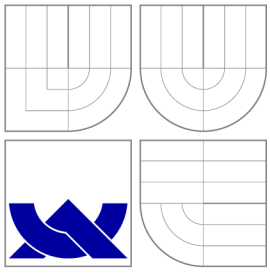
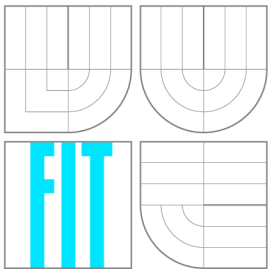


BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

A PROBE OF BUILD AUTOMATION FOR CONTINUOUS INTEGRATION

PŘÍPAD UŽITÍ AUTOMATIZACE SESTAVENÍ PROJEKTŮ V METODICE PRŮBĚŽNÉ INTEGRACE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR
AUTOR PRÁCE
MARTIN KAČMARČÍK

SUPERVISOR
VEDOUCÍ PRÁCE
Ing. ALEŠ SMRČKA, Ph.D.

BRNO 2016

Zadání bakalářské práce/18216/2015/xkacma03

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Kačmarčík Martin**
Obor: Informační technologie
Téma: **Případ užití automatizace sestavení projektů v metodice průběžné integrace**
A Probe of Build Automation for Continuous Integration
Kategorie: Softwarové inženýrství

Pokyny:

1. Nastudujte metodiku průběžné integrace. Nastudujte nástroje nejčastěji používané při automatizaci sestavení projektů. Seznamte se s nástroji Jenkins CI pro průběžnou integraci a Foreman pro správu virtuálních serverů.
2. Analyzujte reálné případy automatizovaného sestavení projektů. Navrhněte systém pro automatizaci sestavení projektů v rámci průběžného vývoje. Uvažujte různé druhy vstupních repozitářů zdrojových kódů, různé možnosti testování a různé běhové prostředí pro sestavení projektů.
3. Implementujte vámi navržený systém. Inspirujte se nástroji Jenkins a Foreman.
4. Ověřte funkčnost vašeho systému na různých případech použití a různých konfiguracích běhového prostředí.

Literatura:

1. Duvall P. M., Matyas S., Glover A.: Continuous Integration: Improving Software Quality and Reducing Risk; Addison-Wesley Professional; ISBN 9780321336385.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrčka Aleš, Ing., Ph.D.**, UITS FIT VUT
Konzultant: Mindru Vaeceslav, RHcz
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstract

This bachelor thesis deals with a probe of build automation for continuous integration. The problem is solved by integrating existing systems such as Foreman, Jenkins and GitLab into a single system. The contribution of this thesis is a functional solution that is ready to be used in low- and middle-scale projects.

Abstrakt

Bakalářská práce se zabývá využitím automatizovaného sestavení projektů s metodikou průběžné integrace. Problém je řešen integrováním již existujících systémů, jako například Jenkins, Foreman a GitLab, do jednoho výsledného systému. Přínosem této práce je existující řešení, které je možné využít pro vývoj menších až středních projektů.

Keywords

Build automation, continuous integration, Foreman, Jenkins, CI, automation.

Klíčová slova

Automatizované sestavení projektů, metodika průběžné integrace, Foreman, Jenkins, CI, automatizace.

Reference

KAČMARČÍK, Martin. *A Probe of Build Automation for Continuous Integration.* Brno, 2016. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Smrčka Aleš.

Rozšířený abstrakt

Tato práce se zabývá využitím automatického sestavení projektů v metodice průběžné integrace s využitím nástrojů Jenkins a Foreman za účelem ověření konceptu.
V následujících kapitolách práce je rozebrána metodika průběžné integrace, konkrétně její účel, výhody, nevýhody, způsoby jakými se tato metodika praktikuje. Poté jsou probrány různé nástroje pro průběžnou integraci a následně důkladněji rozebrány nástroje Jenkins a Foreman. Po představení metodiky průběžné integrace je popsáno automatické sestavení projektů, nástroje k tomu využívané, jejich porovnání a také jsou analyzovány realné případy užití automatického sestavování projektů.
Pro ověření konceptu byl navržen systém, který spojuje verzovací systém typu git, nástroj Jenkins pro průběžnou integraci a nástroj pro správu virtuálních serverů Foreman. Původně bylo zamýšleno použití virtuálních strojů pro simulaci běhových prostředí, ale kvůli ušetření času byl použit nástroj Docker, který viditelně snížil čas potřebný pro inicializaci testovacího prostředí. Za využití virtuálních strojů propojených interní sítí byl tento systém nainstalován a na pěti různých případech užití byla ověřena funkčnost celého systému. Výsledkem práce jsou tedy tři virtuální stroje, pět projektů, jejich testy, konfigurační soubory pro nástroj Docker a konfigurační skripty. Výsledky práce budou dále použity pro implementaci podobného systému ve firmě Red Hat.

A Probe of Build Automation for Continuous Integration

Declaration

I hereby declare, that this paper is my original authorial work, which I have worked out on my own under the co-leadership of Ing. Aleš Smrčka Ph.D and Vaeceslav Mindru from the company Red Hat Czech. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source. In the thesis, a plural "we" has been used simply for the readability.

.....
Martin Kačmarčík
May 15, 2016

Acknowledgements

I would like to thank to my supervisor Ing. Aleš Smrčka Ph.D. for invaluable advice and positive attitude, which he demonstrated throughout the whole process of creating this work. Next, I would like to thank to Vaeceslav Mindru from Red Hat Czech, whose impressive knowledge was immensely beneficial to this work. Lastly, I would like to thank to Philip Gammon for correction of this thesis.

Contents

Chapter 1

Introduction

Nowadays, projects usually consist of several components that cooperate to gather. Each component is often made by a different developer. This means that the developers need an agreement how their parts should work with each other. It also means there is a huge space for errors. One of the ways to avoid these errors is to test the application properly and integrate as much as possible.
Testing is as important as the development itself. Projects these days are much larger than they used to be and that means much larger chance of making a mistake. Using continuous integration correctly, it is possible to reduce integration problems and develop cohesive software more rapidly. Rapid development allows to react on the changes in the market or demand from customers more flexibly. The more flexible the application is the more it can keep up with customers demands and react on trends and competition. Problems with integration can also become very serious ones. When integration is done after each component is finished, in huge projects, it can be severely hard to discover reason of bugs while they can be connected. With continuous integration practice it is possible to avoid these states of the integration by integrating more frequently.
Automation is extremely important within continuous integration methodics. Continuous integration is using automation to make the whole process of testing and integrating unattended. Following its methods, it is possible to create interconnected environment that is automated and self-contained. Everything should be done automatically to save time, reduce errors and support the development.
The goal of this thesis is to introduce the reader to continuous integration methodics, build automation and review the tools that are frequently used to practice continuous integration. Based on the gained knowledge gained, automatized system using continuous integration methodics and tools to check integration and quality of the current state of the software will be created. This system will be fully automatic—build automation, automated testing, supporting version control systems and different possibilities of running environments. The motivation behind this document is to create a proof of concept which demonstrates that this methodics can be effectively used within the Product and Technologies, Developer Operations - System Operations team in Red Hat.
In the Chapter 2, continuous integration will be introduced, defined and explained. In Chapter 3, tools to practice continuous integration will be reviewed. After that in Chapter 4 use cases of build automation will be analysed and system using build automation and continuous integration will be designed and described.

