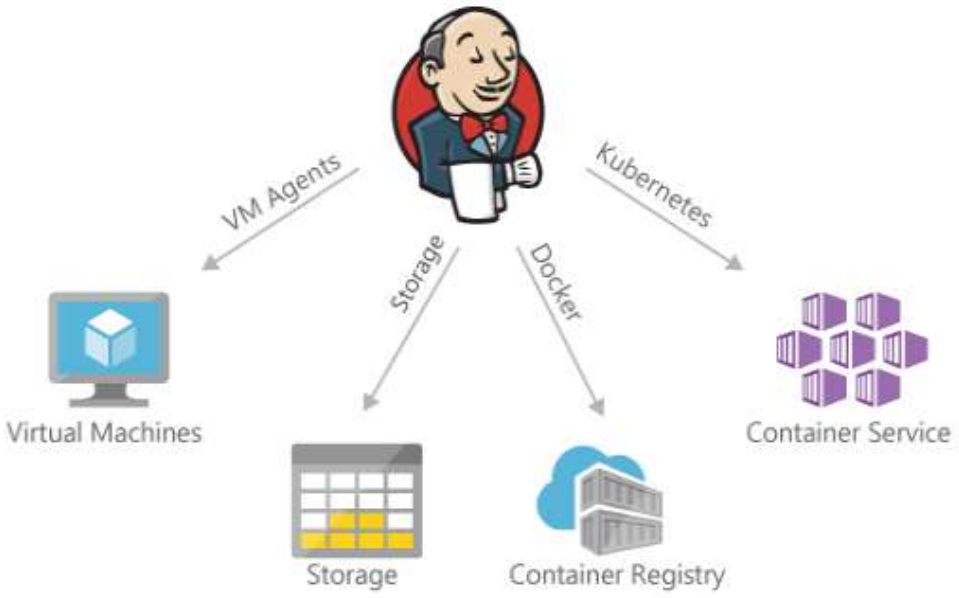

Figure 11 Various artifacts and platforms can be managed with Jenkins CI tool

2.13. Documentation management

The value in computer systems is based on information knowledge and sharing that knowledge between the interested persons. One of the challenges in the development process is the limited capability to share the knowledge in an acceptable way and avoid the outdated documentation. Additionally, if one of the key employees would be unavailable for critical production hotfixes, it could cause issues to the whole process.

The key parts of developing or developed information system to be documented according to Jennifer Davis (Effective DevOps 2016) are listed below:

- System design
- System construction

- System implementation
- Review and maintenance

The document management system in general should cover the whole life cycle of documentation, which consists of the following parts, according to the research of authors Mesnita and Oprea (Information Systems Documentation 2006). Figure 12 presents documentation lifecycle and branches of support documentation.

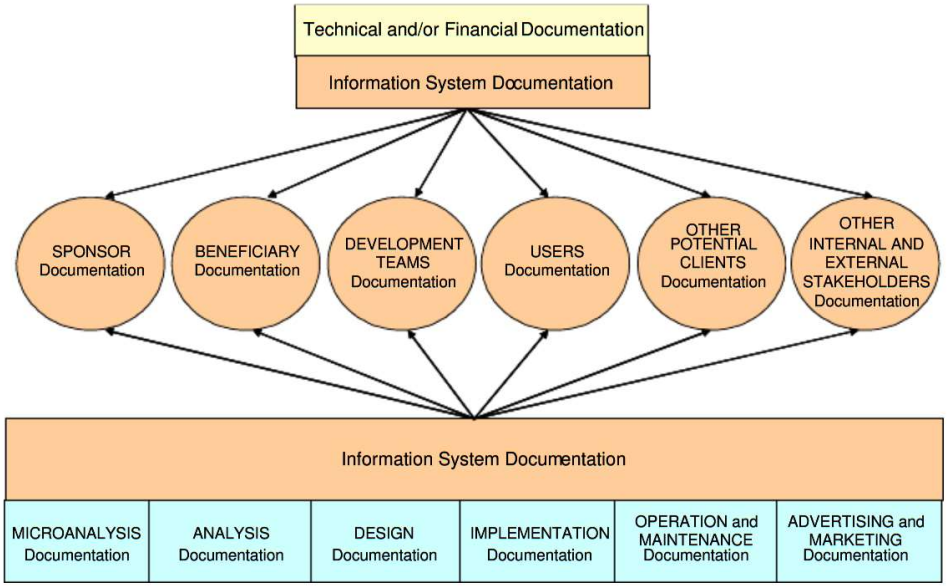

Figure 12 Different documentation categories

The data needs to be held in a secured environment, where tracking of the changes and modifications done by authorized users is available. The readability of the document must be limited to company scope area (chapter on *Centralized User Management*) according to the sensitive data protection handling policy if it is existing or general security best practices.

One common practical problem in development is the outdating documentation in the time when changes are not reflected in the documentation due to the requirement of additional work. Nowadays, a solution to that issue is based on an automatic documentation component, which generates the basic skeleton of documentation from the source code, and programmers are just asked to work for an additional description. The same process can be executed when changes occur, and the notification about the change could be addressed to the original author.

2.14. Continuous inspection

Buildability is not the only metric to evaluate if the source code is good. There is a possibility of monitoring more parameters and evaluating the quality of source code during static analysis, thus bringing additional feedback to the developers if some part of construction is not well made.

The static analysis reports allow avoiding the current or future issues, when a code seems not to be working or when unsecure methods are used. Security in detail is discussed in more detail in the chapter on *DevSecOps*. Practically, it can also be used for detection of copy-paste code parts, and refactorization can be offered. If some metric of code complexity is needed, continuous inspection can provide cyclomatic complexity number "quantitative measure of linearly independent paths through a program's source code". Another issue to make sure with this method could also be coding style and standardization of code formatting.

The drawback of tools performing code inspection lies in the definition of rules and their disability to understand the product or a developer's mindset in practice. A report can contain false alarms, or it may just not recognize some technique or expression. False alarms are highly unwanted for various reasons impacting the mood of contributors.

Markets offer several code review tools, mostly in two forms as a plugin to programing tool (IDE) or as standalone application evaluating code during the build. The representatives of standalone software for continuous inspections are e.g. SonarQube and Get-Codeflow.

2.15. Continuous delivery and deployment

These terms represent the steps following a successful code integration. When there are ready installation packages from build, then there is a space for several delivery methods to the different environments, typically for testing or production space, where the app can be tested or used. The delivery process of the new version is often unsafe because of the danger of interruption in service of software.

Continuous delivery differs from continuous deployment by a number of manual steps needed to complete the installations. The deployment should be automatically based

on most of the definitions compared to the continuous delivery, where the manual tasks to finish software installation are expected.

Practically the difference is based on the target field as critical business software is to be upgraded in a different way from mobile apps via central repository. A bigger amount of small changes requires more automatic delivery practices. The disadvantage is mostly in the need for a detailed set of quality tests. That process also needs support from the receiving side, typically a customer, since without cooperation it would be impossible to put the continuous delivery into service. Continuous integration remains sensible even without using automatic delivery methods, because it can be replaced by manual installation.

