jamk.fi

Implementing modern DevOps development environment for training

Case: N4S@JAMK

Juho Pekki

Bachelor's thesis
May 2017
Technology, Communication and Transport
Degree Programme in Software Engineering

jamk.fi

Description

Author(s)	Type of publication	Date
Pekki, Juho	Bachelor's thesis	May 2017
		Language of publication: English
	Number of pages 81	Permission for web publication: x

Title of publication

Implementing modern DevOps development environment for training
Case: N4S@JAMK

Degree programme

Software Engineering

Supervisor(s)

Mieskolainen Matti, Väänänen Olli

Assigned by

JAMK University of Applied Sciences, Need for Speed program, Rintamäki Marko

Abstract

The thesis was assigned by N4S@JAMK-project, which was a part of the DIMECC's Need for Speed-program (N4S). The Need for Speed research program focused on creating the foundation for Finnish software businesses by finding tools and business models for the changing needs of customers and digital economy. The assignment was to examine the latest technologies and develop a modern DevOps development environment for training including N4S@JAMK's earlier product called Contriboard.

The development environment was implemented using cloud services and container technologies. The development environment includes an automated containerized continuous delivery chain with functional and performance tests and the product monitoring.

The development environment included the following applications: Contriboard service as a product, GitLab for version control and store container images, GitLab-CI for building container images. Rancher for container orchestration. Jenkins automation server for deploying containers using rancher API, Locust for performance testing, Robot Framework for functional testing, Grafana for monitoring the product.

As a result, the development environment was fully functional, which was tested in AWS and DigitalOcean cloud services and in JAMK's internal network. Every application was running in a container except GitLab and GitLab-CI. Self-containerized applications are: Jenkins application stack, Locust, Robot Framework and InfluxDB. Future improvements are: Usage of dynamic IP addresses and servers, Changing Jarmo's data format from JSON to something else. The documentation was published as one of JAMK's contributions to Need for Speed program.

Keywords/tags (subjects)

DevOps, Docker, Rancher, containers, Continuous Deployment, test automation, N4S@JAMK, Contriboard, education

Miscellaneous

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Pekki, Juho	Opinnäytetyö, AMK	Toukokuu 2017
	Sivumäärä 81	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Modernin DevOps-kehitysympäristön luominen opetuskäyttöön

Tutkinto-ohjelma

Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)

Mieskolainen Matti, Väänänen Olli

Toimeksiantaja(t)

Jyväskylän ammattikorkeakoulu, Need for Speed-ohjelma, Rintamäki Marko

Tiivistelmä

Opinnäytetyön toimeksiantajana toimi N4S@JAMK-projekti, joka oli osa DIMECC:n Need for Speed -ohjelmaa (N4S). Need for Speed -tutkimusohjelman tarkoituksena oli löytää työkaluja ja työskentelymalleja, jotka auttaisivat yrityksiä sopeutumaan muuttuvaan ympäristöön ja reagoimaan asiakkaiden kehittyviin tarpeisiin. Opinnäytetyön tarkoituksena oli tutustua viimeisimpiin teknologioihin ja kehittää moderni DevOps-kehitysympäristö harjoituskäyttöön, joka sisältäisi N4S@JAMK:n aikaisemman tuotoksen Contriboardin.

Kehitysympäristö toteutettiin pilvipalveluiden ja konttiteknologian avulla. Kehitysympäristö sisältää automatisoidun jatkuvan julkaisun ketjun, johon kuuluu toiminnallisia testejä ja suorituskykytestejä sekä tuotteen monitorointi.

Kehitysympäristöön kuuluvat seuraavat palvelut: Contriboard, joka toimii tuotteena, GitLab versionhallintaan ja konttien levykuvien varastointiin, GitLab-CI konttien levykuvien luomiseen, Rancher konttien orkesterointiin, Jenkins-automaatiopalvelin konttien käynnistykseen Rancherin API:n avulla, Locust suorituskykytestaukseen, Robot Framework toiminnaaliseen testaukseen, Grafana tuotteen monitorointiin.

Lopputuloksena oli toimiva kehitysympäristö, joka testattiin AWS- ja DigitalOcean-pilvipalveluissa sekä JAMK:n sisäisessä verkossa. Kaikki palvelut pyörivät konteissa lukuun ottamatta GitLabia ja GitLab-CI:tä. Itse kontitettuja palveluita ovat Jenkins-sovelluspino, Locust, Robot Framework ja InfluxDB. Tulevaisuuden parannuksia ovat dynaamisten IP-osoitteiden ja palvelimien käyttäminen, Jarmon dataformaatin muuttaminen JSON:sta joksikin toiseksi. Dokumentaatio julkaistiin yhtenä JAMK:n tuotoksena Need for Speed -ohjelmalle.

Avainsanat (asiasanat)

DevOps, Docker, Rancher, kontit, jatkuva julkaisu, testiautomaatio, N4S@JAMK, Contriboard, koulutus

Muut tiedot

Contents

Figures

TERMINOLOGY

API	Application programming interface
AWS	Amazon Web Services
CD	Continuous delivery
CI	Continuous integration
SDS	System Design Specification document
DNS	Domain Name System
DVCS	Distributed version control system
EC2	Amazon Elastic Compute Cloud
IaaS	Infrastructure as a Service
IP	Internet Protocol
JAMK	JAMK University of Applied Sciences
LXC	Linux Containers
N4S	Need for Speed -program
OS	An operating system
PaaS	Platform as a Service
SaaS	Software as a Service
SDLC	Software Development Life Cycle
SRS	System Requirement Specification document
SSH	Secure Shell
SUT	System Under Test
UI	User Interface
VCS	Version Control System
VNC	Virtual Network Computing

VPN	Virtual Private Network
VM	Virtual Machine
WP	Work package

1 INTRODUCTION

1.1 Background

The IT-sector has been evolving rapidly over 20 years. These years have also changed the way how companies build their software. One of these ways is called DevOps, which is the combination of cultural philosophies, tools and practices for individuals and organizations that affects a way of the people work and why. DevOps aims for developing high quality products in a fast phase, using sustainable and effective work processes. (What is DevOps n.d.)

Cloud services has changed the way companies run their products. Earlier, companies had their own servers and datacenters, which required big investments and employees to manage them. Cloud service providers offer servers, applications, databases and other service over the internet. Cloud services has multiple benefits; they are flexible and users can modify them to match their needs. Cloud services can be invoked fast, which ensures fast and easy scalability. Because of that, companies do not have to reserve capacity in advance. Cloud services prices are based on the usage and extra services. (Why move to the cloud 2015.)

Latest hype started few years ago. It is called operating-system-level virtualization or better known as container technology, which is not a new thing; however, the breakthrough happened because of the container technology called Docker. Containers encapsulate an application and all its dependencies in one package. Containers can be deployed on any environment regardless of the environment's operating hardware. Usually containerized services are modular and consist of multiple containers, where one container includes only one process. It is recommended to develop the system with independent components, which can be started, scaled and replaced quickly. (What is a Container n.d.)

1.2 Thesis objectives

The thesis was assigned by N4S@JAMK-project, which was a part of the DIMECC's Need for Speed-program (N4S). Need for Speed research program focused on creating the foundation for Finnish software businesses by finding tools and business models for the changing needs of customers and digital economy.

The assignment of the thesis was to examine the latest technologies such as containers and cloud services and develop modern DevOps development environment for training that includes N4S@JAMK's earlier product called Contriboard. The development environment services and tools must be free, so students can use them. DevOps is a combination of cultural philosophies, principles and practices such as continuous deployment. Continuous deployment means that the software development process is automated and done continuously from source control to the deployment. Every change in the source code is automatically built tested and deployed to the production. DevOps allows teams to deliver high quality products faster, more reliably and more cost-efficiently than traditional ways. It also enables faster updates and fixes, which are directly connected to the customer satisfaction.

1.3 N4S -Program

Need for Speed (N4S) is DIMECC's research program, which started in 2014 and ended in the end of May 2017. The purpose of the Need for Speed-program was to create the foundation for Finnish software businesses by finding tools and business models for changing needs of customers and digital economy. The program involves several large Finnish software companies, research institutes and universities. Need for Speed-program contained three focus areas, which are called work packages (Need4Speed brochure n.d.):

WP 1: Delivering Value in Real time

WP 2: Deep Customer Insight

WP 3: Mercury Business

Delivering Value in Real Time package provides approaches, methods and tools for fast and cost-efficient design, creation and prototype of new services and products. The main purpose of this is to find solutions to increase the deployment speed and delivery speed, which leads to a better cost-efficiency. (Paradigm Change – Delivering Value in Real Time n.d.)

Deep Customer Insight package focuses on gaining information about the true customer value of products by collecting usage and behavioral data and feedback from customers and environments. The package also provides data analysis and visualization, which helps with decision making. (Deep Customer Insight n.d.)

Mercury Business focuses on finding solutions on how companies can behave like liquid mercury, meaning adapting to the evolving business conditions and opportunities with minimum effort. (Mercury Business n.d.)

1.4 N4S@JAMK

JAMK University of Applied Sciences was a part of N4S-program from the beginning, mainly focusing on the Delivering Value in Real Time work package. N4S@JAMK team has developed service called a Contriboard and a reference production environment called Cooper, which is based on open-source services. JAMK has also produced multiple demos and theses for the N4S-program.

1.4.1 Contriboard

Contriboard is an open source collaboration tool working in real time. Users can create boards, create tickets to them and share boards with other users. Contriboard's technology choices have kept an eye on scalability and multiple different devices. Contriboard has been developed using the following technologies: MongoDB, Node.js, Express.js 4, Socket.IO 1.0, React and Bootstrap 3. (About technologies 2014.) Figure 1 presents Contriboard's board view.

Figure 1. Contriboard's board view (Contriboard n.d)

Contriboard's early version was known as Teamboard, which was a part of FreeNest v1.4, which is an open-source product platform. It was built based on the need to have a ticket board for teams, which could be used in different circumstances such as Kanban board. Teamboard allowed users to create and move ticket across the board. (History of the product 2014.) When Need for Speed project was launched, N4S@JAMK started to develop Contriboard as a product which is separated from FreeNest product platform.

1.4.2 Cooper – the reference production environment

N4S@JAMK has developed multiple production environments and Cooper is the latest of them. The purpose of the reference production environments is to increase the speed of software development process by using the DevOps methods principles. Cooper production environment consist of the commonly used free-to-use services, which are integrated together (see Figure 2). The Key aspect of the production environment is that it is easy to set up and use. (Cooper production environment 2017). Cooper production environment does not involve any software testing, so for some

point of view, the thesis environment can be thought as a missing puzzle. Documentation and installation instructions can be found in JAMK-IT GitHub: https://github.com/JAMK-IT/DOC10-example-project/wiki/Cooper-product-line

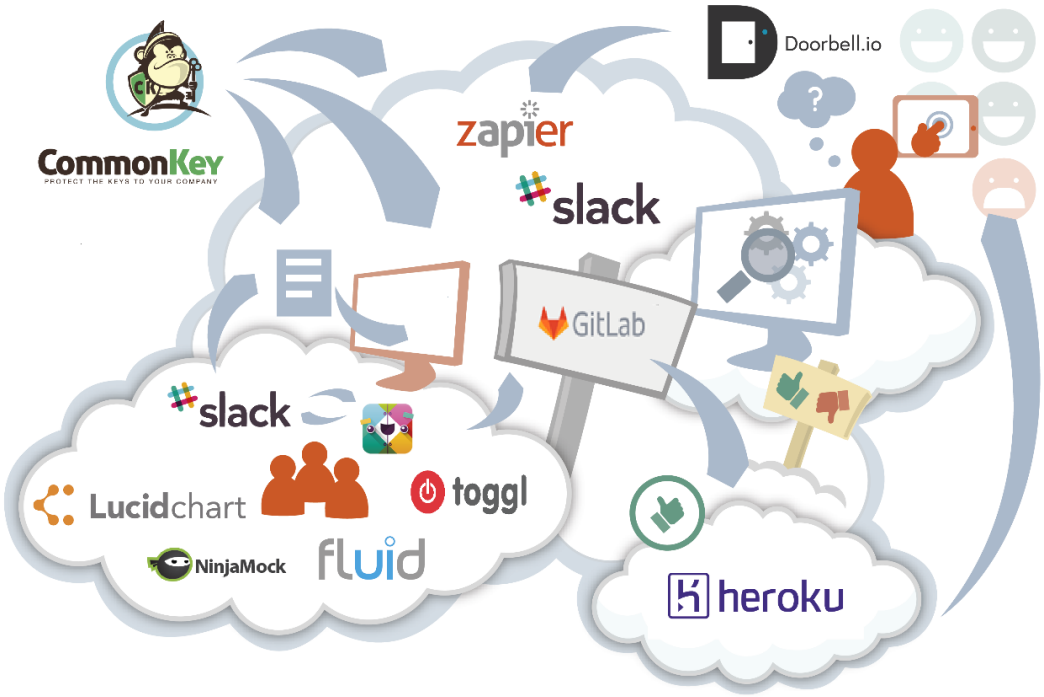


Figure 2. Cooper toolchain (Cooper production environment 2017)

2 MODERN PRODUCTION OF SERVICES

2.1 Software Development Life Cycle

Software Development Life Cycle (SDLC) is a process for developing the software products. SDLC consist of multiple phases and describes the execution order. SDLC offers a plan with a development phases, which will help teams to improve development process and the quality of the product. Each phase in the SDLC has its own activities, which produce deliverables to the next phase. SDLC phases depend on the selected Software Development Model. (Ghahrai 2015.)

The software development models are approaches for software development process. Purpose of the models is to specify a development process stages and their execution order, which should improve software quality and ensure success in the development process. A model is chosen based on the project goals. There are multiple models available to fulfill different objectives. The selected model affects to the testing process, because it determines what, where and when is tested. (Ghahrai 2015; What are Software Development Models n.d.) To get a better understanding about SDLC phases, let's have a look at the phases of the traditional Software Development Model, which is called the Waterfall Model (see Figure 3).

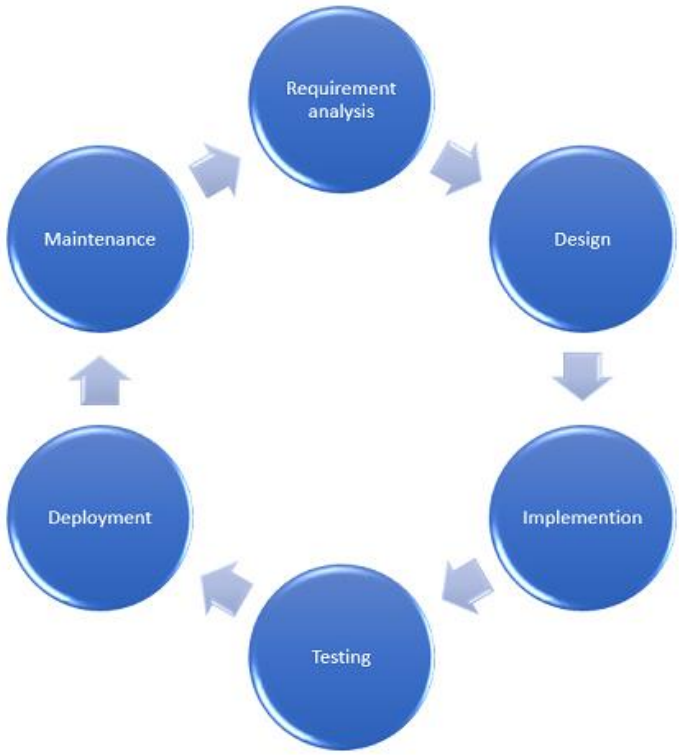


Figure 3. Traditional SDLC phases

The first phase is Requirement analysis, where managers and stakeholders determine the requirements of the product. This purpose of this phase is to gather and clarify the customer requirements, quality standards, terms and make sure that both parties are aware of those. After this, the requirements are validated and a Software Requirement Specification (SRS) document is created. (Ghahrai 2015; What are the SDLC phases n.d.)