Chapter 2

Continuous Integration

Continuous integration is a practice in software engineering which helps us to integrate parts of the developed software. To be more precise, a definition from a famous article about continuous integration by Martin Fowler [10] will be used: „Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.“
According to Paul Duvall [9] that means that:

- All developers run private builds on their own workstations before committing their code to the version control repository to ensure that their changes do not break the integration build.
- Developers commit their code to a version control repository at least once a day.
- Integration builds occur several times a day on a separate build machine.
- 100% of tests must pass for every build
- A product is generated (e.g., WAR, assembly, executable, etc.) that can be functionally tested.
- Fixing broken builds is of the highest priority.
- Some developers review reports generated by the build, such as coding standards and dependency analysis reports, to seek areas for improvement.

2.1 Purpose of Continuous Integration

Now when continuous integration is defined, it would be proper to define what for continuous integration is, what its purpose is. As Martin Fowler [10] states, the main purpose of continuous integration is to increase quality of software and reduce risk. This is achieved by several aspects described in following two subsections.

2.1.1 Knowing the State of the Project

Using deferred integration puts developers practically in a blind spot. They are not aware of which part of integration is working or which is not, if there are bugs and how far they are in the process. On the other hand, using the continuous integration methodics, developers always know the state of the project, what is working and what is not, what are the issues in the integration and by that they greatly reduce the blind spots presented in deferred integration.

2.1.2 Reducing Number of Bugs

Continuous integration does not get rid of bugs by itself. It makes them easier to be found and removed. Because of the minimal intervals between integrations of code, if using the proper continuous integration practice, previous integration is without bugs and therefore developer knows that in the changed code there has to be the bug that newly occurred which makes the bug easier to be discovered. As a result, projects using continuous integration have significantly less bugs than the ones which do not. However, this is directly depended on project's testing suite and team's discipline.

2.2 Advantages of Continuous Integration

Now that the purpose is defined we can summarize some of the advantages of continuous integration.

- Reducing risk by integrating frequently—Risk is an unexpected event and it has a potential to negatively impact the project.
- Analysis of the project state—Direction of the project can be effectively changed during the development process based on the continuous analysis which makes the project more agile.
- Errors discovered at the beginning—not only it will make bugs easier to be fixed; it also helps the team psychologically. People find easier to fix bugs when there are two of them rather than hundreds of them.
- Validating assumptions—one of the biggest advantage of the continuous integration is that it can uncover wrong assumptions before the late project state which can save energy of the team as well as the final project cost.
- Automating processes—even though it takes some time to set these automation processes, if done and designed correctly the automation will save more time in the future. After the automation is implemented it reduces the overhead work. Computer is fast and usually does not make mistakes, it just follow the instructions.
- Continuous integration makes it easier to do changes—if continuous integration is used, it is easy to implement prototypes of new features and see the impact of the feature immediately.

2.3 Disadvantages of Continuous Integration

In general, continuous integration has far more advantages then disadvantages. One of the disadvantages of continuous integration is that it can be an overhead of time and resources needed to implement, integrate and maintain the continuous integration tools and tests. Another disadvantage is that it increases overall project costs. It also slightly decreases the agility of the project when enforced and if it is designed and implemented wrongly it might become a major single point of failure[1].

2.4 Continuous Integration Practices

In this section common methods used in continuous integration will be reviewed. As Jez Humble and David Farley stated in their book [12]: „Continuous integration is a practice, not a tool, and it depends upon discipline to make it effective. Keeping a continuous integration system operating, particularly when you are dealing with large and complex CI systems, requires a significant degree of discipline from the development team as a whole.“ Although, this subsection covers common practices it does not cover them all.
In the Figure 2.1, continuous integration system and its workflow is demonstrated. After developers commit changes to version control repository, continuous integration server detects these changes and executes building and testing process. After the previous steps are done the continuous integration system gives feedback to the developers with results of the process.

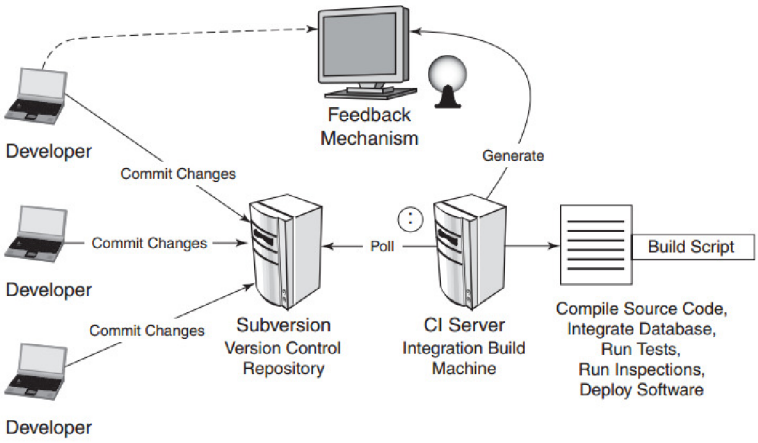


Figure 2.1: Scheme [9] illustrating continuous integration system and workflow.

In the following subsections, common practice will be stated in their logical order.

2.4.1 Agreement in the Team

Team should agree to push their changes frequently. Also, it should be agreed that fixing found mistakes is the highest priority. If these errors are ignored the whole continuous integration idea is going to be wasted. Therefore, it is strongly needed to have discipline within the team and firmly stick to basic principles of continuous integration. The founder

[1] A single point of failure is a part of a system that, if it fails, will stop the entire system from working.[8]

of javaranch.com, Kathy Sierra, said in her blog [20], „There's a big difference between saying, 'Eat an apple a day' and actually eating the apple." Paul Duvall added to this sentence: The same goes for following fundamental practices on a software project. [9]

2.4.2 Version Control

Using a version control system while working on a project is today a common practice. Version control systems can unite source codes; backup data and allows iterating through stages of the project. As Paul Duvall states: The purpose of a version control repository is to manage changes to a source and other software assets (such as documentation) using a controlled access repository [9]. All the source code is therefore accessible from one main location. Every important file from the project should be in this repository. It is good practice to use a version control system while using the continuous integration. Version control system allows automatically react on changes in source code of the project and take further steps of the integration.

2.4.3 Code Inspection

Before the code is built, it is a good practice to review the code for disallowed elements like disposed functions or packages. This can save time in case of human error while developer does not have to wait for a build that will fail inevitably. In case of a long process before result is outputted this can drastically save time. Code inspection is just another case of testing. In this thesis, we assume that code inspection is stated before build automation to keep up the logical order of the steps.

2.4.4 Build Automation

After the code is inspected build automation phase is usually triggered. As Spinellis states in his book [23]: „in summary, automation serves three purposes: it documents the processes, speeds up the tasks and eliminates human errors and forgotten steps." It is also test of its own. It tests if the software even builds. The typical input is source code, dependencies, libraries and configurations. As a result of build automation there is machine-executable form of the program which can be further tested. More about build automation tools can be found in Chapter 3.