2.16. Infrastructure automatization or operations

This chapter changes the description from product development to the development of an operational component. Infrastructure as a code is a new approach which allows dynamic settings of all required resources (software and hardware) based on a specified prescript. This text-based definition of configuration is used for creating virtual machines or other resources from only one configuration file saved in the versioning system with benefits of tracking changes. Practically it could be thought as an installation of operational system, middleware, database server, settings of user rights, placing configuration files and similar variety of tasks (Effective DevOps, 2016).

This new way of thinking was adopted after spreading the approach of virtual appliances and environments with programming access (API). The resources there are not anymore tight with the hardware. This approach is called Platform as Application (PaaS). Software defined infrastructure brings all the benefits of software versioning, which was introduced in the previous chapters, to the scope of infrastructure management and uses them for deployment of resources. The benefit is in the simplicity of definitions on a single point spot with easy to track changes in time including easy to reverse to the previous configuration in case of problems; thus automatic tests could be applied. Self-generated documentation is also an important benefit when used. Repeatability is important for scalability because scripts can be the

definition set for more than one running instance. Figure 13 presents change in application architecture related to hosting system.

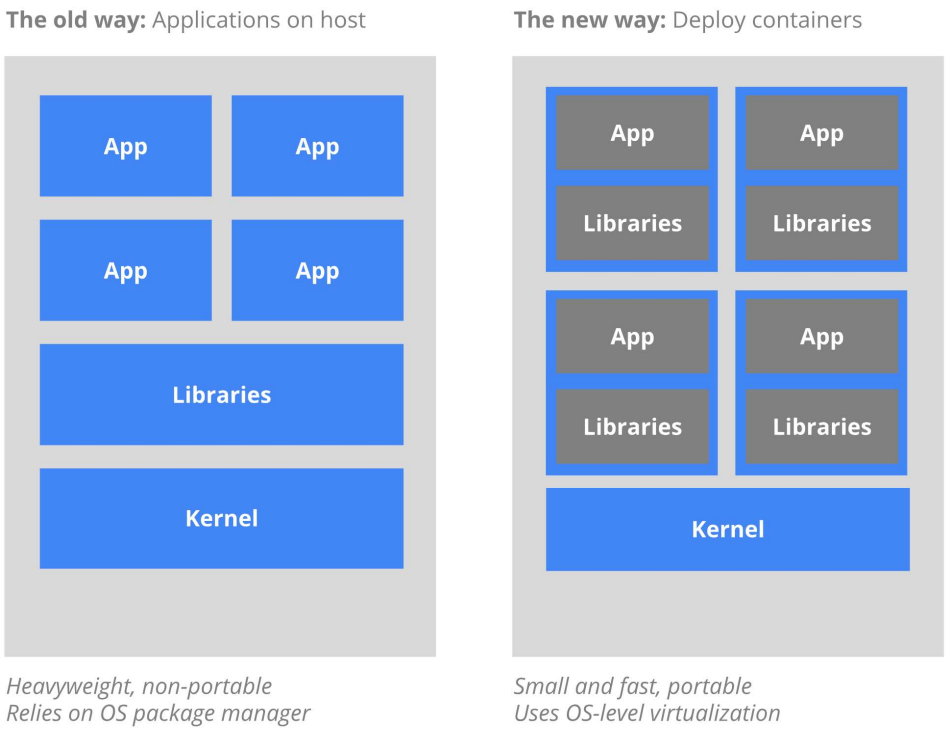

Figure 13 Scheme of used containers and component dependencies overview

The technologies supporting the concept of Infrastructure as Code are used in containers systems for application isolation (primary Docker) and container orchestrators such as Kubernetes, Messos or Swarm. In the outside container world, they are often used in configuration and with deployment managers such as Terraform, Chef, Puppet or Ansible and others, depending on technical aspects.

2.17. Metrics, monitoring and Analysis

Collecting and evaluating the measured values and gathering statistics are only one way how to optimize process and product. It is highly challenging to aggregate the traditional metrics from complex values to the single and easy to understand number. Definition of composition method is a highly exhausting process, which could even be manipulated to make management or customer satisfied, even if the real status of project was different. (DevOps Practices and Their Usage, Vaněk, 2017)

The main reason for the existence of metrics is to evaluate or to have feedback how much certain features are used, to check the speed and ensure the stability of product. Those numbers should be easily accessible for developers according to best practices.

The disadvantage of using metrics is the need for a strong policy in background which increases the demand for management. It is well known that evaluating people or teams based on statistics or reported numbers tends to change responsibility and open discussions or concurrency about categorizing of values. Vaněk (2017) notes: “There is a high need for using the metrics in a positive way, to be beneficial for the customer or management from a wider point of view.”

2.18. DevSecOps

Developer security operations are the philosophy of integrating security practices within the DevOps process, and this philosophy is highly linked with Security as Code approach. The main idea is in merging two seemingly opposing goals: “speed of delivery” and “secure code” into one pipelined process. It means that security testing is performed in iterations invoked at the same time as the continuous integration execution. Critical security issues are recognized as they become apparent, not after a threat or a compromise has occurred in Ilkka Turunen’s view (Securing Modern Applications: The Data Behind DevSecOps 2018).

Article DevOps vs DevSecOps on the portal Sumologic.com summarizes best practices for adopting DevSecOps in a checklist of six steps:

1. Code analysis – deliver code in small increment, so vulnerabilities can be identified quickly.
2. Change management – increase response time by allowing anyone to submit fix.
3. Compliance monitoring – be ready for an audit at any time.
4. Threat investigation – identify potential threats proactively and be able to respond quickly.
5. Vulnerability assessment – when new vulnerabilities are identified with code analysis, then measure speed of reactions to be patched.
6. Security training – train software engineers with guidelines and continuing education.

Practically DevSecOps can be realized with several tools and techniques categorized as Static or Dynamic application security testing. Examples of tools for performing static scan were introduced in Continuous Inspection chapter. The density of dynamic security analysis tools is much higher, as there are many specialized open source or commercial software on the market, for instance, Nessus Vulnerability Scanner, Arachni penetration test or any of the tools grouped in OWASP association. Inside the community, there are also well-mentioned tools such as PumaScan, Coverity and Fortify, more tools fitting each DevOps phase you can see in following figure 14. The tool's scan reports provide developers with feedback on security issues. The team needs communication rules and an open mind for processing those reports for accepting the feedback based on the urgency of issues. (Turunen Securing Modern Applications 2018)