In Design phase, the software developers and designers start designing the high-level architectural suggestions of the product based on SRS. These suggestions are documented in a System Design Specification document(SDS). SDS defines all the used technologies, needed components and behavior of them and internal and external connections. SDS and other parameters such as risks, time and budget are reviewed with the stakeholders, which will select the most suitable solution for them. (Ghahrai 2015; What are the SDLC phases n.d.)

Design phase is followed by Implementation phase. Implementation phase can be also called the coding or development phase. In this phase, the developers start developing the product based on SDS and SRS document. Implementation phase includes developing and simple unit test and reviews, until the product is ready. (Ghahrai 2015.)

The next phase is called the Testing phase. This phase depends on the selected Development model, e.g. modern agile methodologies do not contain this phase, because testing is a part of other phases. The purpose of this phases is to find possible defects and ensure that the product matches the SRS document. Testing is done by using functional and non-functional methods. If any defects or error are found, developers fix them and the hole system needs to be tested again, until the system is defect free. (Ghahrai 2015.)

After the Testing phase is completed, the product is deployed to the testing environment. Usually the first thing that the customers do is beta testing to ensure that the product matches the SRS document and does not have any bugs. If the customer finds errors or the product does not match SRS, the product returns to the Implementation phase. After the customer approves the product, it will be deployed to the customer’s environment. (What are the SDLC phases n.d.)

The final phase of the traditional SDLC is called Maintenance phase. In this phase, the maintenance team is looking for the product if the post-production issues occur. If the maintenance team discovers an issue, it will inform the development team which will perform the necessary fixes. (What are the SDLC phases n.d.)

2.2 Evolution of the Software Development Models

The IT-sector has been evolving rapidly over 20 years. Multiple technologies have emerged and died, and the same pace continues. These years have also changed the way how to build software. These models are evolving continuously and have improved the quality, speed and success rates of the software projects. (Clarke 2016.)

The waterfall model is the oldest and the best known of the development process models. Historically, the waterfall model has been very popular in the 1980s and 1990s, mainly because it was easy to understand and use, and it was the only accepted model available. In a waterfall model, every phase must be fully completed and reviewed before moving to the next phase (see Figure 4). The waterfall model expects that the requirements do not change after the analysis phase or the team must start the process all over again from the requirements phase. The waterfall model has multiple risks; a working product comes out late and the testing phase is after the development phase, which means that problems come out late, are hard to repair and cost much. The waterfall model fits projects, which are small and where requirements are clear and do not change; However, it does not meet the requirements of a modern software development, because it assumes that one user can only have one role, requirements do not change and there is no space for new ideas. (Clarke 2016; What is Waterfall Model n.d.)

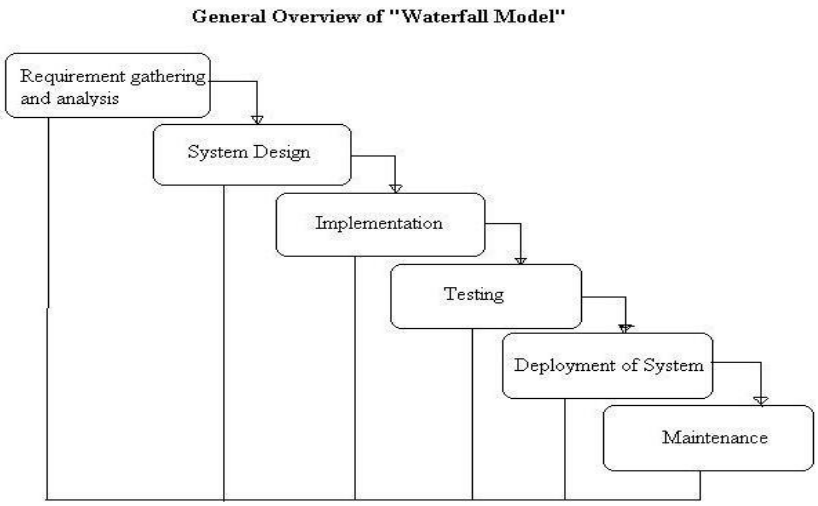


Figure 4. The waterfall model (What is Waterfall Model n.d.)

The waterfall model's process was very restricted and code fixes was very expensive, which lead to invention of multiple alternative models. One of these was a model called the "V-model", which is also called as Verification and Validation model. Nowadays it is one of the most used development process model. (What is V-model n.d.) What does Verification and Validation mean?

Verification is the process to ensure that the product matches the specifications between the different phases of the development life cycle. Verification is a static analysis technique, which is performed by reviewing code and documentation without executing the code. (Reddy & Prasad 2016.)

The validation process ensures that the product matches the customer requirements at the end of the development life cycle. Validation is ensured with dynamic testing where the tests are executed on real product to get reliable results. The validation process is performed using functional and non-functional testing techniques. Common methods are unit testing, integration testing, system testing and acceptance testing, which are explained more detailed in chapter 3. (Reddy & Prasad 2016).

V-model is a modified version of the waterfall model, where the product testing is planned parallel with a corresponding development life cycle phase. Each phase of development life cycle is verified before the next phase. Unlike in the waterfall model, V-model does not have phase called testing. Figure 5 illustrates the phases of V-model, where on the left side are development phases and on the right side are testing phases. V-model logic is simple and it fits projects, which are well organized and where requirements are clear. V-model has multiple disadvantages: like the waterfall model, requirements cannot change or all requirement and test documents must be updated and working product comes out relatively late. (V-model advantages and disadvantages n.d.)

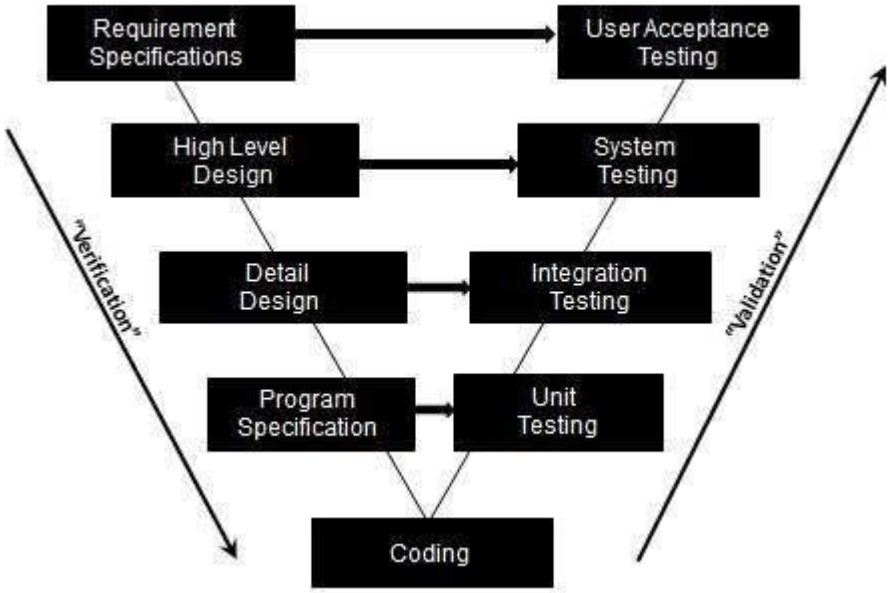


Figure 5. V-model phases (V-model n.d)

Because the success rate of the software development was minimal and the IT-sector was evolving rapidly in the late 1990s, the companies needed new development models and multiple mixed models were introduced. These models focused on improving the success rate by allowing fast changes, building the code daily, early releases of the code and using small developer teams. (A short history of Agile n.d; Clarke 2016.)

One of the most used term in the software development is Agile. A term was announced in 2001, after a group of the software development experts gathered to talk about problems of the software development. After a meeting, group announced Manifesto of Agile Software Development, which is a collection of values and principles to improve the software development. (A short history of Agile n.d) Nowadays, these values and principals are commonly used.

Agile manifesto values are:

> *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> *Individuals and interactions over processes and tools.*
> *Working software over comprehensive documentation.*
> *Customer collaboration over contract negotiation.*
> *Responding to change over following a plan.*
> *(Manifesto for Agile Software Development 2001.)*

Agile is an approach that describes a set of values, principles based on Agile manifesto and practices for software development. Agile encourages teams to adaptive planning, time-based iteration and continuous evolving. (What is Agile? 2013.) Figure 6 demonstrates what agile is.

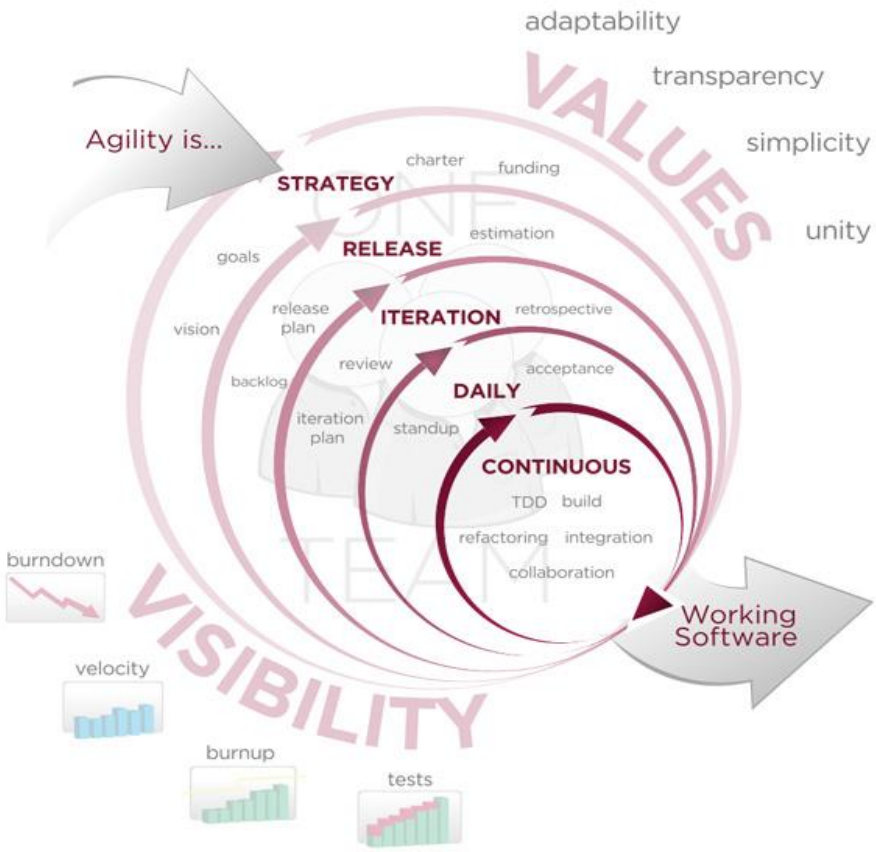


Figure 6. Agile software development (What is Agile? 2013)

Agile software development is also an umbrella term for a multiple incremental and iterative methodologies. The most commonly known agile methodologies are SCRUM, Feature Driven Development, and Extreme Programming. Every methodology has a unique approach but they share the basic Agile Manifesto values and principals and multiple common practices such as backlogs, daily meetings, continuous integration, reviews and user roles. These methodologies rely heavily on communication between the customer and team, and in cross-functional teams build the product in small pieces, using time boxed iterations, which are called sprints. Sprints are planned by using a backlog, where task are prioritized by the customer. By using sprints, the team can respond fast to the evolving requirements. Every sprint should produce a working product. (What is Agile Software Development n.d.) To understand agile methodologies better, Figure 7 presents the differences between Agile methodologies and traditional methodologies such as the waterfall model.

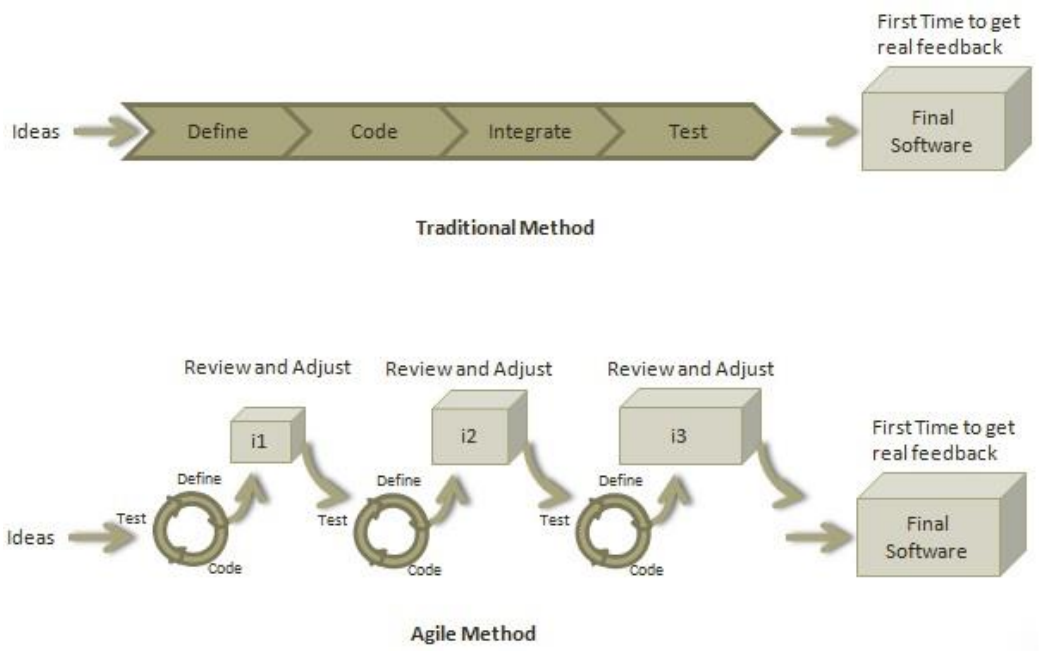


Figure 7. Agile method versus traditional method (Agile n.d)

Agile methodologies got multiple advantages compared to traditional models; they allow the customer be part of the process by prioritizing requirements and add more requirements during it, working software is delivered frequently and the project costs and schedule are predictable. (Benefits of Agile Software Development 2015.)