2.4.5 Automated Tests and Self-testing

After the source code is pushed into a repository, continuous integration tool should ensure that automated test will run. Automated test discover bugs that were brought to the system within the commit. If any of the tests fails the author should revert the commit and work on correction. Automated tests are usually much faster and reliable then manual testing and are one of the basic principles of the continuous integration. Boris Beizer [14] reports that error rate in manual testing is comparable to the bug rate in the code being tested. He estimates that in manual testing; only about half of all the tests are executed properly. On the other hand, automated tests require a great focus on maintainability.
Several kinds of test are applied. The most common are:

- Regression tests. Regression testing is the process of testing changes to computer programs to make sure that older programming still works with the new changes [19].

- Feature tests[2]. Feature tests mainly target newly implemented component that meaningfully modifies the functionality of the tested software.
- Unit tests target testing each unit of the developed software. It usually does not test program as a whole but only the unit itself.
- System tests. System testing is a type of black-box testing (does not peer into the internal structure of the software). It mainly feeds software with input and then examines resulting output.

2.4.6 Testing in Environment Simulating Production

Production environment can have different settings than the environment developer is developing in. To avoid unnecessary mistakes, like different compiler versions, the new update should always be tested on the environment simulating the production one. If you test in a different environment, every difference results in a risk that what happens under test will not happen in the production [10].

2.4.7 Automated Deployment

It is important to automate steps, which deploys the product to any environment [16]. Biggest advantages of auto deployment [22] are:

- Deployments become much less error-prone and are much more repeatable. Important steps in the release can be accidentally missed, people do errors. Automation will reduce human error factor.
- Anyone in the team can deploy software. With automation deployment process everyone can deploy the software and the process is not depended on just few people who know how to do it.
- Engineers spend their time developing software. Engineers use the time that would be otherwise used to perform and validate manual deployment process for something more productive.
- Deploying to somewhere new is not a headache. Automated deployments can be reconfigured when the software is deployed to different environment. It is a simple matter of changing the configuration file.
- It is easier to release more frequently. Automated deployments have a lower overhead. A release process with a low overhead is one that can be repeated frequently. As said before, frequent releasing is one of the fundamental parts of continuous integration.

2.5 Continuous Integration Tools

Continuous integration tools support the methodics and help to deliver software faster by decreasing integration times. There are many tools that help to practice continuous integration. As stated in [17], the process of choosing the automation software for CI is a

[2]More about feature tests can be found on following URL: http://www.tutorialspoint.com/software_testing_dictionary/feature_testing.htm.

matter of finding the tools that fit best to the development environment and processes. Therefore analysis of the system that will use continuous integration is needed and we can never say that certain tools will always be the right one. In Table A.1, some of the most common tools used within continuous integration methodics can be seen.

2.5.1 Comparison of Continuous Integration Tools

In the Table A.1, it is demonstrated that modern integration tools offer multiple option of integration as well as great amount of builders on both Microsoft Windows and GNU/Linux platforms. Users can also choose between proprietary and open-source software. Most of the tools are Servlet Containers—although exceptions can be found—in the Table A.1 it is GitLab CI that runs on Ruby interpret. Notifications can be delivered in many ways. The most standard is email but except GitLab CI all the tools offer multiple other option of notification delivery. Also, very considerable aspect of the tool is if it is part of other system or standalone application. For example Jenkins needs to be installed on server separately which add overhead, while GitLab CI tool is integrated in GitLab versioning system management and saves the overhead. On the other hand, Jenkins offers much more functionality than GitLab CI tool. In practice, Gitlab CI usually work with Jenkins together. To summarize: the continuous integration tool should always be chosen according to needs of the project/environment.

Table A.1: Common tools used in continuous integration [26].

Name	Platform	License	Builders	Other builders	Notification	Integration
Bamboo	Servlet Container	Proprietary	MSBuild, NAnt, Visual Studio, Ant, Maven	Custom Script, Bash, Open-Make Software Meister, Xcode, Phing	XMPP, Google Talk, E-mail, RSS, Remote API	IntelliJ IDEA, Eclipse, Visual Studio, FishEye, Crowd, JIRA, Clover
GitLab CI	Ruby	MIT	Many	Cross-platform command-line	E-Mail	HTTP API (JSON)
Jenkins	Servlet Container	Creative Common-sand MIT	MSBuild, NAnt, Ant, Maven 2, Kundo	Cmake, Gant, Gradle, Grails, Phing, Rake, Ruby, SCons, Python, Shell and Command Line	Android, email, Google Calendar, IRC, XMPP, RSS, Twitter	Eclipse, IntelliJ IDEA, NetBeans, Bugzilla, Google Code, JIRA, Bit-bucket, Trac...
TeamCity	Servlet Container	Proprietary	MSBuild, NAnt, Visual Studio , .NET, Ant, Maven 2/3, IDEA, Gradle	Rake, FxCop, Command Line	E-mail, XMPP, RSS, IDE, SysTray	Eclipse, Visual Studio, IntelliJ IDEA, RubyMine, PyCharm, Php-Storm, Web-Storm, Jetbrains Youtrack, JIRA, Bugzilla...

2.6 Tools Used in this Thesis

In this section we will more closely review the tools that will be used to design the system that uses continuous integration and build automation. This system is the main results of this thesis proving the integration works. In this thesis, Jenkins will be used as continuous integration tool and Foreman as provisioning manager. The reason why these tools were chosen is: both tools are open-source and offer great portfolio of functions. Open source is mainly important for the ideology of Red Hat, for which environment the system in this thesis is aimed. In the Table A.1, it is demonstrated that Jenkins offers great amount of integration, builders and ways to deliver notifications compared to other candidates. Also, Jenkins is actively maintained by its community and professional developers and offers impressive amounts of extensions using plugins.

2.6.1 Jenkins

As Smart states [21], Jenkins, originally Hudson, is an open-source continuous integration tool written in Java. Boasting a dominant market share, Jenkins is used by teams of all sizes, for projects in a wide variety of languages and technologies, including .NET, Ruby, Groovy, Grails, PHP and more, as well as Java. Jenkins is also easy to use. Deployed as an web application it has intuitive user interface and very low learning curve.
Jenkins needs several prerequisites before it can run. These prerequisites are Java Runtime Environment, version control system set which Jenkins will monitor and of course a web browser that can display Jenkins GUI.

Short History of Jenkins

According to Smart [21], Jenkins is the result of developer named Kohsuke Kawaguchi who started to develop, that time Hudson, in late 2004 while working at Sun. As time passed by it was integrated by more and more teams in Sun for their own projects. Sun realized value of the tool and asked Kawaguchi to work on Hudson full-time. In 2009, Oracle purchased Sun. Disagreements between Oracle and Hudson's developer community resulted into forking the original Hudson project into new one named Jenkins.