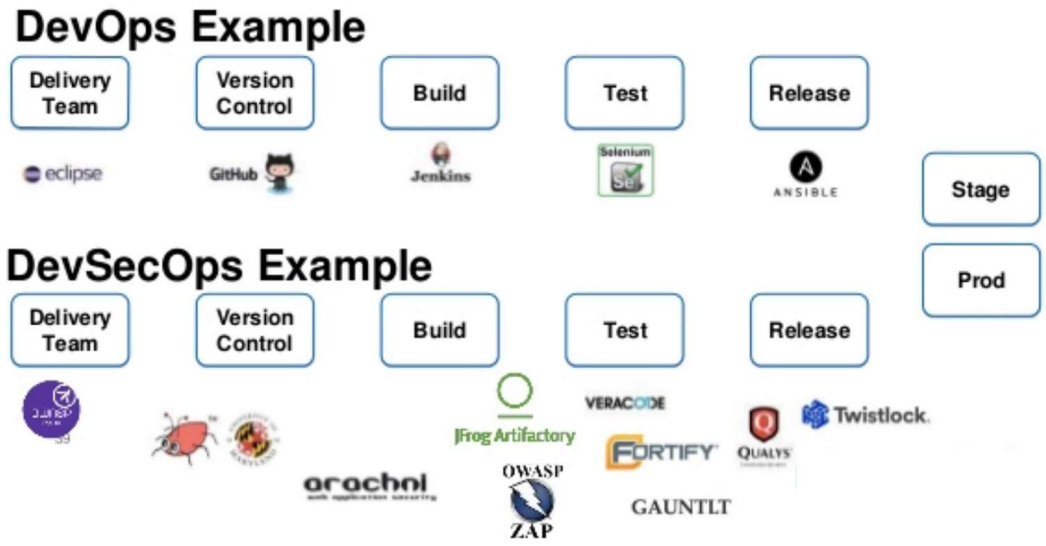

Figure 14 Development and security tools overview per DevOps stage

For each stage of development there is a list of security practices and activities to be adopted for increasing the security and lowering the risks. The list was presented by Victoria Almazova in the presentation "Secure your Azure and DevOps in a smart way" on conference DevOps 2018 Helsinki.

Pre-commit phrase steps

- Threat modeling
- IDE security plugins
- Pre-commit-hooks

- Secure coding standards
- Peer review

Commit phrase steps

- Static code analysis
- Security unit tests
- Dependency management

Acceptance (CD) phrase steps

- Infrastructure as code
- Security scanning
- Cloud configuration
- Security acceptance testing

Production phrase steps

- security smoke tests
- configuration checks
- penetration testing

Operations phrase steps

- Continuous monitoring
- Threat intelligence
- Penetration testing
- Blameless postmortems

Figure 15 deeply analyses security risks per stages of development and describes techniques needed to secure that phases.

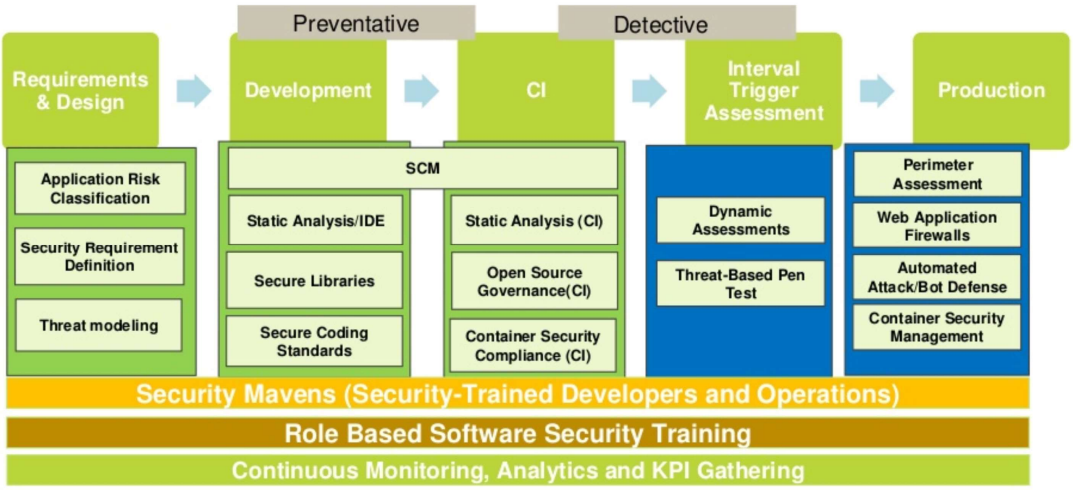

Figure 15 Diagram of security operations per development phrase

The ratio of specialists in development, operations, and security team should be 100:10:1 in a typical technology organization as presented on following illustration 16. This division ratio constraint could be changed when deep automatization is used; nevertheless, some security review is needed from time to time.

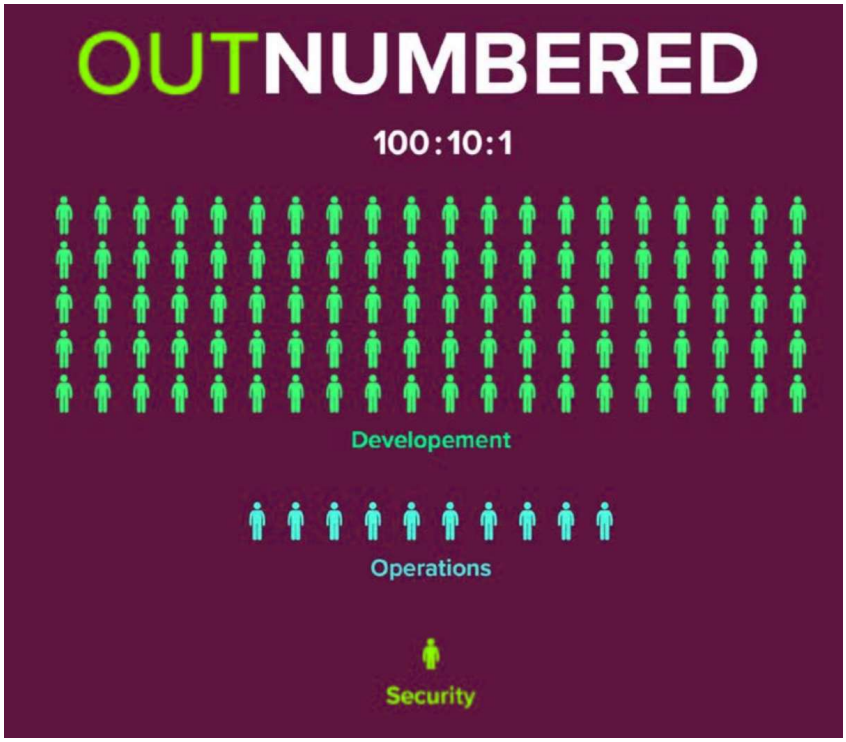

Figure 16 Visualization of ratio 100:10:1

2.19. NoOps and GitOps

Development in continuous integration still evolves, and the boundaries between systems are more and more erased, so the new overall look to that field is possible. The practical impact after implementing ideas behind these new buzz-words is always lower than expectation, as the presented approaches do not fit every case. However, it is useful to deal with modern methods in order to understand future trends.

When the level of authorization for continuous integration and deployment reaches high enough level, it is possible to skip all the manual steps for putting software product to the service by adopting NoOps approach. This allows to remove dedicated operational specialists for installations, monitoring, and management of applications and replace their workload by an automatic script in theory of NoOps. The goal is in complete automatization. The difference between NoOps and DevOps is in the removing of boundaries between teams of developers and operational specialists. The term NoOps extends the term DevOps as continual integration solution. Summarizes Bach in the article Is NoOps the End of DevOps (2017). Figure 17 highlights the difference between the approaches in bullet points.