Another notable term which is connected to Agile is “Lean”. Lean is philosophy not a methodology. It originates from Lean Manufacturing, which is modified version from Just-In-Time (JIT) Manufacturing. JIT focuses on efficiency and was philosophy behind famous Toyota Production System. Lean systems focus on sections that provide value and everything else is waste that must be eliminated. This waste can be extra features, any kind of delays, partially done work etc. Lean is a set of principles for maximizing customer value and minimizing waste. (Daniels & Davis 2016, 36; Marschall 2012.)

2.3 DevOps

The term DevOps comes from words “Development” and “Operations”, which refer to merging these engineering groups together and forming small teams that use shared tools and practices during entire SDLC. There is no official or clear definition for DevOps. DevOps is the combination of cultural philosophies, tools and practices for individuals and organizations that affects the way people work and why. DevOps aims for developing high quality products in a fast phase, using sustainable and effective work processes. Shortly, DevOps is a way of thinking and a way of working. (Daniels & Davis 2016, 13; What is DevOps n.d.)

Because there is no clear definition for DevOps, there are multiple ways to perform it. Every organization have their own ways of performing it, however, there are four common sections for effective DevOps: collaboration between teams, sharing and learning with other teams and departments, finding the right tools and scaling inside the organization. (Daniels & Davis 2016, 58.)

What is the difference between agile methodologies and DevOps? DevOps shares many same principles and practices and also focuses on interactions and collaboration, but DevOps also focuses on cultural aspects. It is important to understand that DevOps is not a software development model. DevOps shares and extends the same principles and practices, these affect the entire organization, unlike Agile methodologies that only affect the software development process. (Daniels & Davis 2016, 33.) Figure 8 presents the differences between DevOps, Agile and Lean. Effective DevOps environment consists of multiple terms and concepts, which are presented below.

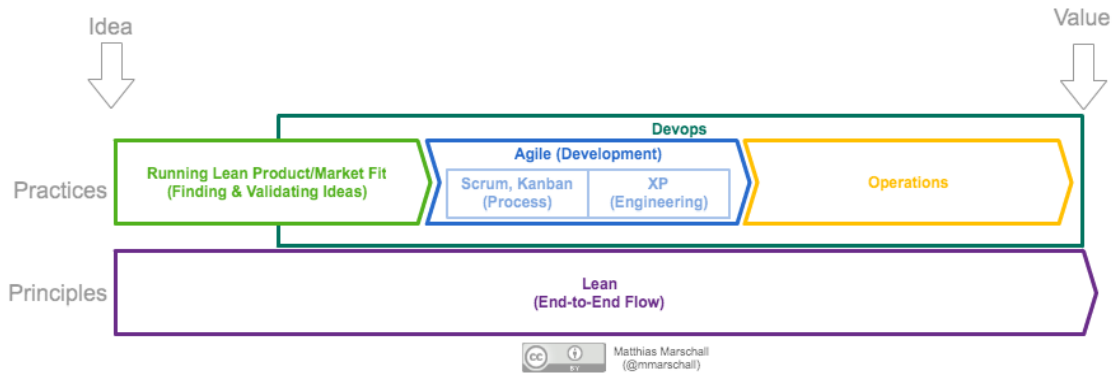


Figure 8. Differences between DevOps, Agile and Lean (Marschall 2012)

Why should organizations and teams use DevOps? DevOps allows teams to deliver high quality products faster, more reliably and more cost-efficiently than traditional ways. It also enables faster updates and fixes, which are directly connected to the customer satisfaction. Software reigns in every business as the key differentiator and factor in business outcomes. No organization, developer or IT worker can or should avoid the DevOps movement. (Guckenheimer n.d.)

2.3.1 Version Control

The purpose of version control is to store and record changes of the files. These files are usually source code, however, they also can be assets and documents. With version control, user can see what, why and when changes to the file were made. This is important when multiple people are working with the same files, because if mistakes were made they are easy to locate and fix. Version control is managed with the Version Control System (VCS) tools. The most common of these is Git. (What is version control n.d.)

Git is an open source version control system. Git is a distributed version control system (DVCS), which means that the developers have a full copy of the entire repository with a revision history on their local computer unlike other version control systems like CVS and SVN. This allows that the developer can do changes and commits without connection to the repository and perform push to the repository next time when the network is available. Git is also very fast because all tasks are executed locally and then pushed to the repository. Git uses snapshots to save the project state

after each commit, unlike other version control systems that use a list of file-based changes. Snapshots are also efficient and unchanged files are linked with earlier snapshots instead of saving them again. (Getting Started – Git Basics n.d.)

Other major advantage in Git is the branching model. Branches enable feature based workflows. Developers can create multiple branches for new features and testing without changing the code in the master branch which contains the final product. Basically, the developer clones a master branch to the new branch and starts developing. After a new feature is developed and tested, the branch can be merged with the master branch. (Branching and Merging n.d.)

Git is mostly used with third party repository hosting services such as GitHub, GitLab and Bitbucket. Usually the hosting services are free cloud-based services; however, many of them are also available for local servers, yet, some may require subscription. The development environment of the thesis uses free GitLab service which is described more detailed later.

2.3.2 Continuous Integration

Continuous Integration (CI) is a development practice where developers constantly commit code changes and new unit tests into a repository using a version control system, which launches automated build and test pipeline after each commit. The purpose of continuous integration is to detect possible issues as soon as possible which leads to better software quality and reduces the time used to validate new features. (Continuous integration n.d.)

All version control services offer the possibility to create multiple branches into the repository. Developers create new branches based on the main development branch, and when creating new features, they ensure that the main development branch stays clean and no one else is making changes to the same files at the same time. When a developer commits changes to the repository, CI service detects the changes and runs pre-defined builds and unit tests to detect possible issues. There are multiple CI-services to choose from, but usually companies select a previously used service. The mostly used CI-services are Jenkins, TravisCI, CircleCI and GitLab-CI.

2.3.3 Continuous Delivery

Continuous delivery (CD) is the next step from continuous integration. While CI focused on building and unit tests, CD expands this pipeline by deploying code to staging an environment where multiple tests are executed. These tests can focus on integration testing or acceptance testing such as load testing and reliability testing. The purpose of CD is to automatically build, test, configure and prepare the code to the production environment. Usually the deployment to production environment is also automated; however, it needs to be approved manually. (Fowler 2013.)

2.3.4 Continuous Deployment

Continuous deployment is basically the same as continuous delivery except the code is deployed to the production environment automatically. This leads to multiple production deployments per day, because every commit that pass the testing stage will be deployed. Figure 9 presents the differences between continuous integration, continuous delivery and continuous deployment.

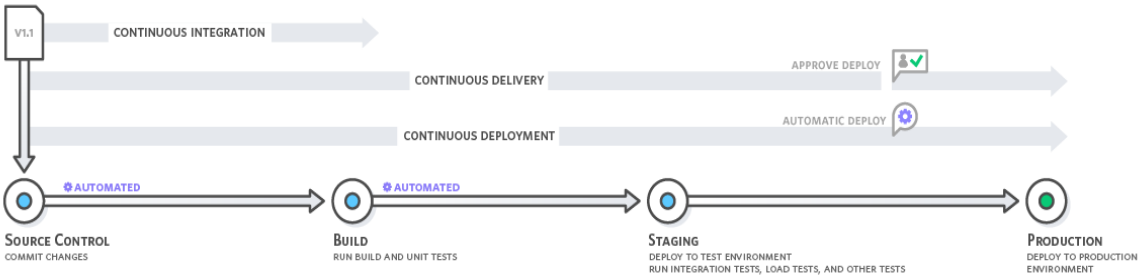


Figure 9. Differences between continuous techniques (Continuous delivery n.d)

Why should companies use continuous deployment? Continuous deployment has multiple benefits against a basic model where releases include multiple features. Continuous deployment allows much faster release cycles, because completed features are deployed to the production environment instantly, and the developer team does not have to gather them to the next planned release, the release date of which can be even six months away. Faster releases also provide faster feedback from the customers, which helps the team to build the right product and not waste valuable

time to the not important features. Continuous deployment also improves developer the productivity and efficiency of the developer teams. The teams do not have to set up or maintain test environments or worry about the deployment to the production environment because has been everything automated. Of course, the creation of continuous deployment pipeline requires plenty of work and test cases also; however, in the long run this leads to reduced costs and better product quality. (Lianping 2015.)

2.3.5 Cloud Computing

Cloud computing also known as the cloud, has been one of the biggest changes in IT sector past 10 years. It has changed the way how companies build their services. Cloud computing was originally concept of hosting computer resources over the internet. Nowadays, cloud computing has evolved and it consists almost anything related to IT such as servers, applications, service orchestration and much more, which why it is more reasonable to call them cloud services. (Bond 2015, 5-10.) The U.S. National Institute of Standards and Technology defines cloud computing as follows:

> *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction* (NIST Special Publication 800-145. 2012).

Cloud computing has multiple benefits compared to dedicated servers. Cloud services are flexible. Users can select a service which fits best to their needs from three service models (IaaS, PaaS, SaaS). Because the applications are hosted in a service provider's data centers which can be all over world, this allows the users to select the best centers for their business geographic area. Cloud services can be invoked fast, which leads to a possibility to scale services up and down fast when required. Cloud service prices are based on the usage and possible extra services. Earlier points lead to cost savings, because users not have to reserve full capacity at once, and starting a

new service does not require heavy investments on hardware. Cloud services do not have special hardware requirements from end user's point of view, the application will work on a web client. (Why move to the cloud 2015.) Figure 10 presents the benefits of the cloud from RightScale 2017 - State of the Cloud report. The results are based on interviews with over thousand professionals (RightScale 2017 - State of the Cloud Report 2017.)

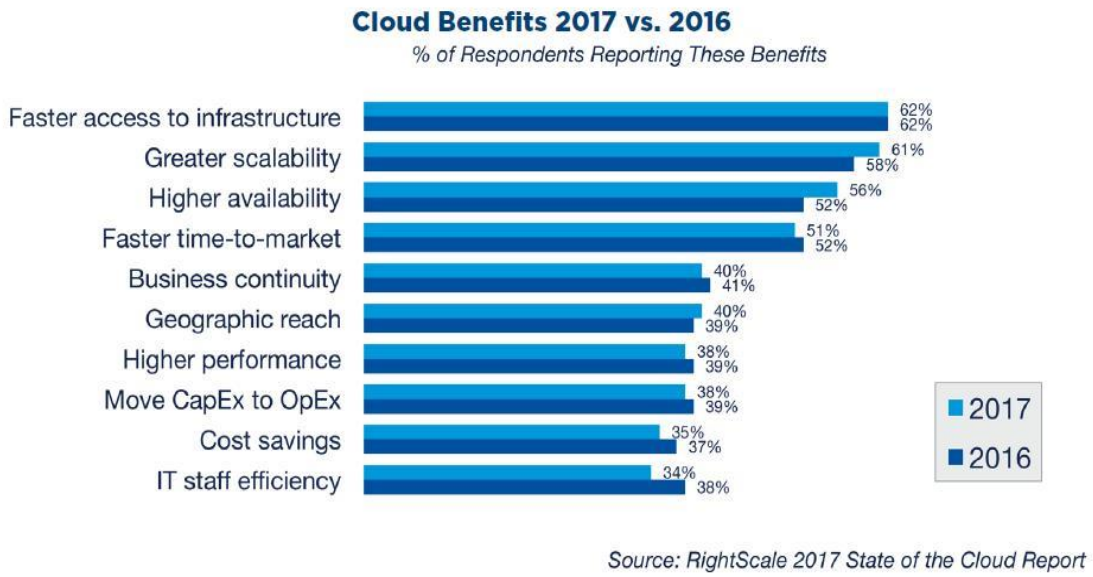


Figure 10. Cloud benefits (RightScale 2017 - State of the Cloud Report 2017)

Just like in many cases, everything is not perfect in cloud services. Cloud services also have risks. Because cloud services are hosted in data centers and accessible via web client, they rely heavily on credential and access management and multiple interfaces between services. In the worst case, an exposed API or leaked account can lead to a data breaches. Because cloud services rely on multiple interfaces between services and resources, there are multiple possible attack surfaces. Another big threat is so called distributed denial-of-service (DDoS) attacks, where multiple sources create large traffic to the service, which leads to the service unavailability. (The Treacherous 12 2016.)

Cloud service architectures are divided into three layers based on the service models. These service models are: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). (Cloud Service Models n.d.) Figure 11 presents the differences between the service models.

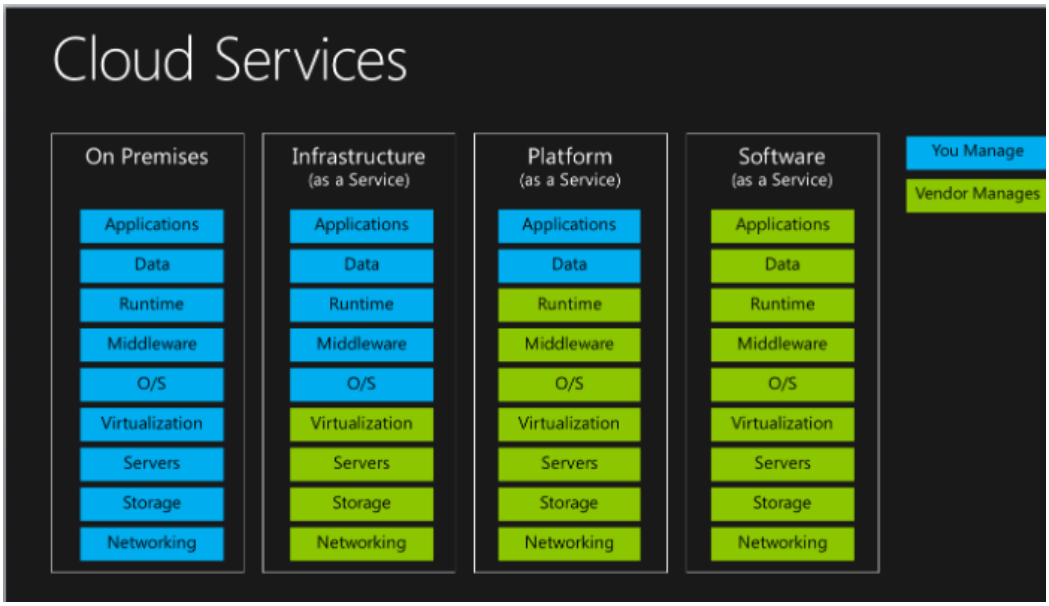


Figure 11. A cloud service models (Azure IaaS n.d)

Software as a Service

In a Software as a Service model the application is delivered to a service provider, which hosts and makes it accessible over the internet. The applications are usually accessible via web browser, or in some cases they use a program interface. By using SaaS applications, the companies do not need to invest for the infrastructure and hardware. The users do not have to worry about hardware or install or update applications. Good examples of the SaaS applications are Google Docs and Microsoft Office 365. (Rafaels 2015, 16.)

Platform as a Service

In a Platform as a Service model a cloud service provider delivers a complete development and deployment environment with hardware and tools for the software development. The service provider manages the infrastructure, e.g servers, network,

operating system and storage. The customer only manages deployed applications and possible environment configuration setting. The applications are developed using technologies and tools that a service provider delivers. Like in a SaaS model, the final applications hosted by a service provider are available over the internet and end-users do not need to install anything on their local computers. Examples of these kind of services are Heroku and Google App Engine. (Rafaels 2015, 17.)