Key Features of Jenkins Tool

One of the key feature of Jenkins tool is its ability to monitor one or more repositories for changes and react on these changes. Each monitoring setup is called a job. As stated in Smart's Jenkins the Definitive Guide book [21] - This may involve simply compiling user's source code and running his/her unit tests. Or user might want a build job to do other related tasks, such as running your integration tests, measuring code coverage or code quality metrics, generating technical documentation, or even deploying your application to a web server. A real project usually requires many separate but related build jobs. Example of Jenkins GUI with set jobs can be seen in Figure 2.2. In this thesis, Jenkins will be used as the primary continuous integration tool.

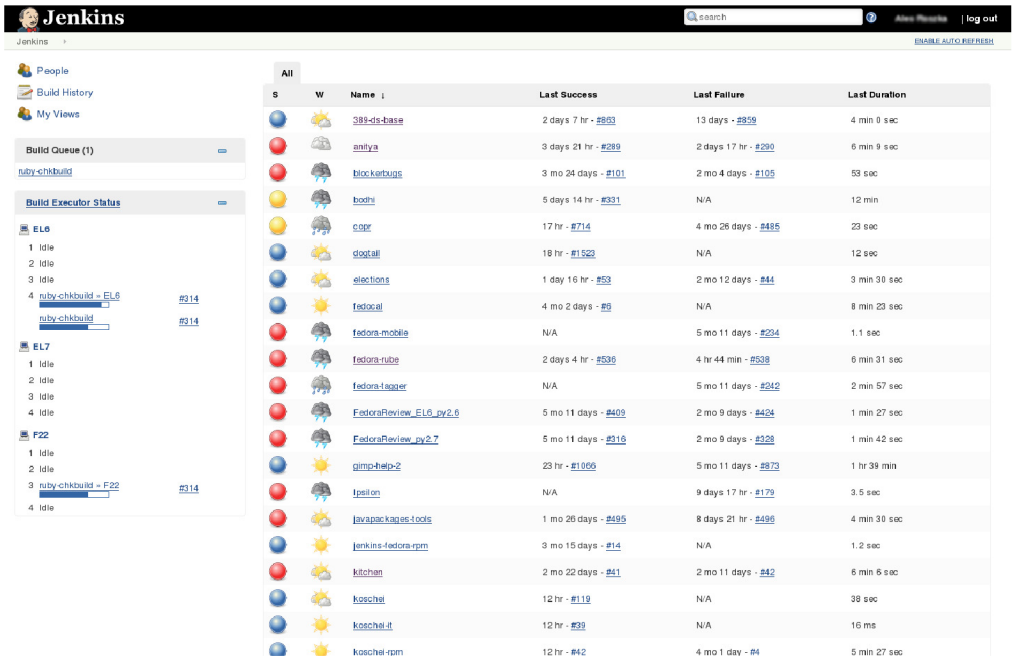


Figure 2.2: Jenkins GUI.

Based on official Jenkins wiki page [13], main features of Jenkins are:

- Easy installation. Jenkins just needs to be deployed. No additional install, no database.
- Easy configuration. Jenkins is fully configurable through its intuitive GUI.
- Rich plugin ecosystem. Jenkins integrates with virtually every SCM[3] or build tool that exists.
- Extensibility. Most parts of Jenkins can be easily modified to user's needs. It is also easy to create new Jenkins plugins.
- Distributed builds. Jenkins can distribute builds and test to multiple operating systems.

2.6.2 Foreman

Foreman is an open source project that helps system administrators manages servers throughout their life cycle, from provisioning and configuration to orchestration and monitoring. Using Puppet, Chef, Salt and Foreman's smart proxy architecture, you can easily automate repetitive tasks, quickly deploy applications and proactively manage change, both on-premise with VMs and bare-metal or in the cloud [3].
In Figure 2.3 Foreman's architecture can be seen. Foreman communicates with Smart proxies using Restful API. To communicate with the user it has its own Web API. Foreman uses database to store data and LDAP for user-management. Puppet provides configuration management for Foreman so it can manage servers in their virtual cloud ecosystem. In this thesis, Foreman will provide a tool for provisioning that will be used to create testing

[3]SCM stands for Software configuration management

environments for Jenkins jobs.

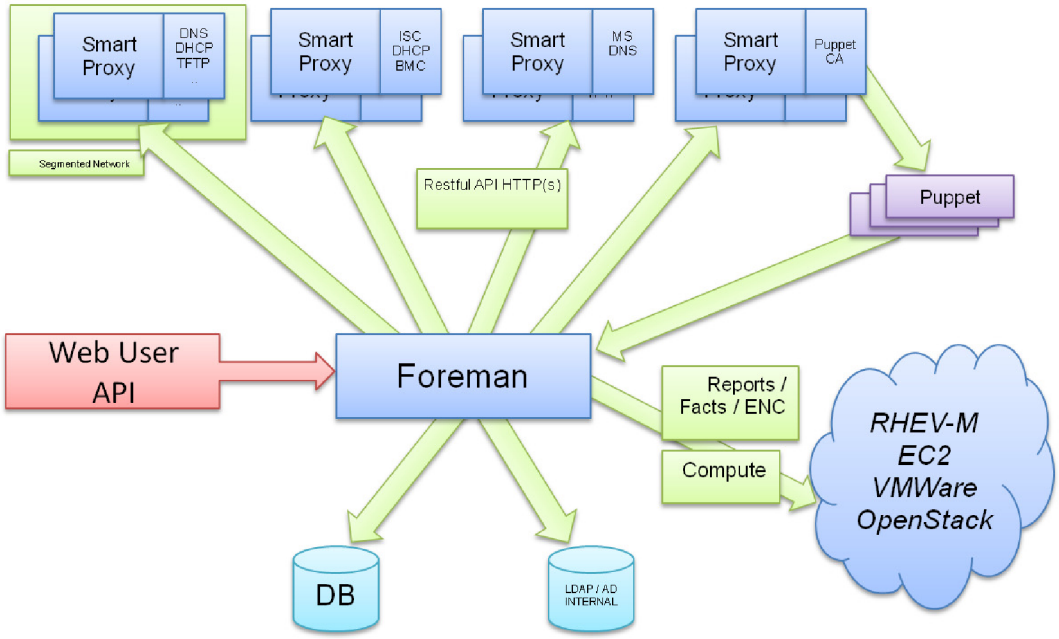


Figure 2.3: Foreman Architecture [3].

Chapter 3

Build Automation

Build automation is a process that results in software build and includes compiling the source code, automated testing, packing and deploying (even though it is not mandatory that all mentioned aspects must be present). Usually, a user creates a build script that will do all the parts of the procedure automatically which saves time and also reduces the human error factor. Using build automation tools to practice continuous integration methodics makes observing the principles of continuous integration markedly easier and without build automation continuous integration disadvantages may outweigh any advantages.

3.1 Build Automation Tools

Build automation tools are used to build the source code so that it can be further used for analysis/testing/deployment. The choice of the right build tool depends on the language and framework that the project is using. Two of the most commonly used build tools are make and Ant [23] and both will be described in this section. Build automation tools can also automatically resolve dependencies, which can be really useful when the dependency tree is complicated. There are also so called server-based build automation tools which is just synonym for continuous integration tools.