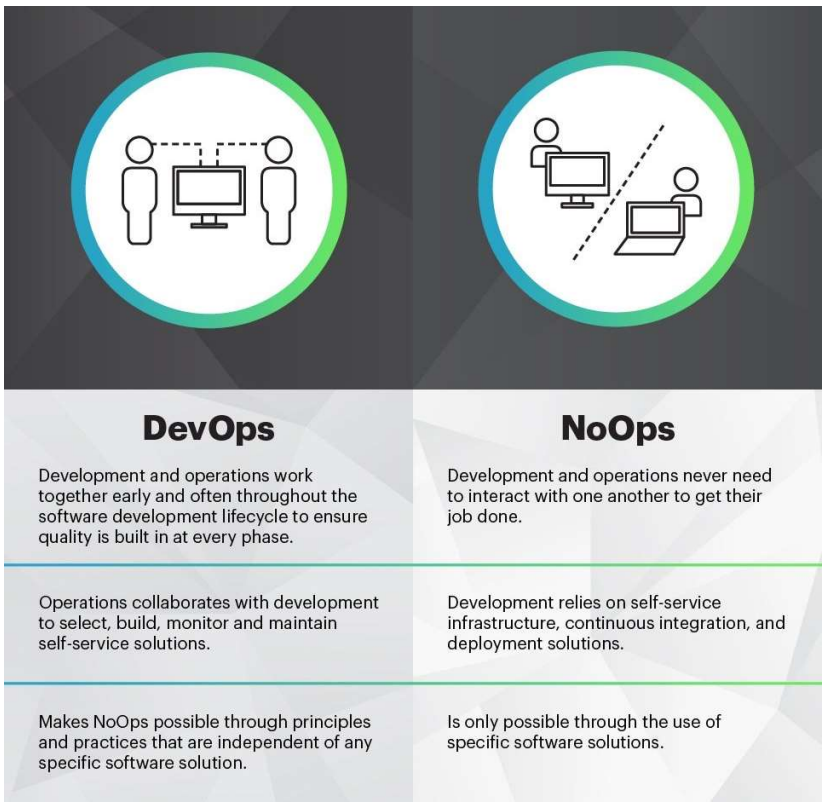

Figure 17 DevOps and NoOps basic differences

For an experienced reader, several disadvantages are obvious: NoOps is limited to applications which fit into the existing PaaS solutions, thus not being beneficial for monolithic legacy applications requiring a high amount of refactorization work or upgrades to get requested behavior, because their architecture was designed before Ops phenomena. Additionally, DevOps definitions discuss more interconnecting people's work to speed up and optimize a software release to support business. NoOps is more interesting for smaller teams, companies or one-man projects where the lack of resources is critical and additional automatization allows them to increase their productivity. (Bach Is NoOps the End of DevOps 2017)

GitOps is the next step in advanced automatization based on declarative provisioning tools. Declarative approach is a crucial part, because it allows to change the specification in sets of truths instead of being a set of commands and comparing the result with the configuration file in the versioning system. The term GitOps is commonly defined as a set of features in solution: (GitOps - Operations by Pull Request, 2018)

- Git as the single source of truth of a system
- Git as the single place of operations of all environments
- All changes are trackable

The main benefit of GitOps is that it empowers developers to perform operation tasks. That approach can be linked with previously expressed term NoOps.

2.20. DevOps cycle

The build is not kept in one piece but usually is divided into several logical blocs where it is possible to focus on specific issues discussed in following chapters. The main idea is that the loop of development is infinite as presented in Figure 18.

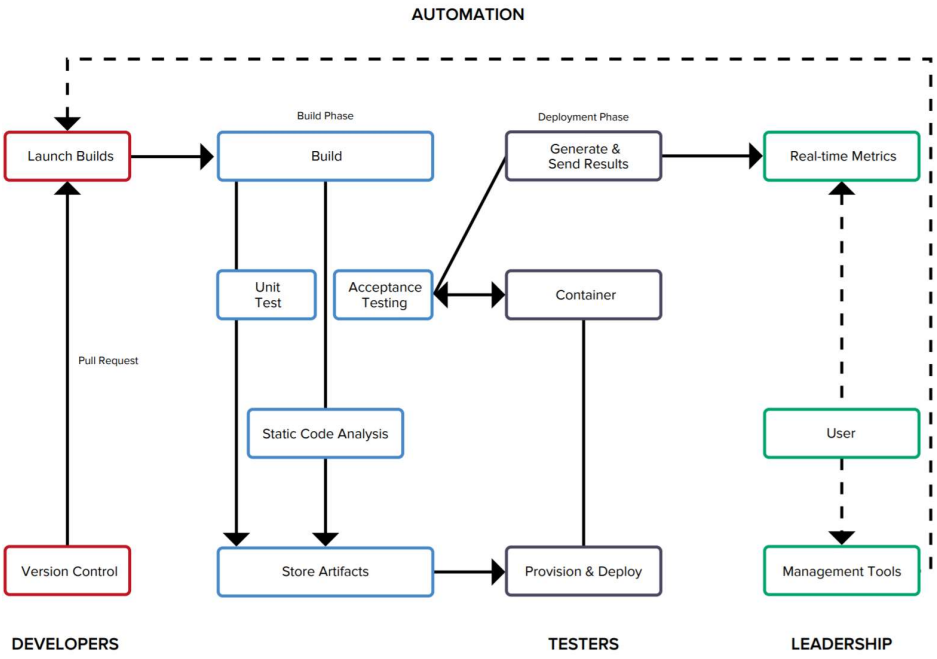

Figure 18 Schematics of blocks linkage and sequences

Planning

The starting point of DevOps cycle occurs, when the required tasks to be finished are defined and described. The main idea is to use a single system, format and style for specifying details and use straight integration to other currently used systems. The main idea of support system is to support the process and task, so it is important to keep the pressure on the specified properties of defined jobs, such as: **atomicity**, the task is finalizable in a short time and interconnected with other jobs by having **identificators**. The system is also able to hold on information about status of the job and to be persistent even in the case of denied or canceled job (Agile Operations 2015).

Best practices also mention that the system for managing development requests should be the same with the operational team to allow the solution sharing and linking. This could also help to create motivation and gives a good overview of team's workflow.

Programming

This is the part of cycle, when the source code is created based on challenge definition and when a great deal of effort is inserted by developers. The selection of used technologies and programming languages is important in following steps of DevOps process, because they affect how build is executed in automatic processing according to (Effective DevOps 2016).

The most important tool needed in the programming part, concerning finishing tools, is Integrated Development Environment (IDE), which supports the development by automatizing build on a local computer. It is important that the series of steps are done by local tool that they can be replicated automatically on remote machine for the purpose of DevOps process. Without that, the support of automatic deployment is highly challenging. Some issues, mostly between people can occur if the local builds go smoothly without the participation of a developer. This could lead to a situation, where those people developing remote processes could be seen as unproductive.