Infrastructure as a Service

In Infrastructure as a Service, a cloud service provider provides virtualized computing infrastructure as an on-demand service over the internet. The infrastructure includes servers and other resources such as storage, network, load balancers etc. A service provider manages the infrastructure and services and the users are responsible for managing the operating system, middleware, runtime, data and application. The major benefit of the IaaS model is easy scalability and flexibility. For example: there is a spike in data traffic in the evening, so the application can be scaled to match the needs in the day and be scaled down in the night, which leads to major cost savings. IaaS model is the most common type of a cloud service model, and billing is based on consumption and extra services. Examples of these kind of service providers are Amazon Web Services and Microsoft Azure. (Rafaels 2015, 17.)

Cloud services can be categorized also by the deployment models: Public cloud, Private cloud, Community cloud, Hybrid cloud. In Public cloud, the cloud infrastructure is open for public usage. Private Cloud is also known as internal cloud, where the cloud infrastructure is restricted for single organization. In Community cloud, the cloud infrastructure is shared between multiple organizations, who share the same mission and security concerns. Hybrid cloud is a combination of previous deployment models, where two or more models are bound together retaining unique entities. (Bond 2015, 14.)

There are multiple companies offering a cloud services. Figure 12 presents adoption between cloud service providers. RightScale's report results are based on interviews with over thousand professionals. According to the report, Amazon Web Services is by far the most popular provider. (RightScale 2017 - State of the Cloud Report. 2017.)

The development environment of the thesis has been tested by public service provided by AWS and DigitalOcean, which are presented more detailed below.

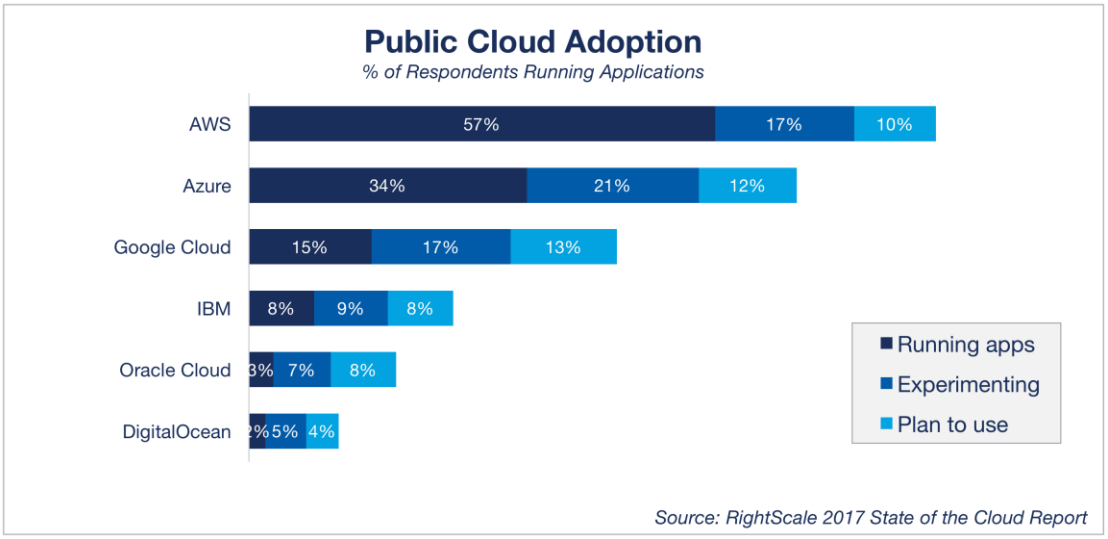


Figure 12. Public Cloud Adoption 2017 (RightScale 2017 - State of the Cloud Report 2017)

Amazon Web Services

Amazon Web Services (AWS) is a scalable and reliable cloud infrastructure platform based on the IaaS model by Amazon. AWS was released in 2006; thus, it is a pioneer of cloud services and today it has data centers in all continents except in Antarctica. AWS offers multiple cloud-based services such as storage, compute, networking, databases and analytics which help organizations to scale, move faster and lower their IT costs. (About AWS n.d.)

Elastic Compute Cloud (EC2) is a web service offering resizable compute capacity. EC2 allows users to control instances using web service APIs, which also offers possibility to automatically scale user application depending on its needs. EC2 instances can be integrated with other AWS services such as Elastic Load Balancing and Route 53. (EC2 n.d.) Users can select their operating system for instance between multiple Linux distros, Windows Server and Amazon Machine Image. The instance pricing model of EC2 is based on usage and instance type.

DigitalOcean

DigitalOcean is IaaS based cloud infrastructure provider. DigitalOcean offers a cloud infrastructure with multiple features such as: load balancers, floating IPs and API. (Company Overview of Digital Ocean 2017) Nowadays, it has grown into the second largest hosting company in terms of the web-facing computers, after AWS (Digital-Ocean becomes the second largest hosting company in the world 2015). DigitalOcean has datacenters in 8 countries. DigitalOcean calls its virtual private servers (VPS) Droplets. Users can select Droplets operating system from 6 different distributions: Ubuntu, FreeBSD, Fedora, Debian, CoreOS and CentOS. DigitalOcean also offers Droplets with pre-installed apps such as Docker, Django and NodeJS; however, in the writer's experiments, the latest pre-installed Docker did not work like a clean installation. DigitalOcean Droplet's pricing starts at 5 dollars and can go up to 1,680 dollars per month. The smallest Droplet contains one processor core, 512 megabytes of RAM, 20 gigabytes of SSD disk and 1 terabyte of data transfer. DigitalOcean offers an API interface for creating and controlling droplets. (Compute n.d.)

Users can create Droplets using a simple web UI. When creating a new Droplet, user selects Droplet OS, size, datacenter region and gives Droplet a name. Users can also select extra features like private networking between Droplets, weekly backups and IPv6- protocol. Web UI is simple and users can see simple graphs from Droplet's bandwidth, CPU and disk states. Users can take Virtual Network Computing (VNC) connection to Droplet, take snapshots and backups and change Droplet's size using web UI. Figure 13 presents a Droplet web UI.

,

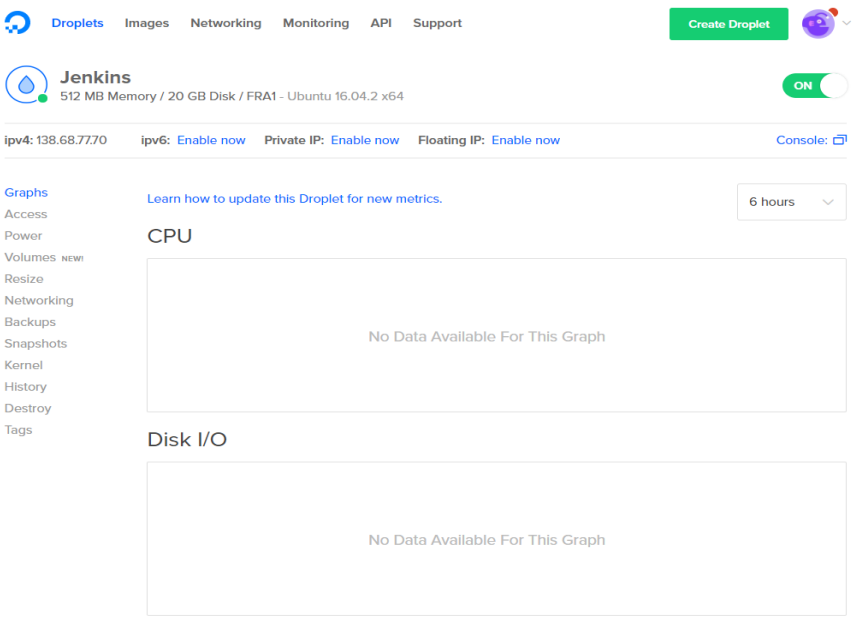


Figure 13 DigitalOcean web UI

2.3.6 Microservices

There is no formal definition for microservices. The microservice architecture is a method of developing applications as a package of small services. Each service usually encapsulates one business functionality, which can be scaled, tested, deployed and managed independently. Services usually communicate through Application Programming Interfaces (APIs). (Understand microservices 2016.)

Microservices can be developed separately using different technologies, which allows to select the right tools for each service. This way the development teams can focus more on the customer scenarios and use technologies that they choose. Microservices also allows faster products release cycle, because the developer can make changes to a single service without the need to build and deploy an entire application like in monolithic approach. Scaling a desired component is easier than in large monolithic applications, because the developer can scale just a single service, not the entire software. (Newman 2015, 5.) Microservices also have downsides: Developers must manage multiple separate entities, more complex deployments and network traffic between services (Newman 2015, 11). Figure 14 presents basic differences between monolithic application and microservices.

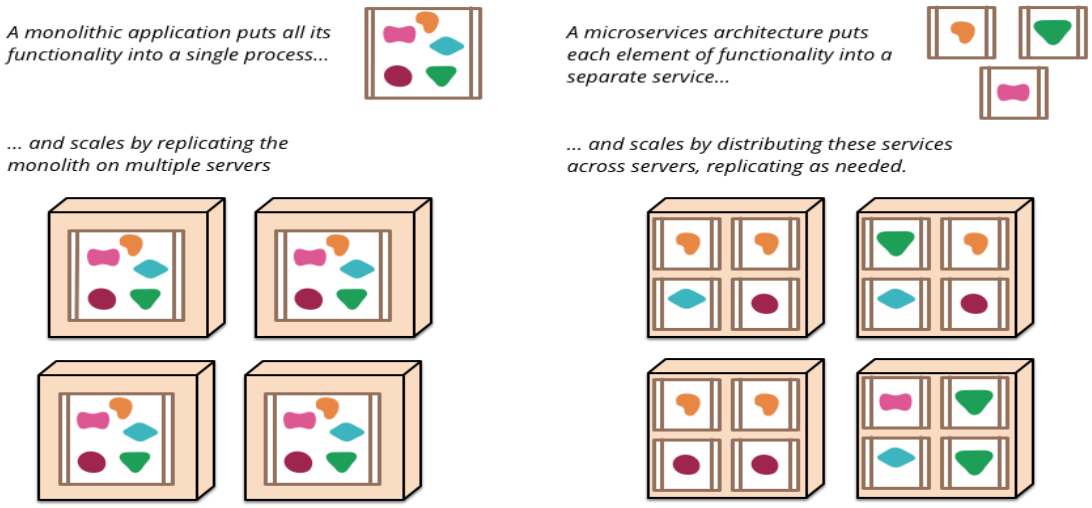


Figure 14. Monolithics and Microservices (Fowler & Lewis 2014)

3 SOFTWARE TESTING

3.1 General

Software testing is necessary to ensure the quality of a software product. Testing is the process of evaluating a system and its components including ensuring that the product satisfies the customer requirements, finding defects made during development, ensuring product performance and ensuring that the product does not return unexpected errors in the future. Software testing is not an individual activity. It is directly linked to the SDLC models, that determine how testing is organized and performed. Testing is a part of the verification and validation process. (Black, Evans, Graham & van Veenendaal 2008, 35; Software Testing n.d.)

Software can be tested manually or using automation. Manual testing means executing tests without scripts or automated tools. In manual testing, tester tests a product at the end user point of view using premade test suites/cases trying to find defects. Automation testing means executing tests using automation tools and scripts. The

benefits of automation testing are increased speed of the test execution and no human presence needed. These points lead to better cost efficiency, however, on the other hand, money investments are needed. (Types of Testing n.d.)

3.2 Testing types

Testing objectives are related to the products evaluation end quality control. The purpose of testing types is to define objective to each test level. There are multiple testing types to archive the general test objectives. Each of them focuses on a specific objective, which could be functional, non-functional or structural. Testing types can be divided in to two approaches: Static and Dynamic. (Black, Evans, Graham & van Veenendaal 2008, 46.)

Static testing techniques do not execute the code. They focus on finding defects by analyzing and examining the source code, design documents and requirement documents manually or using tools. Static testing starts early, in the verification process in the SDLC. Possible approaches are reviews, inspections and walkthroughs. Good examples of the static testing defects are syntax violations, missing requirements and non-maintainable code. (Black, Evans, Graham & van Veenendaal 2008, 57.)

Dynamic testing techniques involve code execution and focuses on functional aspects of the system. In dynamic testing, the code is executed using certain inputs and received outputs are examined and compared to expectations. Dynamic testing involves testing functions of the product of the Dynamic testing includes all testing phases in the SDLC. Dynamic testing is performed at the testing phases in the SDLC and it is also known as validation testing. (What is Dynamic testing n.d.)

3.2.1 Functional testing

Functional testing refers to methods verifying that every function of the software is working and matches the requirement specification. Functional testing covers databases, APIs and interfaces. Every functionality of the system is tested using identified inputs and outputs are verified and compared to the expected results. Functional testing can be manual or automated. Good examples are Unit testing, Integration

testing and Regression testing. (Black, Evans, Graham & van Veenendaal 2008, 46; Functional testing n.d.)

3.2.2 Non-functional testing

Non-functional testing focuses on the non-functional features and quality characteristics of the system such as efficiency, usability, maintainability and reliability. Functional testing answers how well or fast the system is working. Good examples of a test types of non-functional testing are performance testing, load testing, penetration testing and usability testing. (Black, Evans, Graham & van Veenendaal 2008, 47; Functional testing n.d.)

3.2.3 Structural testing

Structural testing focuses on the structure of the component or the system architecture. Structural testing can be categorized based on the coverage techniques: Statement coverage, Branch coverage, Path coverage and Condition coverage. Depending on the coverage technique, every statement, branch, path or condition in the software is tested at least once by using both true and false values. Structural testing is often used to measure a test coverage of structural elements and thus to complement functional testing. Structural testing often referred as a test design technique called white-box testing, because it focuses what is happening “inside the box” and tester must have the knowledge of the implementations of the software. (Black, Evans, Graham & van Veenendaal 2008, 48; Structural testing n.d.)

3.2.4 Regression testing

Regression testing occurs on every testing level. Regression testing confirms that new changes to the product work as they should and they do not affect existing features. Regression tests are the same tests that are executed earlier, which means a very large test suite. The regression test round is executed always when the code, function, environment or even when possible third-party service version changes, which is why regression testing is usually automated. (Black, Evans, Graham & van Veenendaal, 49.)

3.3 Levels of Testing

Software testing can be categorized into testing levels based on the different phases of software development activities. The purpose of testing levels is to help organize the testing process and they are performed in the different phases of the product development life cycle. Every test level analysis and design should be started during the corresponding development level. To understand it a look at the software development model called “V-model” is needed (Figure 15). V-model aims at better quality control, which minimizes project risks and costs. The testing levels are explained in more detail in the following chapters. (Black, Evans, Graham & van Veenendaal 2008, 41; What are Software Testing Levels n.d.)