3.1.1 Make Utility

Make is one of the oldest build tools and was developed in 1977 at Bell Labs. In GNU Make manual it can be read that the make utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them [4]. The input for make utility is Makefile, a text file containing shell commands. In Robert Macklenburg's book [15] make is defined as a language for describing the relationships between source code, intermediate files, and executables. It also provides features to manage alternate configurations, implement reusable libraries of specifications, and parametrize processes with user-defined macros. It is mainly used for C/C++ projects. To get more information about make utility, please refer to the manual [4].

Summary - features

- Used mainly for C and C++ projects.
- Widely documented.

- Relatively simple - easy to use on basic level.
- A good performance.
- Cross-module dependencies.
- Does not provide enough tools for big projects.

3.1.2 Ant

Ant is with the Maven one of the most commonly used tool in Java projects [23]—although it can be used for C/C++ projects as well [1]—and was originally released by the Apache project in the year 2000 to overcome the portability issues related to make [17]. Ant is written in Java and uses XML documents to describe the build process. It has extensive documentation and can be found on official ant project website: http://ant.apache.org/.

Summary - features

- Used mainly for Java projects.
- Widely documented.
- Using XML language.
- Large control over the build automation.
- Is relatively hard to learn.
- Old.

3.1.3 Maven

One the formal definitions of Maven says [5]: „Maven is a project management tool which encompasses a project object model, a set of standards, a project lifecycle, a dependency management system, and logic for executing plugin goals at defined phases in a lifecycle. When you use Maven, you describe your project using a well-defined project object model, Maven can then apply cross-cutting logic from a set of shared (or custom) plugins.“ Simply put it takes the best from Ant and add project management features to it. It still can be used as a simple build tool. More about Maven can be found at the official webpage https://maven.apache.org/

Summary - features

- Used mainly for Java projects.
- Difficult to learn.
- Relatively complicated.
- Verbose and complex.
- Provides exceptional management over the project.
- Great integration with continuous integration (one line in Maven configuration file).

3.1.4 Gradle

Gradle aims to take best from both Ant and Maven projects. It tends to offers both flexibility and conventions for its users. Tim Berglund and Matthew McCullough describe Gradle in their book [7] as rich in features, but is not such an opinionated framework that it will fight you when you want to do things your own way. It offers conventions to those who want them, flexibility to those who need it, and a toolkit for turning that flexibility into domain-specific build standards that you can write on your own. More about Gradle can be found at its official website: http://gradle.org/.

Summary - features

- Modern dynamic tool.
- Great flexibility.
- Re-use source code.
- Multiple language support.
- Customize and configure option for the build process.
- Relatively complex - easy to learn, hard to master.

3.1.5 Summary

Selecting a build tool strongly depends on the requirements of the project. Each build tools has its own advantages and disadvantages and aims for different purposes. The main observed difference is language for which the tool is used. C and C++ community preferably uses Makefile and Java community uses Ant or Maven. Big projects can definitely benefit from options that newer build tools offers. On the other hand, learning curve can dramatically raise and for simple project it can become unnecessary overhead.

3.2 Analysis of Real Examples of the Build Automation using Continuous Integration

In this section, two concrete use cases of build automation and continuous integration are discussed to demonstrate real-life usage of build automation and continuous integration. Then, five other general examples are described and analysed. On these five examples, the whole concept of the system implemented in this thesis is verified.
The first concrete use case demonstrates how real software developing company uses continuous integration methodics and what are the benefits of using it. In this use case, we describe the process of practicing the methodics to improve their software and what are the exact steps of the process. Second use case demonstrates a company providing build automation and continuous integration options for its customers. Providing build automation is useful feature that makes company's product more attractive, saves customer's time and makes his/her project more agile.

3.2.1 Mozilla and Build Automation

Mozilla is a company mostly famous for its web browser Firefox. According to stats from StatCounter [24] in February 2016 Firefox had 16.53% of market share making it as one of the biggest web browser projects. Large projects like Mozilla Firefox cannot afford to make build process manually; the amount of time needed to build and test the browser on all operating systems and all different architectures would be unbearable. As for that, the whole process is—as it is written at official Mozilla website [6]—fully automated. The web page also states that these automation tools are specifically made to increase other teams productivity.
Mozilla is using the build automation on full scale and use it very similar way as the system that is designed in this thesis. Another source of information, beside Mozilla wiki [6], was also blog article [11] about Firefox's build automation revealing some internal specification of the process. The main component of Firefox's build automation is software called Buildbot. Buildbot is similar to described Jenkins tool and serves as a job scheduling system. Buildbot monitor a repository and react on push into the repository. A job is then scheduled and assigned to a slave. Mozilla operates a few master and a few thousand slaves and therefore the usage is high. After this point, machine need to be configured. Using Puppet for configuration and kickstart —and other unattended installation procedures for other operating systems—new machine is created. On this machine, tests are run and result is returned as a report. This whole process keeps Firefox at the consistent change. Firefox is open-source project with high community contribution. It is very important to have at least some control over Firefox state of consistency and therefore it is a necessary to use automation with this high number of changes in short intervals.

3.2.2 Docker and Build Automation

Docker is an open-source engine that automates the deployment of applications into containers [25]. These containers can be run under an operating system practically simulating the use case of virtual machines with little differences like using operating system's resources. Docker is considered as a new generation of virtualization [25].
Building a docker image is fairly simple. There are several default docker images that contains desired operating system in its default settings. Using `Dockerfile`—special Docker image description file—user defines attributes of the system and then using `docker build` command docker image is build.
As for the build automation, Docker offers automated builds on Docker Hub. As the official Docker wiki states [2]: „with Docker Hub you can build your images automatically from a build context stored in a repository. A build context is a `Dockerfile` and any files at specific location. For an automated build, the build context is a repository containing a `Dockerfile`.“ Automated builds have several advantages:

- Images built in this way are built exactly as specified.
- Repository is kept up-to-date with code changes automatically.
- Developer can focus on developing itself rather than “system administration“.

Docker demonstrates solid usage of build automation with no need of user to step into the building process.

3.2.3 Build Automation and Continuous Integration For Developers

This subsection will describe the use case of developing an application using build automation and continuous integration with five different configurations. With this use-case we worked the whole thesis, describing build automation and continuous integration from the developer and/or system administrator point of view. Simplified versions of all five configurations of the original use case are demonstrated in the designed system, which is later described in Chapter 4.
First use case represents a process of developing a Python script—that automatically download, install and configure Oracle Java—and testing its output. Second use case is represents a development of NodeJS web-server and testing its functionality. Third use case represents testing configuration of Apache web server and changes made to the configuration. Fourth use case represents developing a HTML, JavaScript and CSS based web page and testing its source code and dependencies. Fifth and the last use case represents developing a PHP back-end and testing compatibility with several PHP versions.
There can be other use cases of build automation for continuous integration like automatic deployment. Even though continuous integration and deployment meets in certain aspects, the topic of deployment is more covered in continuous delivery methodics which is not described in this thesis. More about continuous delivery can be found in the book Continuous Delivery [12].