Build

It is a phase where the source code is converted into an executable application. The conversion is highly dependent on the used programming language and technologies. There are some similar steps in the general description which are repeated across different programming platforms in every case. Under the build there is the main part compilation of source code, which is followed by linking third party libraries, drivers of current platforms, copying config files, generating user interfaces and lot of other subtasks. (Effective DevOps 2016)

Configuration and DevOps specialist Bob Aiello (Configuration management best practices 2010) summarizes the build procedure and determines several important properties accordingly:

1. Simplicity – build should be easy and simple, approach “one button launches” is the best for this case. Using complex technologies and procedures decreases the chance to let it run smoothly.
2. Repeatability – the process must provide same results over the same source data.
3. Speed and usability – build should be easy to launch locally or in central environment via tool without long and complicated procedures.
4. Universality / multi-environment – correctly designed build should be applied to all intended environments as testing, preproduction and production. Platform differences like connection strings should be modified at the place via configuration files or during deployment to environment. Testing or production should use the same datafile.

5. Trackability – every build should provide version and detailed info about used source and what requirements were fulfilled.

Testing

Quality assurance is a very complex and resource consuming area. Testing is one of the last tasks of developers to get the proof that the implemented feature is correct and that there are no other parts affected in existing software. After the developer's check, there is space for automated tests and QA specialist manual tasks for a final review. The complexity of that review grows exponentially with the code line numbers, cyclomatic complexity and/or list of implemented features. (Vaněk 2017)

DevOps practices try to bring the development and quality assurance teams closer, because in traditional companies the teams are easily separated due to the lack of communication or personal reasons, mostly related with time demands or the pressure of providing feedback. There is an existing emphasis on repeating automatization of used tests and not waste employee's time and skills for routine tasks.

It is also a well-used practice to begin from the start every single time when the deployment is executed, so no special or manual commands persist. That behavior is allowed by Infrastructure as Code pattern that was explained earlier. Some practices also advice to support that model to deny administrators to access manual creation of any objects and let them only be produced by a script. This restricted rule is performed by a read/write permission per group.

Types of possible tests to be performed in sequence (Vaněk 2017)

- Unit testing
- Static analysis
- Security scan
- Integration test
- Component test
- Performance test
- System test
- Acceptance test

All of these test types check different levels of program code, from elementary unit tests for checking the methods or condition blocks continuing by a component test, which checks the functionality of a block. Complexity and running time usually increase with every level of tests to be able to perform a detailed check of the whole program. One of the final tests from a developer´s point of view is a system test which checks the functionality from the user's point of view. That kind of test focuses on GUI or some other output interface; no internal method is checked separately, and the access should be only from outside. The last step is an acceptance test, which is performed by the receiving customer, or user confirmation that everything is implemented as ordered.

Release

When a build is finished and successfully tested, there is time for completing the installation package. Vanek in his thesis (DevOps Practices and Their Usage 2017) summarizes several steps need to be completed to convert the source code into a full software artifact.

- Marking the build with an unique identification
- Storing meta information about the build to a repository
- Creating a change list
- Making packages per platform for distribution
- Collecting an artifact for deployment
- Documentation update
- Preparing environments for deployment

Usually a package is deployed to a pre-production environment to ensure that all of the packages and applications are well-made and tested. The DevOps approach emphasizes the full automation of the release process and all necessary steps to provide fast and repeatable initialization for a specific build based on unique identification.

Deployment

The main goal of the deployment phase is to deliver the application to a customer, allowing them to start using it. Manual deployment or automatic continuous delivery is essential to do fast, smoothly and without downtime. Usually the production

deployment is a final step of acceptance tests confirming that a software product is eligible to replace the currently running version in the production environment.

A list of steps of necessary actions (Vaněk 2017) needed for deployment may include:

- Package preparement
- Monitoring of active users for planning of outage window
- Uploading changes
- Switching system to maintenance mode
- Applying the changes, cache flush,
- Restart and memory initialization
- Smoke tests
- Switching to normal mode
- Log record of updated version to planning system

It is crucial to have a detailed overview of deployed applications and their version of overall environments where they run, especially when the deployment is performed automatically with a support tool.

When a new version is deployed, the responsible person should pay increased attention to the error occurrence in logs and other monitoring spots. The update process needs to be ready for reverse direction and run also in the unexpected case needed for downgrading the version.

Run

The requirements for the lifecycle phase vary highly in dependencies for application complexity, technological solution and agreed service level of support. Some small projects running as a web app in a public cloud usually do not need attention and basic monitoring is possible with cloud-native tools. On the other hand, large enterprise proprietary software running in private clouds or self-managed servers usually needs a great deal of attention and well-made agreement defining the level of support (SLA), downtime allowance timer, connection to control and service desk for handling user tickets and many others with a growing complexity in time of usage.

List of activities (Vaněk 2017) related to regular app maintenance:

- Hosting hardware maintenance

- Software operation system maintenance
- App runtime environment administration
- Data backup and restore services
- Network and communication monitoring
- Security and protection service
- Performance scaling
- Monitoring

Monitoring

Checking the current status and running condition is the last part of DevOps interest in development process, operations team being fully responsible for those activities in a production environment. System monitoring is able to collect information from various sources over the application, from information records of the usage to an utilization of hosting system resources or other statistics. (Effective DevOps 2016)

There are many reasons why to collect and conduct analysis over information from a running system. The first usage of debugging and running logs is to solve error states, fix the bugs or some other way to improve unexpected behavior. A more advanced approach is to look for potential improvements independently, such as time utilization and peak points. (Vaněk 2017)

From DevOps point of view, this step is about providing the accessible feedback between a running application and developers in form of logs, or any other debug resources. The feedback loop leads to the development of better quality and performance of a software product. A tool providing the monitoring can also report differences between findings in the deployed versions. It is important to mention that the collected data has a privacy status and needs to be handled well according to the security and confidential guideline standards.

2.21. The future

The current situation in DevOps sector is evolving to be more linked with the business sector. There are more upcoming studies about how software release model and DevOps solutions impact the business decisions and influence the customer satisfaction.

However, technical issues are still challenging, in particular for more complicated or complex setup environments, and new tools will be introduced within time. The technical topic for the near future is the integration of tools that perform the steps mentioned in this thesis to a single DevOps tool instead of many independent single-purpose applications.

Finally, questions about culture and interpersonal relations stay as one of the major challenges for DevOps transformation process, as they are needed to change the current habits and the workflow, which often seems to be a painful process.

The currently starting projects should include DevOps pipeline from the start of the development and adoption process to the production and eventually, to the whole lifetime of a software product.

3 Practical implementation

3.1. Company and product introduction

The author got a position in Solteq, an international stock market company in a branch located in Jyväskylä. The company was founded in Finland and operates in Scandinavian countries for most part. Solteq delivers digital services to their clients who are mostly from the e-commerce sector, including some services of digital agency. The author worked in the business unit which is not entirely part of e-commerce sector since his team was a previously separated company recently acquired by Solteq.

The software solution that was in a background of the assignment for the thesis was a customer relationship management software for energy companies' sector. CRM is used for managing the records and requests of customers, putting them to process flow. The main goal of the software developed by the closest team is a tool for ordering urban network connections (such as water and electricity supply) via web-based form and to deliver an order for the service desks of construction and energy companies.