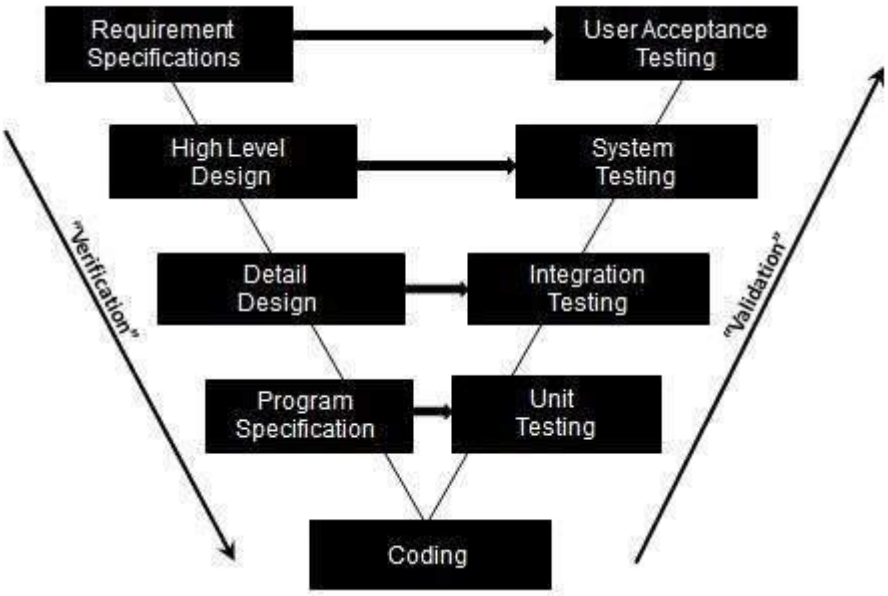


Figure 15. V-model development activities and testing levels (V-model n.d)

The development environment of the thesis includes system testing and acceptance testing. The tests focus on reliability, performance, stability, scalability and usability. Testing is performed using a test automation framework called Robot Framework and a load testing tool called Locust.

3.3.1 Unit testing

Unit testing means testing individual units. The tested units can be methods, functions or classes. The purpose of unit testing is to select the smallest part of the software and validate that the unit works as it was designed. The tested units should be isolated from other units, which means that one test only focuses on one unit. One unit is usually tested with multiple tests and inputs to ensure unit functionality also in error conditions. Unit testing is a structural testing type, because it requires deep knowledge of the system. It is usually executed by developers on their computers during the development process, which leads to rapid error fixing. Unit tests are usually run after every change in the source code, which leads to test automation using the testing frameworks/tools for reducing testing time and costs. (Unit Testing n.d.)

3.3.2 Integration testing

Integration testing is the next step from unit testing. Integration testing focuses on testing the interfaces and interactions between integrated units known as components and interactions between other system parts, e.g hardware and file system. Integration testing can be divided into two parts: component and system integration testing. Component integration testing focuses on interactions between software components, while system integration testing focuses on interactions between other systems. (Black, Evans, Graham & van Veenendaal 2008, 42).

There are multiple approaches to carry out integration testing. The most common ways are big bang and incremental strategies. In big bang strategy, all units are combined to demonstrate the final product and they are tested together at once. Big bang strategy is faster than other strategies, however, on the other hand, testing requires better planning. Incremental strategies test integrated components one by one and require external programs simulate connections. Top-down strategy starts from top-level components such as main menu and follows the architectural structure to the lower-level components. Top-down strategy requires programs called “Stubs” to answer the tested component calls. The disadvantage of top-down testing is that lower-level components are tested relatively late or developers must create temporary modules to simulate the missing lower-level units. Bottom-up strategy

starts from the basic lower-level units and approaches the top-level units. Basically, bottom-up is a reversed version of top-down strategy, which requires programs called “drivers” to create calls to the tested module. (Black, Evans, Graham & van Veenendaal 2008, 43.)

3.3.3 System testing

System testing focuses on testing the final and fully integrated software. The purpose of system testing is to check that the software corresponds to the requirement specification and finds defects in a complete system. The test cases are made from end-user point of view to simulate real-life scenarios. The software is installed in the company’s own environment which simulates the production environment. System testing contains more than 50 types of functional and non-functional tests. This leads to a challenge to select the right types for one’s project. Basically, the tester must estimate which parts of the software require the most testing in relation to the time and budget. System testing is usually performed by testing teams. (Black, Evans, Graham & van Veenendaal 2008, 44; What is System Testing n.d.) Figure 16 illustrates types of system testing.

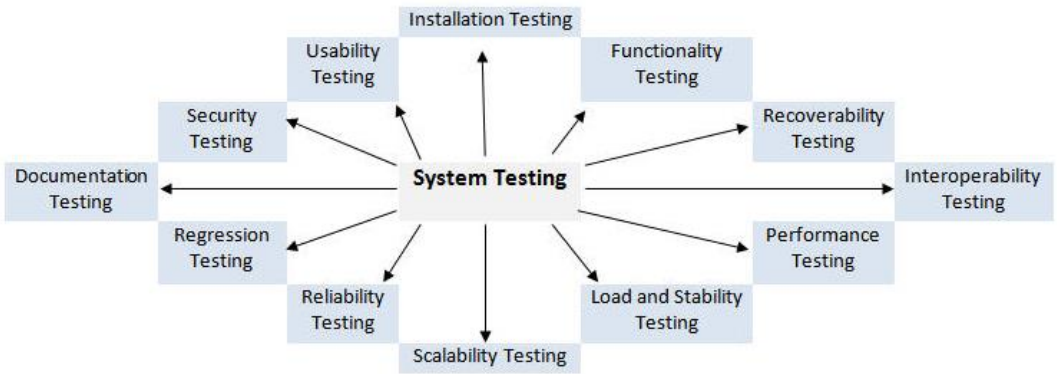


Figure 16. Types of system testing. (Types of System Test 2017)

3.3.4 Acceptance testing

Acceptance testing is the last SDLC level where the customer evaluates the final product against the requirement specification. The purpose of acceptance testing is to determine if the final product meets the requirements, customer needs and if it is ready for production. There are multiple types of acceptance testing: user acceptance testing, operational acceptance testing, alpha and beta testing. User acceptance testing focuses on evaluating the product's functionality against requirements. User acceptance testing is performed by the customer in a testing environment, which demonstrates production environment. Operational acceptance testing focuses on evaluating the product's manageability, supportability, security and ability to recover. The purpose of alpha testing is to find and fix bugs before the release to the end-users. It is not performed on the final version of the product. It is performed inside a company, by the users who replicate potential users. Beta testing is performed after alpha tests. Beta testing is performed by the end-users, who try to find faults in the product and give improvement suggestions. After the tests are done, the product's feature request and bug fixes are delivered to the developers who will perform the necessary actions. If there are some changes on the product, testers perform another testing round called regression testing. After successful acceptance testing round, the product and related documents are delivered to the customer. (What is User Acceptance Testing n.d; Why Operational Testing n.d).

4 CONTAINERS

4.1 Background

Operating-system-level virtualization or better known as container technology has been rising in the IT sector for a few years mainly because of technology called Docker. The biggest problems with virtualization technologies such as virtual machines (VM) are that they cannot be reliably moved to other environments. The production environment might have a different operating system, libraries or other dependencies, which often lead to multiple environment-specific configurations or in

the worst case, even making the new product. Containers solve this problem by containing an entire runtime environment. Containers encapsulate an application and all its dependencies in one package. Containers can be deployed on any environment regardless of the environment's hardware. If containers are observed from cloud computing perspective, it is easy to say that containers belong under PaaS section. Shortly, if they are working on the developer computer, they will also work in the production environment. Containers share the host's operating system (OS) kernel with other containers, unlike virtual machines (VM) where every VM uses their own OS (Figure 17). This leads to the support of multiple containers on same the host. Another major advantage with containers is the size, which is usually only a few hundred megabytes whereas VMs usually are few gigabytes. This leads directly to better performance and easy scalability. (What is a Container n.d.)

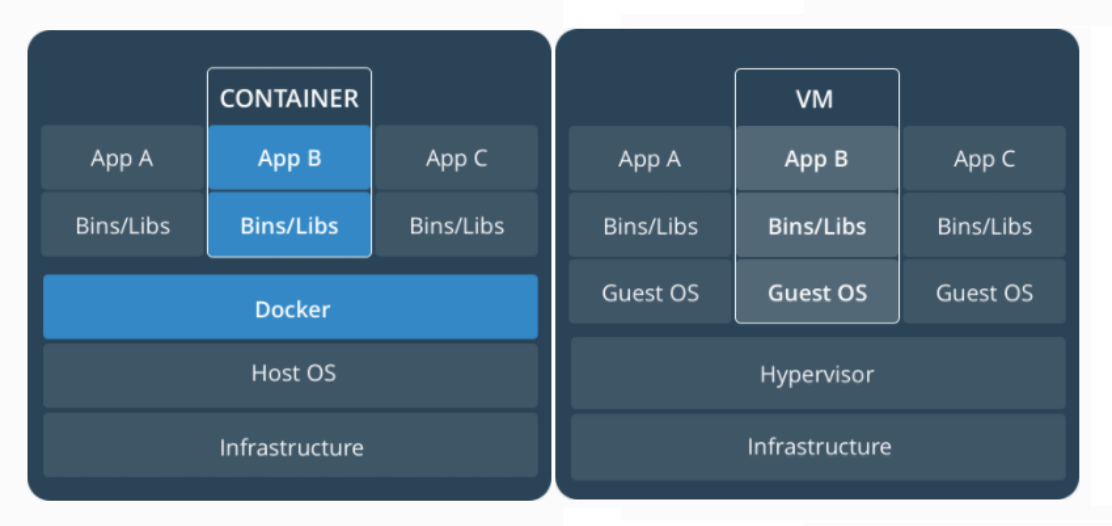


Figure 17. Docker containers vs VMs (What container n.d, modified)

Container technology is not a new thing, however, often people think that Docker and containers stand for the same thing. Container technology was invented in 2001 in a Linux VServer project. According to VServer, the first change log states:

> *This is the first release of a very interesting project: Running a several general purpose Linux server on a single box with a high degree of Independence and security. Check it out. (Vserver 0.0 changes log 2001).*

The follow-up for VServer was known as a generic process container also known as cgroups. Cgroups was developed in 2006 by Google engineers Paul Menage and Rohit Seth. Basically, cgroups was a kernel feature that allowed processes to be grouped together with isolated system resources. (Hildred 2015.)

According to cgroups version 1 documentation, cgroups was defined as:

> *Control Groups provide a mechanism for aggregating/partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behaviour. (cgroup-v1.txt n.d).*

The next steps were taken in 2008 when Linux kernel user namespaces was implemented, which enabled processes to have their own group of users and which also allowed container root privileges. Linux Containers (LXC) project was launched later in 2008. LXC layered userspace tool above kernel cgroups and namespaces. (Hildred 2015.)

The first modern version of containers was launched in 2013. It was an open-source project called Docker. Docker is presented in more detail in next chapter. Nowadays, containers are commonly used in companies and the technology is constantly evolving. For example, Google runs all their services in containers and it also offers a free orchestrating tool called Kubernates. The downsize of the rapid increase of technology is that there are undiscovered problems, for example Docker security was questionable and orchestrating containers was hard without programs. (Hildred 2015.) Nowadays, there are multiple orchestration tools available such as Docker swarm, Kubernates and Mesosphere.

4.2 Docker

Docker is an open-source client-server application for creating, deploying and running containers. The Docker server, which is also called as Docker daemon, manages and runs containers. The Docker client is used to manage the Docker server actions. The client is controlled via command line tool, using simple commands. The Docker daemon and the Docker client communicate over UNIX sockets or using a network interface. One Docker daemon can manage multiple containers, however, if the infrastructure consists of multiple servers, one client can communicate remotely with them. (Docker architecture n.d; Kane & Mathias 2015, 11.)

Other important section is the Docker registry. The purpose of the registry is to store Docker images and images metadata (Kane & Mathias 2015, 11). Images can be stored on a local host machine, however, is recommended to export them to external service where other developers can access them. These registries can be self-hosted and there are also multiple external services available, e.g Docker Hub, GitLab Container Registry and Google Container Registry. Figure 18 presents the Docker workflow.

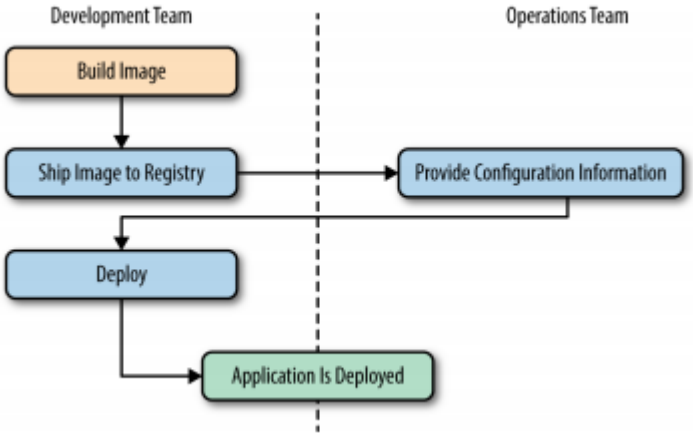


Figure 18. Docker deployment workflow Kane & Mathias 2015, 9)

4.3 Docker containers

The basic functionality of the containers was described earlier but to really understand how Docker container work, Docker images need to be inspected. Docker containers are instances from images. Basically, the functionality of the container is described in Docker image. Docker images are made up of filesystems layered over each other. Figure 19 presents the layers of the Docker container. The bottom is a shared kernel layer. Image layers are read-only, and the container layer is empty at a container startup. Every change made in a running container creates a new container layer with a new cryptographic content hash on top of the existing one. Container layers disappear after the container is deleted. (Images and layers n.d.)

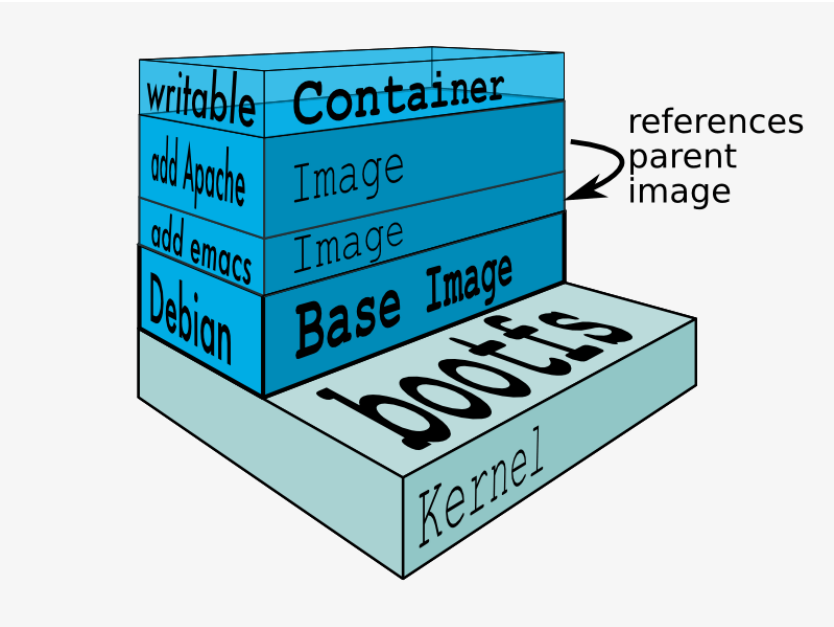


Figure 19. Docker image layers (Parent image n.d.)