Use Case 1: Python Script

In this use case we work with a Python script that downloads Oracle Java development kit when executed. The idea of the script is that based on arguments specifying version and update of Java as well as some company/customer specific requirements for configuration, script will download required Java version to the given path and set requested configuration settings or install providers etc.
The benefit of build automation is that after commit is pushed, the script is automatically interpreted, tests are run checking that the output and if all settings were applied correctly. This whole process is automatized using bash scripts. Continuous integration in this context is the process of notifying the continuous integration tool that new commit was pushed and running tests.
The benefit of the usage of build automation and continuous integration is that developer has immediately knowledge if the script is working correctly and where are the errors eliminating unnecessarily long testing which can be fully automated. Also, if all the test passes, the script can be deployed into production environment. Overall the whole development process is accelerated and certain level of quality is ensured.

Use Case 2: NodeJS Server Development

This use case focuses on developing a web server using NodeJS, which is a JavaScript environment. While developing a server, the developer must usually re-run the NodeJS to see if the new code works correctly. With build automation and continuous integration, after a commit is made, continuous integration tool can spawn a new instance of the running environment and deploy the server there. If tests are written, it can be also checked if the whole server is working correctly. This way, it is possible to achieve some minimal level of quality and see results immediately. This automated process rapidly accelerates the whole development process and allows the developer to focus solely on the development itself.

Use Case 3: Apache Configuration

In this use case we focus on installation and configuration of Apache web server. Even though this is not exactly a development, the use case is very similar to the other ones.
After the Apache web server is installed, which is an automated process, administrator should ensure that Apache works correctly. Checking can be done either manually or using some pre-defined scripts like if the Apache is listening on port 80.
With build automation continuous integration the checks are done automatically and immediately after the installation, giving the result to the administrator, speeding up the whole process and freeing some of the administrator time that can be used for other important tasks.
Occasionally, a configuration is needed to satisfy company specific needs. Administrator needs to ensure all the aspects of the configuration are working correctly before applying changes to production. This can again ensure using build automation (mainly the testing part) and continuous integration saving administrator's time and reducing risks. If the new changes pass tests that continuous integration tool executed, changes can be applied into the production environment. If anytime administrator changes the configuration again, he/she will immediately get information about the result of the configuration which will save his/her time, ensure better quality of the service and better experience for customers.

Use Case 4: HTML, JavaScript and CSS Development

This use case focuses on developing a HTML web page using JavaScript and CSS. If the JavaScript code is wrong, the HTML part is not always expanded correctly. With CSS another developer can delete certain elements so there can be a script that checks all the key elements are present. While developing a web page, developers can use build automation and continuous integration to check for certain key aspects of the web page are present bringing a minimal level of quality engineering into his/her project.
After each commit is pushed, continuous integration tool can runs checks, that all needed files are included, that there is no syntax error and all key elements like the navigation bar are present. This reduces human errors that could potentially slow the whole project down.

Use Case 5: PHP Development

In this use case we will describe a developer that is working on a web page backend in PHP. The developer declares his/her application can run using PHP 5.5-7.0. For each new commit he/she should assure the script works for all declared PHP versions.
As in all previous use cases certain level of quality can be assured with the build automation and continuous integration. After each commit, tests will be run to check the basic functionality of the developed backend to assure that the key features are working and the new contribution to the project did not cause any new errors. The tests are executed on each PHP version that developer declared are supported. If any of them fails, developer knows that he/she break compatibility for certain version.
In extensive projects that are open source and developed by community, it can decided, based on the results of the tests, if the commit should be pushed into upstream or not saving a huge amount of time.

3.3 Summary

In all use-case examples it is evident that the build automation with continuous integration assures certain level of quality and in huge projects and also speed up the whole process which leaves more time for developers to focus on other important issues. Assuring good quality of the final product and discovering bugs before the product is released can have a massively positive impact on customers. Also, certain types of testing are ideal for build automation and continuous integration. Compatibility tests are using the maximum potential of this methodics and with build automation the whole process, once it is set up, is very fast.

Chapter 4

System Using Build Automation and Continuous Integration Methodics

In this chapter, a system that uses build automation and continuous integration will be designed and described. System aims to demonstrate integration of build automation and continuous integration tools into one system that supports development by automating processes and assuring certain level of quality of the final product.
First, in Section 4.1 system will be introduced and described in general. In Section 4.2 system will be described from component view, how the components of the system are connected and work together, then in Section 4.3 behavior of the system will be described and lastly in Section 4.4 the results will be discussed.

4.1 Description of the System

The system consists of several components that interact together. It consists of version control system, of continuous integration tool, of life-cycle manager and actors, in this use case developer.
Version control system provides access to the source codes for others—for example other team members or to community—backups and also versioning [18]. Repository also provides logical structure and sorting.
To automatize continuous integration, one of many available tools is needed. Without automation the continuous integration methodics would be ineffective and would cost much more time, there is no reason not to automatise a task that is repeated frequently.
Life cycle manager provides complete management over newly created virtual machines which will simulate production environment—used by tests. Based on continuous integration tool request, the life-cycle manager will initialise specified system and configure it with provided configuration. After the tests are completed, the manager will release resources and destroy the virtual machine.

4.2 Components of the System

In this section, system will be described from components points of view. Following description will be based on components diagram which can be found in Figure 4.1.

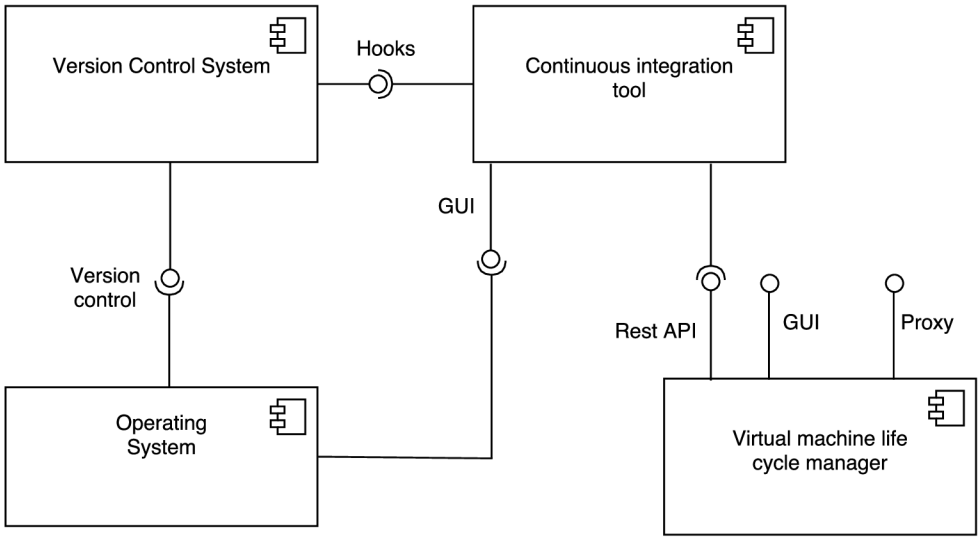


Figure 4.1: System's component diagram.