3.2. Current state of development

There was no automatic deployment process when the author joined the team of developers: however, the atmosphere inside the team seemed to be prepared for that step with the support of management. The procedure used for deployment of the application's new version was fully manual in terms of steps sequence. There was an exception of build and post process script which were in operation and which became the starting point of further automatization and expansion.

3.3. Target state

The minimum viable solution request was for an automatization of preparation software package and its deployment to an internal test server. The goal was to remove the human factor from the process and to allow nightly builds after working day to check the progress in terms of continuous integration. Eventually, the goal was

to try the effects of new features or changes in the running application, which could not be done without running it.

The planned wider benefit was also in sending notifications to developers, if the increment integration would not be successful and if the software product would not meet the quality standards. Later, the tools could perform the same process of an update on the customers' test environments to provide installation of the requested version and software on demand via a self-service portal.

3.4. Development process

The first step was to understand the original, previously used script handling build call, moving configuration files and producing an archive file. That script was written in an archaic style for a command line shell with hardcode paths and values. After getting familiar with that, the author suggested that the script should be replaced by a modern PowerShell script meeting the standards for parametrization by design and safe access standards with implemented exceptions. This was later approved of and decided by other technicians, and it was a crucial part in terms of learning new scripting constructions.

The following step was to learn how to use continuous integration tool Jenkins which was selected by a mentor after discussion with the manager and technical architect. Jenkins allows by default to use two ways of a definition of automatization jobs. The first one is called a freestyle job, and it is defined in the graphical web interface as a sequence of commands. This method is easy to understand, the flow is simple to design, and it allows fast job construction. Its disadvantage is in the storing process inside web application without a possibility of versioning or safe replication. For the purposes of the team, the best way was to use pipeline job, which is a declarative definition of the flow and uses for some cases scripted pipeline commands blocks by Groovy syntax script. That single file is stored inside the versioning system as the software product itself, which brings mentioned benefits such as change trackability.

Pipeline script was a completely new technology for the team, the author had to the learn syntax and use it on his own because there was no any other expert in the business unit working with that specified technology. One of the key construction patterns of the pipeline was to separate performing logic from flow logic instead of

writing just one script doing everything. Practically, that design separates commands executing calls on hosting system to external PowerShell file, and the pipeline calls that file with parameters from the Groovy script. Basically, Groovy script hosts a flow logic of the process. The other scripts perform the actual work. This solution provides control and simplicity over automatization process part and complies with good design standards.

3.5. Architecture of automatization

Automatization process is defined in the main job file with JenkinsFile extension stored in the Git repository with the application itself. The main job contains stages (described later in a chapter 3.6) executing related PowerShell scripts with correct values and paths. Those values receive the script from its own parameters coming from higher upstream job.

The upstream job is also groovy script with a declarative definition. It is a one-stage job holding values only for a specific customer product and deployment. It also includes paths to store files over the test server and passes credentials for remote locations. The upstream job is also stored in the Git repository and it is customer specific by principle. Developers can change only the end file based on their customer's domain knowledge. The downstream job is shared, and its modifications are not allowed to people who do not want to change the build process for every customer. Figure 19 clarify organization of described jobs.

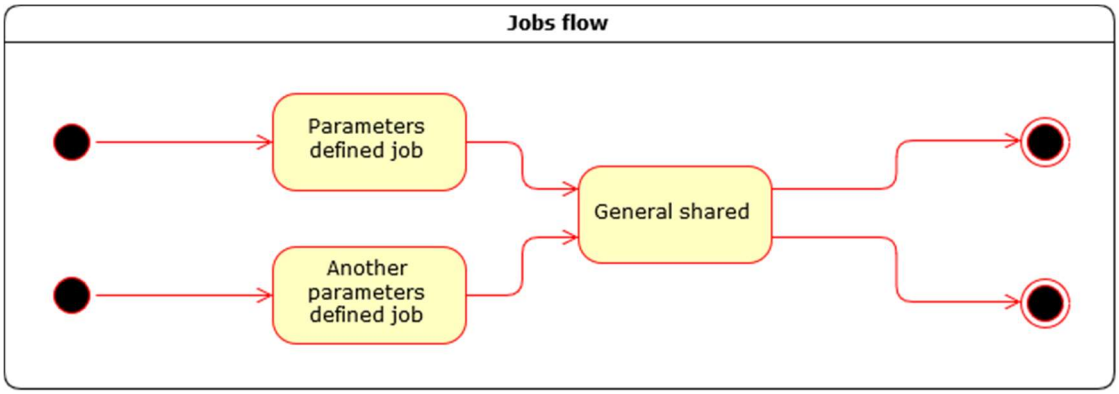

Figure 19 Organization of upstream and downstream jobs

The described pattern of design works for the team's purposes as in all the instances the software for specific customers still share the same build process. Shared process

is a great advantage, because duplication of build procedures could bring inconsistency over delivering process. The described solution allowed having customer specific build and package production without redundancy in a single commands' definitions. The disadvantage lies in more complicated customization; however, it is allowed by a special script which can execute out of a standard requirement at the end of the build process. The team uses the mentioned special procedure for one customer and one use case which is evaluated based on values in input parameters. The current state of development described previously is the output of one-year evolution of the solution.

3.6. Phases of creation

Preparation

This phase contains definitions of parameters needed for a build such as customer code, Git branch name, path to store files and around 20 more variables. A defined timer for a build, set execution node group and job name for internal Jenkins identification are also there. This phase provides a Git checkout; thus, after the run all necessary files are ready in the workspace folder to be processed.

Integration

Prebuild

The destination environment is prepared by cleaning the location where the old solution could be stored based on the path received via parameter. The folder structure is recreated after the cleaning. The next step is to check if the files given to build fulfill the requested version from Git and commit hash. If they do not, the build is terminated. This construction is occasionally used to avoid the wrong build from unsuitable resources.

When the check is done, the script testes the availability of tools needed specifically: TypeScript, Node.js, and NuGet. All of them are executed with specific subcommands and options in the next step. Specifically, it was challenging to work with npm packages and Gulp for Node.js, which have complicated integration with hosting operation system. The documentation of the integration steps was a part of development for reusability. This stage ensures at the end the presence of third-party dependency libraries.

Build

The build itself starts with checking the location of MSBuild executable file from an installed resource on the hosting system. The following step is to execute the build with all parameters and constructions. These commands were researched previously and prior to DevOps practices, and were implemented into software product via pubxml release profiles to provide maximal performance and features. It was specifically challenging to isolate the concurrent builds on the same host machine and define the external delivery location.

At this stage, all the logs are also collected which are evaluated based on a result of the build to provide feedback in case of failure of code integration. There is also a developer feature which attempts to learn the details about defective solution increments via Git versioning system. Solution build is the main part of converting source code into executable binaries.

Unit testing

The first quality testing mechanisms coming from developers are unit tests, and this stage executes a predefined list of unit tests and produces soft warnings or failure notification when necessary conditions are not fulfilled. The execution of unit tests is very similar to the execution of build, which means that it contains various input values controlling the testing process.