As can be seen in Figure 19, there are multiple image layers. How is it possible? To understand that files used to create images are looked into next. These files are called Dockerfiles. Figure 20 presents Dockerfile of the Locust container used in this thesis. Each line of the Dockerfile creates a new layer, which stack on top of each other because the lower levels are read-only. (Images and layers n.d.)

Dockerfile 277 Bytes

```
FROM ubuntu:16.04
RUN apt-get update
RUN apt-get -y install git python-pip libevent-dev python-all-dev software-properties-common
RUN pip install locustio pyzmq

COPY start.sh /home/root/start.sh
RUN chmod +x /home/root/start.sh

EXPOSE 8089

ENTRYPOINT ["/home/root/start.sh"]
```

Figure 20. Dockerfile of the Locust container

After Dockerfile has been created, it must be built to an image. This is done by executing Docker Engine's simple docker build command. (see Figure 20). It can be clearly seen that each step creates a new layer with a hash. It is obvious that layers affect directly the size of the image. For keeping image sizes small, users should combine all same commands together and delete the unneeded installation components on the next layer. After the image is successfully built, it can be stored locally or pushed in to third party registries such as GitLab registry or Docker Hub (see Figure 21).

```
Removing intermediate container 2d0c00e342b8
Step 5/8 : COPY start.sh /home/root/start.sh
 ---> 6d30811db5c3
Removing intermediate container e5b02b42dee6
Step 6/8 : RUN chmod +x /home/root/start.sh
 ---> Running in 849d7dc089b9
 ---> 481115b39160
Removing intermediate container 849d7dc089b9
Step 7/8 : EXPOSE 8089
 ---> Running in 73b2f710eb25
 ---> 560f39a0bcec
Removing intermediate container 73b2f710eb25
Step 8/8 : ENTRYPOINT /home/root/start.sh
 ---> Running in 8b7f6f87ab50
 ---> f863d1385f2e
Removing intermediate container 8b7f6f87ab50
Successfully built f863d1385f2e
$ docker push registry.gitlab.com/jamkit/locustio:latest
```

Figure 21. Locust container build steps

Single containers can be run from command line with docker run command, however, it does not support container stacks. For that, Docker offers a tool called Docker Compose. Docker Compose uses docker-compose.yml file for managing the application stack. Compose file defines all containers with possible options and parameters and links between containers (see Appendix 1).

5 ASSIGNMENT

5.1 Overview of the architecture

The assignment was to get familiar with containers and cloud services and develop modern DevOps development environment for training that includes Contriboard. The selected services and tools had to be free so students can use them. The development environment tools were decided almost completely in advance. There were comprehensive test suites for certain tools, made during the development of Contriboard and it was logical to select some tools that are familiar for student.

The basic idea of the development environment is when a user commits the Contriboard code changes in version control system, a continuous delivery pipeline is launched. Depending on the modified component, a new container image will be built and stored. Next, the SUT that includes new changes is deployed with monitoring tools. The next steps execute functional and regression tests against the SUT. If the tests pass, the final product is deployed with monitoring tools (see Figure 22). The environment implementation is described in chapter 5.3.

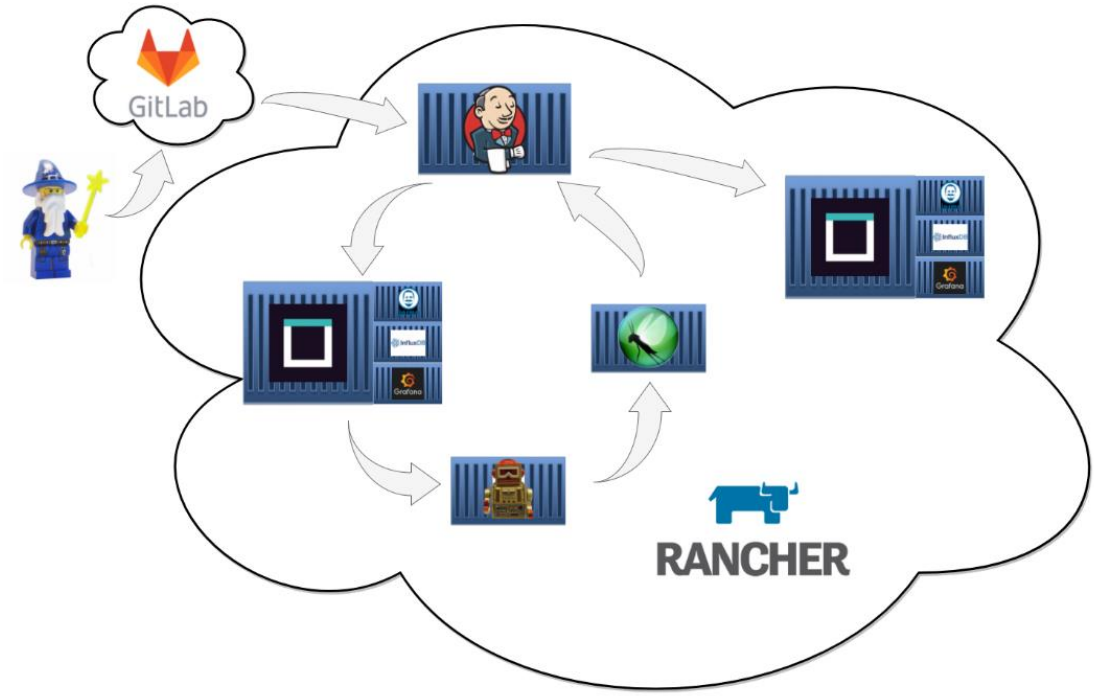


Figure 22. Environment description

5.1.1 Rancher

Rancher is an open source container management platform developed by Rancher Labs. Rancher offers advanced orchestration tools for infrastructure and containers, e.g networking, storage, load balancer and Domain Name System (DNS) tools. Users can create multiple environments and use private hosts or public host providers, e.g Amazon and Digital Ocean. Private hosts must be added manually and public service hosts can be added using service API keys (Figure 23). Hosts must be Linux based, because orchestration services use Docker containers on their services. Rancher supports the most popular container orchestration frameworks, e.g Docker Swarm, Google Kubernates, Apache Mesos and Rancher's own framework called Cattle. (Overview of Rancher n.d.)

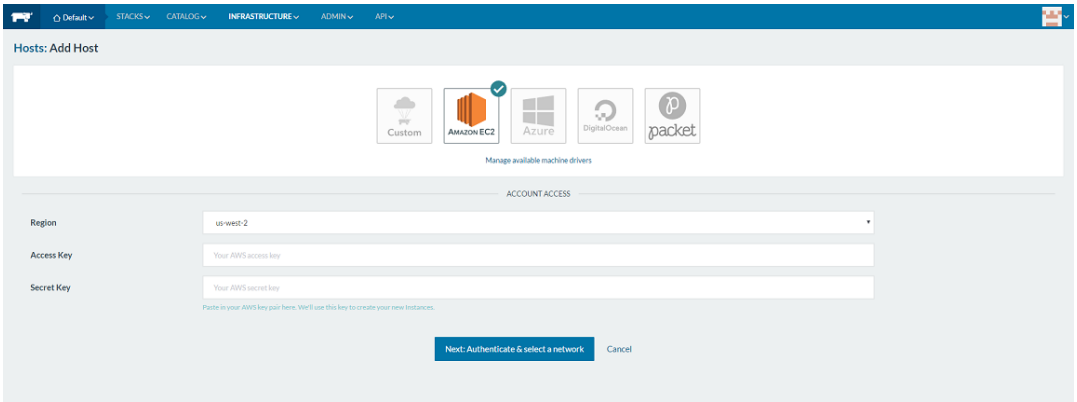


Figure 23. Rancher, add new host view

With Rancher, users can manage a complete lifecycle of containers. Users can deploy, clone, stop and destroy containers using web UI or API. Rancher supports Docker compose files and Rancher's own Rancher compose files, which are multi-host version of Docker compose (Rancher Compose n.d). User interface is simple and offers monitoring tools for containers.

Rancher also offers an application catalog. With application catalog users can deploy an application stacks fast by pressing a single button (see Figure 24). There are two catalog types: Rancher certified catalog and community catalog. Rancher catalog is managed by rancher and it offers multiple tools for managing network and containers. Community catalog is managed by community and it has multiple software available, e.g Jenkins, GitLab and multiple DNS services. Users can create private collections and add their own containerized applications in it. (Rancher Catalog n.d.)

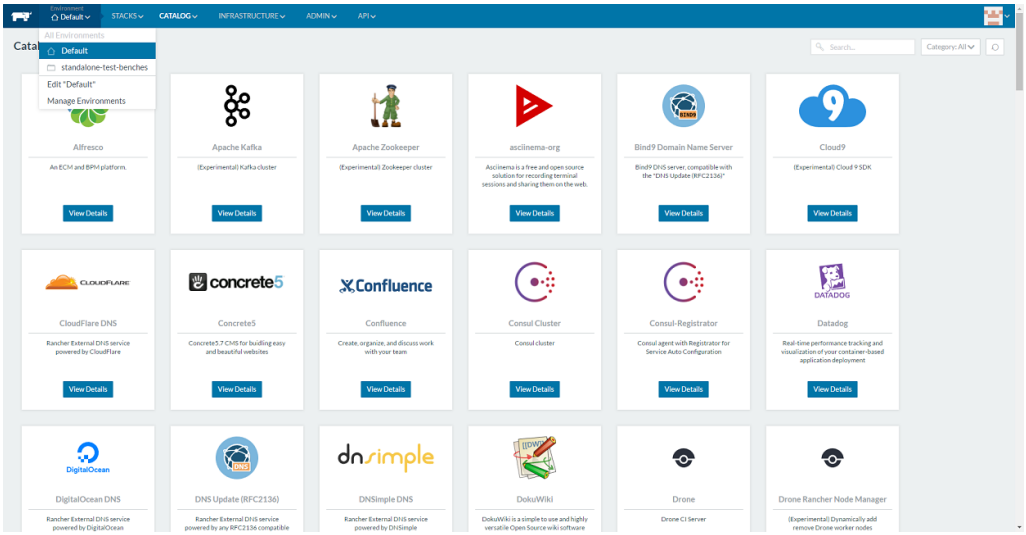


Figure 24. Rancher application catalog view

Rancher was selected to the development environment, because it is open source and it was already used during the JAMK's summer course called Challenge Factory. Rancher is not as advanced as its competitors such as Kubernates or Mesos. Rancher offers a lightweight framework, simple API and most importantly simple UI, which usage does not require deep knowledge.

5.1.2 GitLab

GitLab is an open source software for managing Git repositories. GitLab offers multiple features, e.g. repository management, issue tracker and wiki documentation. GitLab's web UI is simple and users can create and edit all files using it (see Figure 25). Users can invite other people to the project and set different permissions to them. GitLab also supports forking and merging repositories and project imports from other services, e.g GitHub and Bitbucket. (Products n.d.)

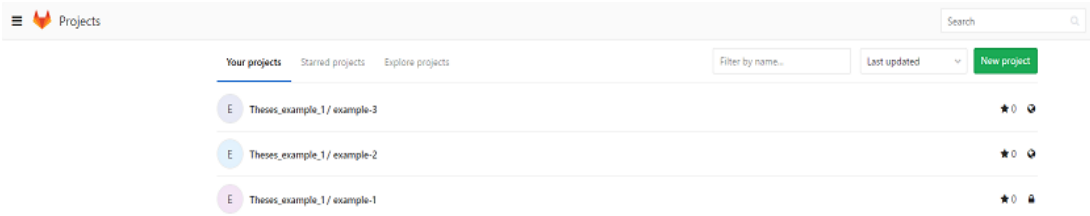


Figure 25. GitLab web UI

GitLab also offers free private repositories, container registry, webhooks and integrated CI and CD service called GitLab-CI. The container registry was announced in fall 2016 and made it possible to store and distribute container images. Webhooks allow user to setup pipelines between other services. Webhook sends http POST request to the predefined IP address when the defined action takes place. These trigger actions can be new commits, CI-pipeline status changes or even a new issue creation. (GitLab Continuous Integration n.d.)

GitLab-CI is a service for building, testing and deploying applications. GitLab-CI can be configured by using.yml file in the repository and it requires an isolated machine called GitLab runner for executing jobs. GitLab Runners are hosted by users and registered to GitLab; however, GitLab offers few runners for free in their hosted service. GitLab is available as Community Edition and Enterprise Edition which can be installed in the user's own environment. GitLab also offers a free service hosted by them. The Community Edition is free to use, and the Enterprise Edition demands subscription. The Enterprise Edition offers multiple extra features, e.g. workflow controls and fast support. Shortly, GitLab offers an effective all-in-one service without a need for external services. (GitLab Continuous Integration n.d.)

GitLab was selected to be a part of the development environment, because of GitLab-CI tool and the container image registry. These features made it possible to aggregate all in one place without a need for external services, e.g. container builder, container registry and CI/CD service. The selected service was GitLab's Enterprise Edition package hosted by GitLab, because repositories are available from anywhere and free and it offers maintained GitLab Runners for running jobs.

5.1.3 Jenkins

Jenkins is an open source automation server for continuous integration. Jenkins is used to automate builds, tests and deployments of the software projects. The project was originally known as Hudson, which was started by Kohsuke Kawaguchi and developed by the community and Sun/Oracle. In 2011, developer community separated from the project and started new project called Jenkins. Jenkins is written in Java and running it requires the server to have Java Runtime Environment (JRE) installed. (Ferguson Smart 2011, 3.) Jenkins offers a simple web UI, where jobs can be created, managed and monitored. Jenkins features can be extended by using third party plugins for example GitLab plugin which connects with GitLab API and allows jobs to start when a new change is committed in to the repository. Figure 26 illustrates the main view of Jenkins web UI, where users can manage and monitor Jenkins and its jobs.