System consists of four components. First, there is a version control system that provides a version controlling to the user who is using operating system, the second component. The operating system is a mediator between user and the designed system. Operating system can also interact with third system - continuous integration tool using GUI interface, which almost every continuous integration tool provides. Continuous integration component also use interface of version control system, more specifically a tool named web hooks. Hooks are notifications that are sent as a HTTP POST message to the continuous integration tool after a push into the repository is executed. Continuous integration tool uses virtual machine life cycle manager's Rest API interface. This interface grants a way for continuous integration tool to communicate using HTML POST method with the virtual machine life cycle manager. Virtual machine life cycle manager also usually provides some interfaces like GUI and proxies.

4.3 Description of Behaviour

In this section, the work-flow of the system will be described. For illustration in Figure 4.2. the system's usual work-flow can be seen.

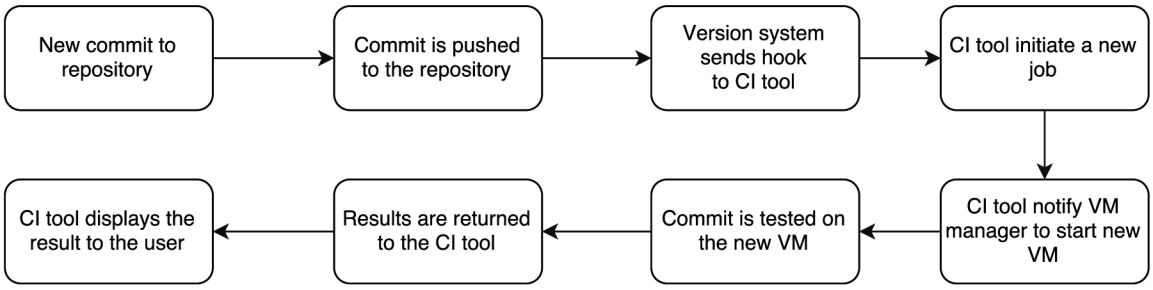


Figure 4.2: System's activity diagram.

In the beginning, user of the system commits a new source code into the version control system. After the commit is pushed, new source code is send into the repository. This

triggers a hook that is sent into the continuous integration tool. Continuous integration tool as a reaction to these changes in the repository starts a new job that aims to pull and test these changes. The tool notifies the virtual machine manager to initiate a new virtual machine and configure it. After the machine is ready, tests are run on this machine and results are sent back into the continuous integration tool, which display the results to the user, finishing this work-flow.

4.4 Description of the Implementation

For this thesis we have developed a system that was generally described in previous sections. The system is made of 3 independent virtual machines that runs under VirtualBox hypervisor. Each virtual machine has access to the internet and is connected to all other virtual machines via internal network `10.13.13.0/24`. More information about the virtual machines can be found in Table A.2.
First virtual machine provides GitLab CE that system uses as a version control system. GitLab is connected to Jenkins tool via web hooks giving a notification each time a push is executed. Jenkins firstly prepare new Docker image for Foreman and then Jenkins requests the Foreman to deploy a new Docker image. Docker images were used because of much bigger effectiveness. After the tests are executed, Jenkins processes the results and returns them to the GitLab which immediately gives feedback to the user showing an status indicator icon. If the job wasn't successful, user needs to go to Jenkins web UI and analyse the problem manually.
Using this, system we have been able to demonstrate the use cases analysed in Subsection 3.2.3. Results of the demonstration can be found in Section 4.5. Jenkins tools support many version control systems such as AccuRev, CVS, Subversion, Git, Mercurial, Perforce, Clearcase and RTC. The whole job does not depend on the version control system. For case studies, GIT was chosen as an version control system.

Table A.2: Information about virtual machines of the implemented system.

VM	Hostname	IP address	Running key application
1	jenkins.maklocal	10.13.13.110	Jenkins
2	foreman.maklocal	10.13.13.111	Foreman
3	gitlab.maklocal	10.13.13.112	GitLab CE

4.5 Evaluation of the System

The system implementation had issues but in the end it was possible to successfully integrate all the components and all the components work as designed. Foreman had a few bugs as the tool is still in development. The Docker plugin for Foreman is missing some features such as optional Docker command. On the other hand, these features are planned and will be added in the near future. Both Jenkins and GitLab were without any problems. After implementing basic tests, system allows user to successfully practice the continuous integration methodics. A user can solely focus on development and automatically the system warned him/her about errors his/her commit has made. Choosing Docker as environment provider for tests showed as an useful improvement. Installing whole new virtual machine takes approximately 7 minutes while running a docker image is almost instant.

Chapter 5

Conclusion

The main purpose of this thesis was to make a proof of concept of system using build automation for continuous integration. Based on this problem, a system using existing tools—which supports the main pillars of continuous integration—was designed. This system is able to use user-defined tests and configurations to automatically build and test project after a push is made into the version control system. The user of the system has results of his tests immediately after the tests are finished. Based on these results, the next action is decided and it can save time and resources of the developer/team.
The system was successfully implemented and the concept was proved. The concept was also demonstrated on several examples. These examples validated that the implemented system works correctly. In this thesis, only the use case of the developer working on a project using a GIT repository was tested. However, the system can work with any repository that the continuous integration tools supports. In my case it was Jenkins who can work with a huge variety of version control systems.
In the beginning the idea was to use virtual machine to simulate production environments for the tests. After a short research we have decided to use Docker containers, which dropped initializing the environment from several minutes (the time needed to install a completely new virtual machine plus apply configuration) to several seconds. This time reduction positively affects user, reducing the time needed for results and speeding the whole process.
The system brings many benefits to the user, namely (i) it speeds up the whole process thanks to automation, (ii) it helps to assure certain quality of the final product through the whole development cycle and (iii) it allows developers to focus solely on developing because everything else is automated. It can also help drastically in large open source projects to decide which commit should be pushed into upstream.
The whole system has several downsides as well. To deploy and correctly setup the whole system is not a basic task. Sometimes, writing proper tests can also be extremely time consuming and therefore delaying the development process.

5.1 Future development

The idea of the system is going to be further developed within the System Operations team in Red Hat which is looking for a continuous integration solution using open source tools to support the development. Also, the team looks for a solution using Docker containers which my system supports and therefore experiences gathered while working on this thesis

will help to implement the system into the environment of Red Hat.

Bibliography

[1] *The Apache Ant project* [online]. Revised: 03, July, 2015 [cit. 2. February 2016]. Available at: `http://ant.apache.org/`.

[2] *Automated Builds on Docker Hub* [online]. [cit. 2. February 2016]. Available at: `https://docs.docker.com/docker-hub/builds/`.

[3] *Foreman official webpage* [online]. Revised: 03, July, 2015 [cit. 2. February 2016]. Available at: `http://theforeman.org/introduction.html`.

[4] *GNU Make Manual* [online]. Revised: May, 2006 [cit. 2. February 2016]. Available at: `https://goo.gl/ua5q8N`.

[5] *Maven: The Complete Reference* [online]. 1. vyd. [cit. 2. February 2016]. Available at: `http://books.sonatype.com/mvnref-book/pdf/mvnref-pdf.pdf`.