Post build

After-build script is there to finish the preparation of files for installation package. Firstly, it removes unnecessary logs and temporary files from destination location. Secondly, it executes gulp publish command, which then replaces references in used JavaScript files or stylesheets into final HTML markup tags because the solution has web page character. Thirdly, the script renames the default configuration files for IIS web server to templates one, because all existing installations have their own specific file. A new installation always needs to be manually edited. A procedure renames this configuration file to keep the sample file present yet avoiding unwanted rewriting during the file structure update. The specific procedure is also used to remove sample customer data from package’s folders and prepare readme file for package identification.

Deployment

Website

When build process is completed by preparing the package, it is then time for updating the test server environment where dynamic tests of functionality can be performed. Deployment to any server should be a separated job from the build job; however, it was not developed like that yet since it was not a part of the primary request. The team needed the local update of the main test server in the first place, later a backup test server was placed, and it required updates as well. This phase describes the update on a local device.

The local deployment process is invoked from JenkinsFile by Groovy shell but performed by PowerShell script with Import-Module WebAdministration for managing the IIS server configuration. That module works only in an x64 mode. The PowerShell console for 64 bit architecture mode requires explicit initialization by calling correct binaries from the operating system. Working with IIS requires the usage of full URI which is assembled in the script from parameters and processed per each web application name. The script solves the previous existence of deployment by checking the configuration or recreating web application site and web pool when selected. First, IIS ApplicationPool is created by calling Set-ItemProperty command and later the website is created by calling New-WebSite command. There is also a need for specifying the list of parameters such as address binding, physical path, host header, encryption settings and linkage with ApplicationPool.

The new developer version also requires an update of support database by data migration in case of a change in structure and web files from the build. These steps are performed while the supporting Windows service is stopped, alongside an internal scheduler, IIS sites and pools. Named services have timeout and their shutdown needs to be verified after sending the interrupt signal. When the entire application per customer is securely stopped, the update can be performed; otherwise there are troubles with the file locking. The update is made by copying files from build to the location of IIS root path to customer specific subfolder. When this is performed successfully, the external DB migration is executed from the script which is waiting for result code. The last step of the IIS installation/updating process is to start all the stopped resources such as websites, web pools, and support services in reversed order.

Before testing the availability of websites, there is a need to check network availability of URL by testing DNS records and to provide warning if it is not available. The backup testing server cannot use this method as the DNS records point to the main server; hence, in that case, Windows hosts file is modified to ensure the availability of other than DNS sourced web pages.

Every web application in this process has a specific configuration, including connection string for database and other installation specific settings as mentioned earlier. During this installation it is essential not to overwrite web.config file because in this case, the configuration will be lost. Since there are many disaster scenarios like this, the files before deployment process are copied to an external location in a cloud and achieved. This basic protection helps to keep the files in a safe place, because the configuration is not saved in the versioning system for an obvious reason.

Installation archive

Installation packages for external installation are created outside company test environment, such as on customer's server. Originally, they should be created before the internal update but due to the flow and time optimization the package is done right after without any affects. The package is prepared by archiving built binary files with libraries that are cleaned from logs and configuration files. Part of the process is the creation of info readme file which holds information about the builds itself like version, Git commit hash, customer code, and others. When the package is done by 7-zip console archive tool, the package is saved on a cloud-based network drive from where it can be used for various installations.

Notifications

The concept of continuous integration was thoroughly presented during the theoretical part of the thesis. Daily feedback for developers from Jenkins CI tool is essential. It is delivered through result notifications. Before the implementation, it was a deeply discussed topic which channel to use for informing people, and eventually Teams from Microsoft which supports 3rd party integration was chosen. Unfortunately, an original plugin for Jenkins messages did not meet the requirements. Because of that, the custom format of messages was developed. The project's build notifications are informing about build failures and additionally informing about the details such as related Git commits and detailed descriptions of failures from the logs. Additionally,

a direct link is included for building a console for personal review. The whole message is sent to Microsoft Teams Hub URL via specifically formatted JSON. Inside Teams the user interface allows developers to make comments or discuss solutions similarly as in social networks.

The challenges in the development were to avoid false alarms and unnecessary notifications. Additionally, there were some troubles with obtaining last successful git commit hash for providing details about the changes, because internal system variables were unreliable and giving null values that were interrupting the script. False alarms were solved by calling notification script from post-action Jenkins pipeline in regression phase, which avoided false alarms and increased the satisfaction with the system. The elimination of the rest of problematic messages was reached by selection of building stages eligible to do the notification.

Quality testing

Software delivery and customer satisfaction are connected via the term quality. During the development of continuous integration and delivery chain, also a part of automatic testing was included. The first test is based on the basic opening website on the test server. There is much more to try before marking results are ready to ship. Since the product is very complex and manual testing is slow and unreliable, an automatic robot framework test process is used designed by quality assurance specialist from our team. The tests are executed in the same way as the other PowerShell scripts. There were extra difficulties in installation and documentation of testing tools support to the hosting operation system because the tests run natively.

One part of the quality testing is security checking. The review was implemented in basic form of testing 3rd party libraries. Separate penetration test and other tests are still under development. Developers and other engineers are still responsible for manual security testing for the major version according to company rules.

Console window report following in Figure 20 of previously described build stages.

Figure 20 Jenkins CI interface with stages and run time values of smooth build.

4 Conclusion

The research and practical implementation process showed that DevOps practices are mature enough to be used on a daily basis. This paper explained the roots of DevOps history, introduced the terminology and presented the philosophy of the field and its approaches.

There are many paths for a successful DevOps journey; yet, even more ways lead to failure. The key requirement is to have the management support for the necessary transformation of the current workflow instead of heavy technical background only and/or collection of supportive tools. In either way, the personal contact with other developers in the team is highly wanted and unavoidable, because there is a need to understand the new steps and their benefits.

The research results proved that the best way to start implementing automatization in integration and deployments is to identify and process the most internal operations closest to the software build and expand them in the direction of more complex and abstract steps.

An advanced development of product specific DevOps tool includes the stages of necessary steps that add extra value in the form of feedback or statistics for all involved participants. These parts are called static code analysis, security scanning, generating documentation, measuring and reporting. The named steps as standalone processes require extra work from developers, therefore they could be skipped or forgotten unless they are automatic. The procedures contained in listed steps were introduced in related chapters of this thesis.

The benefits of DevOps practices are visible with the regular use of them as they simplify the software delivery process. Additionally, the value of the solution can easily be added by implementing more features. The first receivers of those benefits are the developers; however, also the operation specialists and management gain from them. After all, a correctly done DevOps solution can support the decisions of business leaders and possibly improve the overall company revenue or other metrics.

During the internship, a complex automation system was developed. The process contained various stages, and in several cases, no straight guidelines was given for

proof of concept trial. Some issues were raised because it was the first extensive implementation of automatic deployment process in our business unit.

4.1 Limitations

A part that turned out to be specially challenging was to design the architecture for multi-customer environment. The documentation nor the articles covered the model that was related to the organization model of company's product. The first task was to create a solution for a single customer. However, concerning the multi-customer environment, there were problems with location of resources in Git branches and concurrent processes inside Jenkins tool, test server and MSBuild tool which had to be isolated. To get expertise knowledge in that area took significant amount of time.