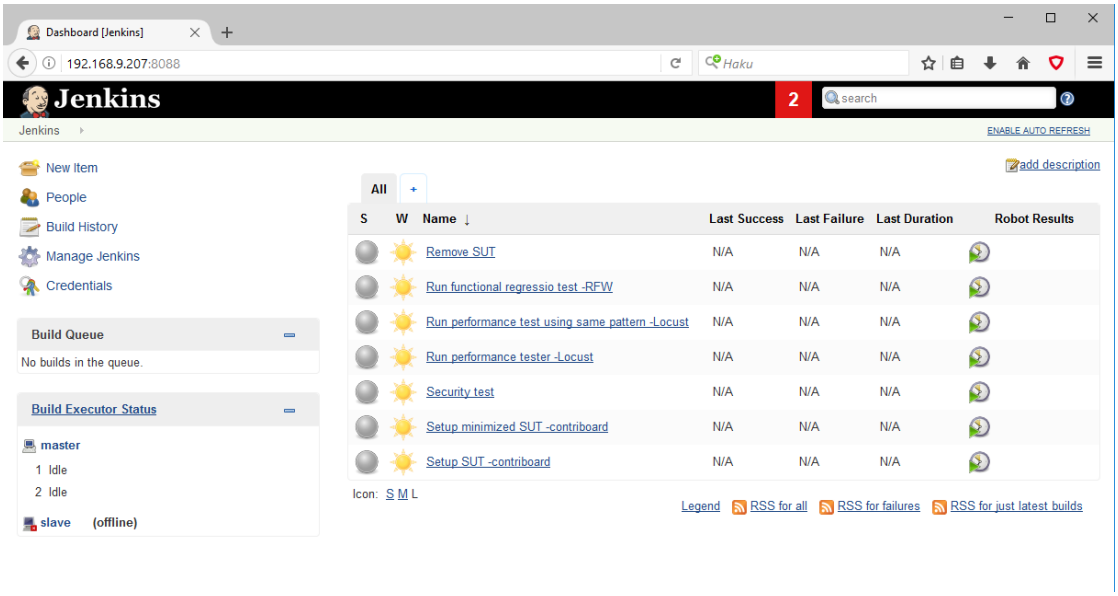


Figure 26. Jenkins web UI

Jenkins jobs are used to build, test or deploy projects. Every stage of the project is usually defined at one job, e.g. Figure 26 consists of a deploying different kind of system under tests (SUT), multiple variations of tests and removing SUT. Jenkins jobs are usually configured by using execute shell build-step, which is build-in shell command

line. Users can chain jobs by defining post-build actions where the next job is automatically started at a certain time or depending on the status of previous job which the shell command returns. Both build and test process have three possible statuses and user can also define exit conditions using execute shell. For example, in thesis development environment, Jenkins Locust test job executes the shell step analyzes report from the Locust and if its failure rate is under 5 percent, Jenkins marks the job status to pass.

Jenkins jobs can be distributed using Jenkins agents to reduce the workload on the master computer. This way the master computer running Jenkins delegates the job execution to the so-called slave computer or in this thesis case, to the other container. Jenkins has a built-in SSH client so slaves can be started using SSH. This method requires a connection between master and slave computer, and users must setup SSH keys between computers. Launching slaves with SSH connection does not require manual configurations on the slave computer because Jenkins master automatically configures it when connected. If the master computer cannot establish SSH connection with a slave computer, another method is to manually install and configure a slave daemon to the slave computer and take the connection from slave to master. This method can be required if a slave computer is in a different network than the master computer and does not have a public IP address. (Distributed builds 2016.)

Jenkins was selected to part of the development environment mainly because it is used in JAMK's courses and students are familiar with it. GitLab-CI tool could handle the same jobs with Rancher API easily, but Jenkins offers a possibility for multiple jobs, which can be also triggered manually and with visual aspects as well. Jenkins plugins also offer effective presentations for the test results.

5.1.4 Robot Framework

Robot Framework is an open-source a test automation framework, which uses keyword-driven syntax, where functions are called with actual words. It is developed for acceptance testing and acceptance test-driven-development, however, it can be used on any automation testing, e.g. system testing such as the thesis development environment does. Robot Framework was originally developed by Nokia Networks and nowadays, it is sponsored by Robot Framework Foundation. (Robot Framework n.d.) These words are called keywords, which contain all functionality of the test step. In keyword-driven testing, user creates data files using keywords, which contain all functionality to complete one test step (Keyword-driven testing n.d). There are high and low-level keywords and technical keywords. High-level keywords are used to test a specific part of a system and low-level keywords are used to separate the required functionality testing to the several low-level keywords to keep the tests cases minimal (Robot Framework – Test automation the smart way n.d.) There are multiple external libraries available for Robot Framework; however, thesis development environment uses Selenium keyword libraries, which contain the basic functions for processing browsers. Users can extend these libraries by writing new keywords using Python or Java. Robot Framework automatically creates log and report files after tests are done. Users can see the results of the test suite and all steps sorted by keyword layers. (Robot Framework Quick Start Guide. 2017.) Figure 27 presents a syntax example used in testing Contriboard.

```
*** Settings ***
Resource          resource.txt

*** Test Cases ***
Open Login Page
    [Tags]    reg
    Open Browser To Login Page

Register User Testbaboon
    [Tags]    reg
    Register User    testbaboon@test.com    t3stmonkey
    Log Out

Close
    [Tags]    reg
    Close Browser
    [Teardown]
```

Figure 27. Robot Framework syntax example

N4S@JAMK's earlier product Contriboard was tested using the Robot Framework, which is why it was an obvious choice to be part of the development environment, because there were hundreds of tests made for it. Students also study it during JAMK's software testing course.

5.1.5 Locust

Locust is an open source user load testing tool. Locust is a lightweight and distributed framework for testing websites and services. Locust can be used for load-testing and figuring out how many concurrent users a system can handle. Locust is event based, so one machine can simulate multiple concurrent users. Locust can also be used distributed, where multiple machines run the same tests against the SUT. These machines are controlled by one master machine. Locust test scripts are written in Python. Locust can be managed with a command line tool or with a Web UI. Users can select the number of concurrent users to simulate and set their hatch rate. While tests are running, users can see measures such as requests/second and failed requests. When tests are executed, Locust provides a report. (What is Locust 2015.) Figure 28 presents Locust web UI.

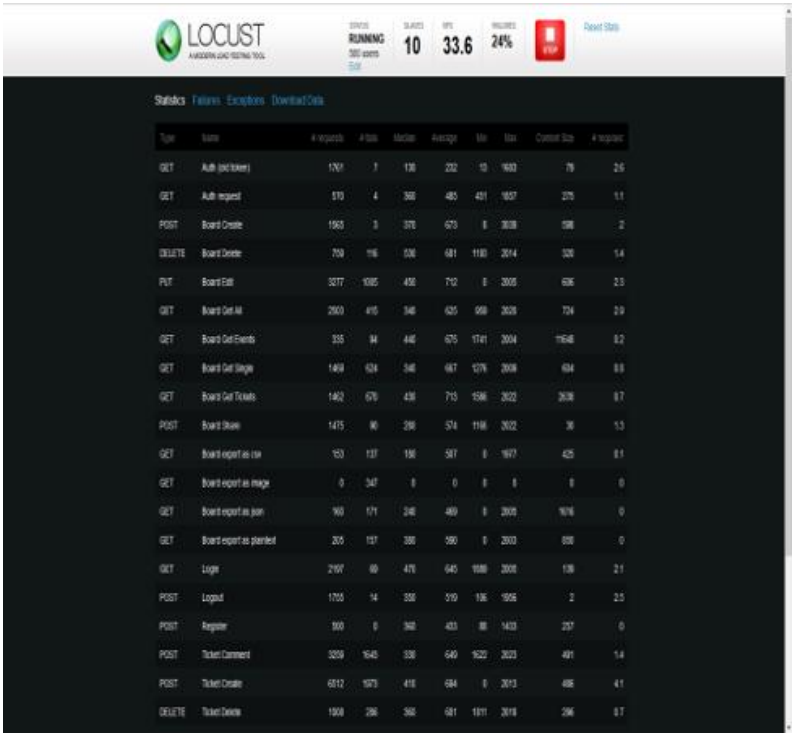


Figure 28. Locust web UI

Locust was selected to be part of the development environment, because just like in Robot Framework's case, Contriboard was earlier tested using it; hence, there was a comprehensive test suite available.

5.1.6 Jarmo

Jarmo is a simple daemon such as StatsD (see Figure 29). Jarmo is developed by N4S@JAMK project team for collecting and aggregating application data from Contriboard. Contriboard has two middleware components: Jarmo Express and Jarmo Socket.IO. These components collect and send Contriboard's statics such as response times, component status and the amount of connections to Jarmo in JSON format via UDP. Jarmo parses the data and adds a timestamp to it. After that, Jarmo InfluxDB Reporter module pushes it to the InfluxDB, which is a time series based database. (Hartikainen 2015).

Figure 29. Jarmo-logo

5.1.7 Grafana

Grafana is an open source analyzation and visualization tool for time series data. Grafana supports multiple data sources such as InfluxDB, Graphite and Elasticsearch. Grafana allows users to make dashboard templates, which allow users to define variables that are substituted when making queries. This way dashboards are interactive and dynamic. Application metrics can be visualized in dashboard by multiple panels and styles such as graphs, bars and tables. (Grafana Documentation n.d.) Figure 30 presents Grafana dashboard view. Grafana has an official container available in Docker Hub. Grafana was an obvious choice for monitoring Contriboard, because it supports InfluxDB which Jarmo used to store Contriboard's metrics.

Figure 30. Grafana dashboard

5.2 Developed containers

5.2.1 Jenkins containers

Jenkins container stack consists of Jenkins-master, Jenkins-slave, Jenkins-data and Jenkins-nginx container. Jenkins has official container available, however, it does not support the development environment needs. Developed images are based on JAMK's Challenge Factory- project team called "IoTitude" containers, but are heavily modified. As said earlier, Jenkins stack consist of 4 containers; Jenkins-master container is an application container, Jenkins-slave is so called node container, which is dedicated to running Jenkins jobs, Jenkins-data container ensures data persistence and Jenkins-nginx is a simple proxy server.

Jenkins-master container image is based on official Jenkins image version 2.32. Dockerfile updates the package list, creates a few required folders for Jenkins and adds the permissions to user Jenkins. Dockerfile also defines the required plugins for Jenkins install-plugins script and declares two environment variables, which set Java heap size and disable Jenkins SetupWizard on start (see Figure 31).

```
FROM jenkinsci/jenkins:2.32

USER root
RUN mkdir /var/log/jenkins && \
mkdir /var/cache/jenkins

RUN chown -R jenkins:jenkins /var/log/jenkins && \
chown -R jenkins:jenkins /var/cache/jenkins

RUN apt-get update && apt-get install -y sudo

USER jenkins
RUN /usr/local/bin/install-plugins.sh git matrix-auth workflow-aggregator ssh-slaves gitlab-plugin envinject robot

ENV JAVA_OPTS -Xmx8192m -Djenkins.install.runSetupWizard=false
```

Figure 31. Dockerfile of the Jenkins-master container

As said earlier, Jenkins-slave container is dedicated to running Jenkins jobs. It is based on Ubuntu 14.04 and has Java8 and SSH daemon installed for connection between master. The container also has a Docker and Rancher Compose installed, because without them Jenkins job's execute shell does not understand required commands. Appendix 2 presents the Dockerfile of the Jenkins-slave container.

Jenkins-data container is used to store all the data. At the start, it delivers configurations for jobs and node. Image uses Debian Jessie base image. First run command adds a new user called Jenkins. After that, Jenkins folder is created with permissions and the folder is mounted to Jenkins-master container folder using Docker VOLUME command. Finally, pre-configured jobs and setting are added (see Figure 32).

```
FROM debian:jessie
MAINTAINER JAMKIT

# Create the jenkins user
RUN useradd -d "/var/jenkins_home" -u 1000 -m -s /bin/bash jenkins

# Create the folders and volume mount points
RUN mkdir -p /var/log/jenkins
RUN chown -R jenkins:jenkins /var/log/jenkins
VOLUME ["/var/log/jenkins", "/var/jenkins_home"]
ADD config /var/jenkins_home/
USER jenkins
CMD ["echo", "Data container for Jenkins"]
```

Figure 32. Dockerfile of the Jenkins-data container

The last container in Jenkins stack is called Jenkins-nginx. Nginx is an HTTP and reverse proxy server. The container image is based on Linux CentOS and only Nginx and required Nginx and Jenkins configurations are added. The source code is available in JAMKIT's GitLab: https://gitlab.com/JAMKIT/jenkins-nginx.

5.2.2 Locust container

Locust Dockerfile was created using the latest Locust installation instructions. The base image is the latest official Ubuntu release (16.04). The first run command updates Ubuntu's package list, which contains the latest information on the packages and dependencies. The next step installs Git and multiple packages that Locust requires for installation and running. On the next step, Locust and recommended package pyzmq are installed from Python Package Index using pip. Locust default port 8089 is opened using EXPOSE command. Dockerfiles always need CMD or ENTRYPOINT command, which identifies which executable should be run when the container starts. Without those, the container just exists after run command. In this case, ENTRYPOINT points to the Bash script called start.sh, which is copied and the executable permission is given (see Figure 33).

```
FROM ubuntu:16.04
RUN apt-get update
RUN apt-get -y install git python-pip libevent-dev python-all-dev software-properties-common
RUN pip install locustio pyzmq

COPY start.sh /home/root/start.sh
RUN chmod +x /home/root/start.sh

EXPOSE 8089

ENTRYPOINT ["/home/root/start.sh"]
```

Figure 33. Dockerfile of the Locust container

As said earlier, the container uses Bash script for describing launch executable and parameters. A script uses environment variables that users must define; URL to the repository where tests are located and which is cloned, Repository folder where locustfile is located, IP address of the tested service, Number of request made during the test run, Number of simulated users and hatch rate (see Figure 34). Variable LOCUST_TARGET_HOST comes from Jenkins that asks for it from Rancher API.

A script clones the defined Git repository and starts the Locust tests with defined parameters. Finally, after test is done, the script moves the test logs to the Jenkins-data container. This way Jenkins can access the logs and analyze them.

```
#!/bin/bash

GIT_TEST_REPO=${GIT_TEST_REPO:=}
GIT_REPO_NAME=${GIT_REPO_NAME:=}
LOCUST_LOCUSTFILE=${LOCUST_LOCUSTFILE:=}
LOCUST_TARGET_HOST=${LOCUST_TARGET_HOST:=}
LOCUST_NUM_REQUEST=${LOCUST_NUM_REQUEST:=}
LOCUST_USERS=${LOCUST_USERS:=}
LOCUST_HATCH_RATE=${LOCUST_HATCH_RATE:=}
LOCUST_OPTIONS="-f $LOCUST_LOCUSTFILE -H $LOCUST_TARGET_HOST TeamboardUser --no-web -c $LOCUST_USERS \
-r $LOCUST_HATCH_RATE --print-stats --only-summary --num-request=$LOCUST_NUM_REQUEST --logfile=results.log"

mkdir -p /home/root/test
cd /home/root/test
git clone $GIT_TEST_REPO
cd /home/root/test/$GIT_REPO_NAME

echo "Running..."
locust $LOCUST_OPTIONS

echo "Moving logs to Jenkinsdata..."
rsync -a /home/root/test/$GIT_REPO_NAME/ /var/jenkins_home/locustReports/
```

Figure 34. Locust container starting script

5.2.3 Robot Framework container

Robot Framework container includes both Mozilla Firefox and Google Chrome browsers. Because browsers are run in container, they will run in headless mode using Xvfb display server. Installation of Xvfb and browsers required the installation of multiple libraries and packages. Robot Framework and multiple dependencies were installed from the Python Package Index using pip. Robot Framework uses Selenium WebDriver and keyword libraries. Both browsers require Selenium WebDriver extensions called GeckoDriver and ChromeDriver. These extensions were downloaded from their official sites, extracted, given all permissions and moved to the /usr/bin folder. Contriboard functional tests included the extended keyword library for Selenium2Library, which was added in /usr/local/bin/lib/python2.7/dist-packages/ path. All steps of the Dockerfile can be found in Appendix 3.