[6] *Mozilla Wiki - Build:Release Automation* [online]. [cit. 2. February 2016]. Available at: `https://wiki.mozilla.org/Build:Release_Automation`.

[7] Berglund, T. a McCullough, M. *Building and Testing with Gradle*. 1. vyd. [b.m.]: O'Reilly Media, Inc., 2011. ISBN 9781449304638.

[8] Dooley, K. *Designing Large Scale Lans*. 1. vyd. [b.m.]: O'Reilly Media, 2001. ISBN 9780596001506, 0596001509.

[9] Duvall, P., Matyas, S. M. a Glober, A. *Continuous Integration: Improving Software Quality and Reducing Risk*. 1. vyd. Boston: Addison-Wesley Professional, 2007. ISBN 978-0321336385.

[10] Fowler, M. *Continuous Integration* [online]. Revised: 2006 [cit. 2. February 2016]. Available at: `http://www.martinfowler.com/articles/continuousIntegration.html`.

[11] Gregory, S. *Analysis of Firefox's Build Automation* [online]. Revised: 16, July, 2013 [cit. 2. February 2016]. Available at: `http://goo.gl/6evppJ`.

[12] Humble, J. a Farley, D. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. 1st. [b.m.]: Addison-Wesley Professional, 2010. ISBN 0321601912, 9780321601919.

[13] Kohsuke, K. *Meet Jenkins - Features* [online]. Revised: 07, January, 2016 [cit. 2. February 2016]. Available at: `https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins`.

[14] McConnell, S. *Code Complete, Second Edition.* Redmond, WA, USA: Microsoft Press, 2004. ISBN 0735619670, 9780735619678.

[15] Mecklenburg, R. *Managing Projects with GNU Make (Nutshell Handbooks).* [b.m.]: O'Reilly Media, Inc., 2004. ISBN 0596006101.

[16] Mráz, M. *Use of continuous integration in web application development.* Brno: Masaryk university, 2013. Master thesis.

[17] Palviainen, J. *Introducing Continuous Integration for C and C++ Software Development Projects on Linux Platform.* Lappeenranta: Lappeenranta University of Technology, 2009. Master thesis.

[18] Pilato, M. *Version Control With Subversion.* Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2004. ISBN 0596004486.

[19] Rouse, M. *Regression testing* [online]. Revised: Frebruray, 2007 [cit. 2. February 2016]. Available at: `http://goo.gl/xhbqi5`.

[20] Sierra, K. *Why „duh"... isn't.* [online]. Revised: 7 September, 2006 [cit. 2. February 2016]. Available at: `http://goo.gl/BDVpZ8`.

[21] Smart, J. F. *Jenkins: The Definitive Guide.* [b.m.]: O'Reilly Media, Inc., 2011. ISBN 1449305350, 9781449305352.

[22] Smith, C. *The 5 Big Benefits of Automated Deployment* [online]. Revised: 26 March, 2015 [cit. 2. February 2016]. Available at: `https://www.red-gate.com/blog/5-big-benefits-automated-deployment`.

[23] Spinellis, D. Software Builders. *IEEE Software.* 2008, vol. 25, s. 22–23. ISSN 0740-7459.

[24] StatCounter. *Top 5 Desktop Browsers from July 2008 to Jan 2016* [online]. [cit. 2. February 2016]. Available at: `http://gs.statcounter.com/#desktop-browser-ww-monthly-200807-201601`.

[25] Turnbull, J. *The Docker Book: Containerization is the new virtualization.* [b.m.]: James Turnbull, 2014. ISBN 9780988820203.

[26] Wikipedia. *Comparison of continuous integration software* [online]. Revised: 20, January, 2016 [cit. 2. February 2016]. Available at: `https://goo.gl/mHT5ZX`.

Appendices

List of Appendices

Appendix A

Content of DVD

As a part of this thesis there is a DVD which contains following items:

- Latex source codes in folder `latex`.
- Text version of the thesis in pdf format.
- The implemented system in folder `vms`.
- Url of video that demonstrates functionality of the system—a YouTube link. In the video all important parts of the system are demonstrated thus it saves a lot of time.
- Folder `src` with use-cases that contains source codes, tests and configuration bash scripts.
- File `credentials` that contains passwords and user-names for various applications.

Appendix B

Manual

This appendix contains description of steps which leads into a fully functional system implemented in this thesis. Follow each section below in their logical order to successfully get the system operational.

B.1 Installing VirtualBox

To install VirualBox, please follow instructions that can be found on official VirtualBox website https://www.virtualbox.org/manual/ch02.html. If you have RPM based system, you can also use manual that can be found on following url http://goo.gl/86pSP. Please note, that you need to have latest version of kernel to successfully install VirtualBox. If you have problem to run VirtualBox it is probably an issue of wrong kernel.

B.2 Configure VirtualBox

To configure VirtualBox for our system, please execute following command as root, which will ad internal network used by system's virtual machines: `VBoxManage dhcpserver add -netname intnet -ip 10.13.13.100 -netmask 255.255.255.0 -lowerip 10.13.13.101 -upperip 10.13.13.254 -enable`

B.3 Reassembling VirtualBox file

This section contains manual that describes how to reassemble parts of VirtualBox import file into one file. The split had to be done because of the file size—almost `12G`—which cannot be placed into one DVD.

- To reassemble VirtualBox file please gather all files from `vms` folder of each DVD (xaa, xab, xac) into one directory where you want to reassemble the VirtualBox file containing VMs.
- For Windows run in cmd: `copy /b xaa + xaab + xaac xkacma03-bp-vms.ova`
- For GNU/Linux run shell command: `cat xa* > xkacma03-bp-vms.ova`
- Please do not have any file starting with `xa` in your folder as `cat` would add their bites to the file corrupting it.

B.4 Importing virtual machines from VirtualBox import file

To import system's virtual machines into VirtualBox, please do:

- Start your VirtualBox and then click on `File > Import Appliance` and navigate to the `xkacma03-bp-vms.ova` file that you reassembled earlier.
- Click `Next`. Now you can adjust the number of resources like RAM and number of cores that each machine get if your computer doesn't have needed resources.
- Leave everything checked as VirtualBox will import discs as well as Network Interfaces etc. which are essential for the system and then click on `Import`.
- Wait until VirtualBox import system's virtual machines.

B.5 Summary

If you followed all steps, the system is fully functional. Please start all three virtual machines and then use Jenkins virtual machine to operate the system. You can access Jenkins tool from `jenkins.maklocal:8080`, Foreman from `foreman.maklocal` and GitLab from `gitlab.maklocal` through any supported browser like Firefox, Chrome etc.

B.6 User manual

To use the system, login into the virtual machine `Jenkins`, start terminal and go to the directory `/repos` (or you can clone a repository to any desired location). In the directory you can find various projects that are prepared and linked with continuous integration tool Jenkins. If you push any changes to the repository Jenkins will run new tests and the result can be observed inside the Jenkins web user interface. Please watch the video—link can be found on first DVD—to learn more about using the system.