Several refactorization in PowerShell and Groovy scripts were needed before the well working practices settled down. The naming convention also had to be changed as it related to getting knowledge about the actual project. The described processes were challenging and time consuming. The troubles with architecture and some technical issues could have been eliminated if there would have been internal or even external specialist to help.

The used scripting language Groovy for Jenkins declarative pipeline was lacking a good documentation. The DevOps scripts were launched from Git version system for trials, therefore the development was slow. On the other hand, the solution is robust and support the tracking of changes. The script seemed to fulfill the requirements of best practices for software development. This was visible when it was compared with the original script which included many hardcoded values, was not safe and protected the command calls.

4.2 Relevance and future research

The benefits of the process were appreciated after they were delivered to production, and the developers started to enjoy the solution of deploying the current version to the test environments. The automatization system was not used for straight customer updates during the internship because there is a need for a deep audit; however, the

delivery manager appreciated the introduced solution and enhanced the wish list of features.

There is still space to expand the process and to include more supportive tools and analysis. The lightweight version was requested e.g. for doing the automatic check of latest code increment. The gained knowledge allowed the author to deliver the requested functionality in a fast way. Also, the other changes and improvement requests could be done conveniently.

The company and the team leader of assigned team are in favor of continuing cooperation and extending the developed tool. In future, there is a possibility to extend the process by adding new features to it or to use them in completely new project. Additionally, it would be interesting to study the effects and productiveness caused by the implemented tool, based on the opinions and experiences of developers themselves.

During the internship the author learned how to implement DevOps ideas in practice. The scripting knowledge and usage of new technologies was learnt and rapidly extended. The long-term cooperation with the company was successful in terms of not just delivering a proof of concept but virtually to finish the development for a production environment and support it after the deployment. Additionally, the importance of soft skills was highlighted during the process, as the developed methods were presented and explained to colleagues as the part of integration process. These skills would be impossible to gain in closed theoretical environment.

References

Adam Bertram. Kaizen and Continuous Improvement Through DevOps [online] Accessed on 24.11.2018 Retrieved from https://biztechmagazine.com/article/2016/06/kaizen-and-continuous-improvement-through-devops

Aiello, B. & Sachs L. 2010. Configuration management best practices: practical methods that work in the real world. Upper Saddle River, NJ: Addison-Wesley, c2011. ISBN 978-032-1685-865.

Alexis Richardson. GitOps - Operations by Pull Request - weave.works [online] Accessed on 10.12.2018. Retrieved from https://www.weave.works/blog/gitops-operations-by-pull-request

Ali Raza. Puppet vs. Chef vs. Ansible vs. SaltStack - intigua.com [online] Accessed on 9.12.2018. Retrieved from https://www.intigua.com/blog/puppet-vs.-chef-vs.-ansible-vs.-saltstack

Charron, R. Harrington, H., Voehl, F. & Wiggin, H. 2015. The Lean Management systems Handbook. New York: Productivity Press. Accessed on 23.11.2018 Retrieved from https://doi.org/10.1201/b17201

Conference DevOps 2018 in Helsinki (13.-14.12.2018) [attendance] Details retrievable from https://devops2018.com/keynote-and-program-details/

DevOps Culture (Part 1) - IT Revolution [online]. Accessed on 11.11.2018 Retrieved from https://itrevolution.com/devops-culture-part-1/

DevOps Dictionary - omniti.com [online]. Accessed on 12.11.2018 Retrieved from http://devopsdictionary.com/wiki/CAMS

DevOps is for everyone - opensource.com [online] Accessed on 22.11.2018 Retrieved from https://opensource.com/article/18/11/how-non-engineer-got-devops

DevOps Platform guide – Eficode [brochure online] Accessed on 1.12.2018. Retrieved from https://www.eficode.com/hubfs/documents/eficode_platform_www_en_280317.pdf?hsLang=en

DevSecOps: early, everywhere, at scale – Sonatype [eBook online] Accessed on 20.12.2018. Retrieved from https://fr.sonatype.com/hubfs/eBooks/Ebook-%20DevSecOps_V.2.pdf

Elisa: What's Behind DevOps the Buzzword? – Eficode [media picture online] Accessed on 12.12.2018. Retrieved from https://www.eficode.com/blog/elisa-whats-behind-devops-the-buzzword

Embrace DevOps - chef.io Accessed on 26.11.2018 Retrieved from https://pages.chef.io/rs/255-VFB-268/images/EmbraceDevops.pdf

Gene Kim. Agile Operations and the Three Ways: Insights from DevOps Experts. Accessed on 1.9.2018. Retrieved from https://www.ca.com/content/dam/ca/us/files/ebook/agile-operations-and-the-three-ways.pdf

Jennifer Davis & Ryn Daniels. 2016. Effective DevOps - Building a Culture of Collaboration, Affinity, and Tooling at Scale - O'Reilly Media [book], ISBN-13: 978-1491926307

Jordan Bach. 2017. Is NoOps the End of DevOps? Think Again [Infographic online] Accessed on 11.12.2018. Retrieved from https://blog.appdynamics.com/engineering/is-noops-the-end-of-devops-think-again/

Kubernetes vs. Docker: What Does It Really Mean? – sumologic [online] Accessed on 9.12.2018. Retrieved from https://www.sumologic.com/blog/devops/kubernetes-vs-docker/

Mesnita, G. & Dumitru, O. 2006. Information Systems Documentation - Another Problem for Project Management [online] Accessed on 20.12.2018. Retrieved from https://www.researchgate.net/publication/228217510_The_Information_Systems_Documentation_-_Another_Problem_for_Project_Management

Michael Ernst. Version control concepts and best practices. Accessed on 30.11.2018 Retrieved from https://homes.cs.washington.edu/~mernst/advice/version-control.html

Pinterest technology picture by neo [online media] Accessed on 10.11.2018. Retrieved from https://pinterest.com/pin/551620654353293504/

Rugged DevOps vs DevSecOps: What's the Difference? Injecting security into the continuous delivery process [online] Accessed on 7.12.2018. Retrieved from https://www.sumologic.com/devops/devsecops-rugged-devops/

State of DevOps Report. Puppet [online]. IT Revolution Press, 2013 Accessed on 5.10.2018 Retrieved from https://puppet.com/resources/whitepaper/2013-state-devops-report

State of DevOps Report. Puppet [online]. IT Revolution Press, 2018 Accessed on 5.10.2018 Retrieved from https://puppet.com/resources/whitepaper/state-of-devops-report/thank-you

Understanding Continuous Integration - medium.com/@mohan08p [online] Accessed on 22.11.2018 Retrieved from https://medium.com/@mohan08p/understanding-continuous-integration-ci-and-continuous-delivery-cd-part-iv-4ab602c1f011

Václav Vaněk. 2017. DevOps Practices and Their Usage in Software Development [online] Accessed on 2.9.2018. Retrieved from http://hdl.handle.net/10084/119166

You Build It, You Secure It: Introduction to DevSecOps – SumoLogic [online presentation] Accessed on 19.12.2018. Retrieved from https://www.slideshare.net/Sumo_Logic/you-build-it-you-secure-it-introduction-to-devsecops