Just like in Locust container's Dockerfile, Robot Framework container's Dockerfile uses Bash script in ENTRYPOINT parameter. A script uses environment variables what users must define: URL to the repository where tests are located and which is cloned, Repository folder where test suite starting script is located, Name of the test suite

starting script, Name of the resource file, Folder where test files are located, Name of the Jenkins folder (see Figure 35).

A script starts Xvfb display server and clones the defined Git repository. After that, the script opens the cloned resource file and replaces the SUT_IP_ADDRESS with the IP address from Jenkins and Rancher API. Next, a script starts the test suite and when the tests are completed, it moves the generated log files to the Jenkins-data container.

```
#!/bin/bash
set +e

#ENV variables
GIT_TEST_REPO=${GIT_TEST_REPO:=}
GIT_REPO_NAME=${GIT_REPO_NAME:=}
TEST_SCRIPT=${TEST_SCRIPT:=}
RESOURCE_FILE=${RESOURCE_FILE:=}
SUT_IP_ADDRESS=${SUT_IP_ADDRESS:=}
TEST_FILES_FOLDER=${TEST_FILES_FOLDER:=}
JENKINS_FOLDER=${JENKINS_FOLDER:=}
DEFAULT_PATH="/home/root/test"

echo 'Starting Xvfb ...'
export DISPLAY=:99
2>/dev/null 1>&2 Xvfb :99 -shmem -screen 0 1920x1200x16 &
exec "$@"

mkdir -p $DEFAULT_PATH
cd $DEFAULT_PATH

echo 'Cloning test repo...'
git clone $GIT_TEST_REPO

#change resource.txt ip address
sed -i "s/IP_ADDRESS/$SUT_IP_ADDRESS/g" $DEFAULT_PATH/$TEST_FILES_FOLDER/$RESOURCE_FILE

echo 'Starting robot tests...'
cd $DEFAULT_PATH/$GIT_REPO_NAME
sh $DEFAULT_PATH/$GIT_REPO_NAME/$TEST_SCRIPT

echo "moving test results to jenkins"
mv $DEFAULT_PATH/$TEST_FILES_FOLDER/log.html $JENKINS_FOLDER/log.html
mv $DEFAULT_PATH/$TEST_FILES_FOLDER/output.xml $JENKINS_FOLDER/output.xml
mv $DEFAULT_PATH/$TEST_FILES_FOLDER/report.html $JENKINS_FOLDER/report.html
```

Figure 35. Robot Framework container starting script

There were multiple problems during the development process. Xvfb does not know how to launch headless Google Chrome for some reason, the problem was solved using Google Chromium launching script instead of Chrome's. Other big problem that still exists is that browsers, Selenium and third-party WebDrivers are developed by different companies and the versions change all the time, which leads to compatibility problems. For example, one known problem is that the latest GeckoDriver with the newest Firefox does not support Selenium2Library Moveto or Double-click keywords, which worked in the earlier versions.

5.2.4 InfluxDB container

InfluxDB container was developed for storing data from Jarmo. Jarmo parses the Contriboard's data and the reporter module pushes it in to the InfluxDB. Grafana uses that data for visualizing Contriboard's metrics. InfluxDB has an official Docker container available, however, the data that Jarmo and its middleware components are handling is in JSON format and InfluxDB's JSON write protocol was replaced by the line protocol after version 0.9. For that reason, it was better to build the new container with the old version of InfluxDB than to start to modify Jarmo and its components without deep knowledge of them.

InfluxDB container image is based on Ubuntu 14.04. The first run command updates Ubuntu's package list, which contains the latest information of the packages and dependencies. After that, wget is installed in order to download InfluxDB package. Next, InfluxDB is downloaded, extracted and the required ports are opened. Starting script is copied and permissions granted for it. Just like in other cases, ENTRYPOINT is defined by the Bash script (see Figure 36). A script starts the InfluxDB service and creates a database, user and grants privileges for the user. For some unknown reason, the writer failed to keep the container running after InfluxDB was started. Now a script's last line is an empty tail command for keeping container running, which is not a good solution, because if InfluxDB service crashes, the container stays alive.

```
FROM ubuntu:14.04

RUN apt-get update
RUN apt-get -y install wget

RUN wget -q http://influxdb.s3.amazonaws.com/influxdb_0.9.0_amd64.deb
RUN dpkg -i influxdb_0.9.0_amd64.deb

COPY ./scripts/ /home/root/scripts
RUN chmod -R 755 /home/root/scripts

EXPOSE 8086
EXPOSE 8083


ENTRYPOINT ["/home/root/scripts/start.sh"]
```

Figure 36. Dockerfile of the InfluxDB container

5.3 The development environment setup

5.3.1 Rancher

Rancher was installed in the server by using an official Rancher container, which requires a Docker engine installed in the server. Docker was installed in the server by using official Docker installation instructions. Next, Rancher was installed by using simple docker command:

sudo docker run -d --restart=unless-stopped -p 8080:8080 rancher/server

After the command, Rancher was fully functional. Next other servers were added in the Rancher Cattle by generating a Docker command in Add Host section. Because the command was a Docker command, Docker must have been installed in other servers as well. After the generated commands were pasted in the servers, the servers appeared in Rancher's infrastructure section (see Figure 37).

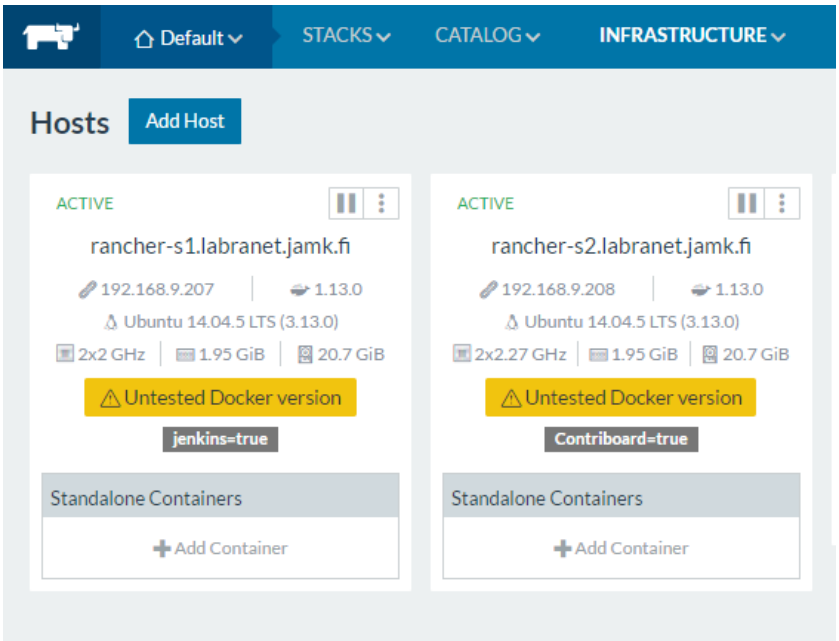


Figure 37. Rancher's infrastructure view

Servers were labeled to Jenkins and Contriboard. That way the containers can be launched on the desired server. The development environment also needs the Rancher environment API key for deploying containers. Rancher has two API keys: Account and Environment. There was no need for Account API key, and Environment API key was generated under API menu by simple pressing add environment API key and create.

5.3.2 Jenkins

Jenkins application stack was launched using docker-compose.yml file (see Appendix 1). User must define four environment variables, needed for deploying containers later: RANCHER_HOST, RANCHER_API_USER, RANCHER_API_PASSWORD and RANCHER_STACK_NAME. One important factor in Dockerfile is that the host's docker socket is shared with slave-container. This way a slave-container can deploy container directly to the host.

Rancher host is the URL of the server where Rancher is running. Rancher API user and password were generated earlier, while setting up Rancher. Modified docker-compose file is imported in docker-compose section and named after the environment variable (see Figure 38).

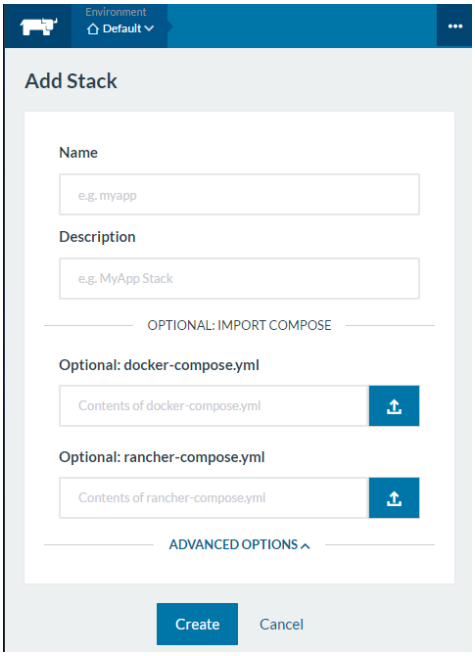


Figure 38. Rancher Add Stack

When Jenkins containers are started, the next step is to configure the connection between the master and slave container. On the left side is the field Build Executor Status, where the slave is offline. By clicking the slave image a new window opens with configure button. That button opens a configure window. All other options are pre-configured, however, host credentials must be given manually (see Figure 39). These credentials are left visible on purpose inside the jenkins-slave Dockerfile. After giving the credentials, the changes are saved and the slave is re-launched. Shortly after that the slave is online. Now the user must remember to take the URL from a job called "Setup SUT -Contriboard" and add it into GitLab's contriboard-haproxy repository, using secret variables. This happens simply by left clicking status image and selecting "copy image address".

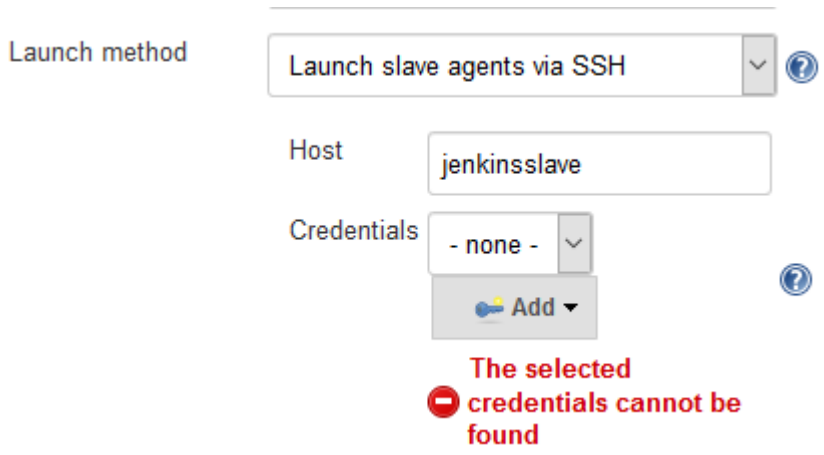


Figure 39. Jenkins slave configuration

Jenkins jobs were pre-configured and delivered via data-container. If the user uses JAMKIT's GitLab repositories, the environment is ready to use.

If user has cloned the repository, he must change the correct path to the new repository and change the folder names to match it in every job!

5.4 Implementation of the environment

GitLab was selected as version control service, so Contriboard's source code was moved there from GitHub. Contriboard and the developed containers source codes can be found at: https://gitlab.com/JAMKIT

The source codes are built to images using a GitLab-CI tool. Every repository contains a file called gitlab-ci.yml, which describes the built steps (see Figure 40). After every commit, GitLab-CI launches GitLab-runner, which executes the image built and pushes it to the GitLab's registry. The idea is the same in every repository, only the image and registry names change.

```
build_image:
  image: docker:latest
  services:
  - docker:dind
  stage: build
  script:
    - docker login -u gitlab-ci-token -p $CI_BUILD_TOKEN registry.gitlab.com
    - docker build -t registry.gitlab.com/jamkit/jarmo .
    - docker push registry.gitlab.com/jamkit/jarmo:latest
```

Figure 40. GitLab's image build steps

Because the development environment is for training, only the contriboard-haproxy repository has a development branch and deploy stage inside gitlab-ci.yml file (see Figure 41). The basic idea is the same as with all other files but after the image is pushed to the registry, GitLab-CI curls Jenkins job URL in order to start the first job. JENKINS_URL is a secret variable for the runner, which is added into repository CI/CD pipelines settings. GitLab offers webhooks, however, the user is not able to determine the exact step when they will trigger. In this case, webhooks are triggered after the commit and the second time after the gitlab-ci.yml stages were finished, which is the reason why they are not in use.

```
stages:
  - build_image
  - deploy

build_image:
  image: docker:git
  stage: build_image
  services:
  - docker:dind
  script:
    - docker login -u gitlab-ci-token -p $CI_BUILD_TOKEN registry.gitlab.com
    - docker build -t registry.gitlab.com/jamkit/contriboard-haproxy .
    - docker push registry.gitlab.com/jamkit/contriboard-haproxy

deploy:
  stage: deploy
  script:
    - curl $JENKINS_URL
```

Figure 41. GitLab's deploy stage

After Jenkins has received the request, Jenkins-slave container deploys the SUT and monitoring stack. The deployment steps are defined in Jenkins execute shell (see Figure 42). Older deployment folders and files are removed if they exist. Next, the repository with docker-compose file for SUT stack deployment is cloned to the Jenkins-slave container. The next step requests an IP address of the server with Contriboard label, using Rancher API. The IP address is parsed from received answer and saved in a text file for later use. After that, the docker-compose file is modified and all words with "SIKA" are replaced with the saved IP address and the SUT stack is launched using Rancher Compose tool. Docker-compose file for SUT deployment can be found at: https://gitlab.com/JAMKIT/contriboard-config

```
#!/bin/bash
set +e

rm -R /var/jenkins_home/contriboard-config || true
rm -R /var/jenkins_home/contriIP.txt || true

git clone https://gitlab.com/JAMKIT/contriboard-config.git /var/jenkins_home/contriboard-config

IP=$(curl -s -u "$RANCHER_API_USER:$RANCHER_API_PASSWORD" -X Get \
    -H 'Accept: application/json' \
    -H 'Content-Type: application/json' \
    "http://$RANCHER_HOST:8080/v2-beta/hosts?description=Contriboard"| grep -Po '"instanceId".{0,35}' | grep -Pom1 '(?<="ipAddress":")[^"]*')

echo $IP
echo $IP > /var/jenkins_home/contriIP.txt
sed -i "s/SIKA/$(cat /var/jenkins_home/contriIP.txt)/" /var/jenkins_home/contriboard-config/docker-compose.yml

rancher-compose --url "http://$RANCHER_HOST:8080/" --access-key $RANCHER_API_USER --secret-key \
$RANCHER_API_PASSWORD --project-name Contriboard \
--file /var/jenkins_home/contriboard-config/docker-compose.yml --verbose up --upgrade --pull -d -c
```

Figure 42. Deploying SUT

After the SUT application stack is launched, Jenkins job sends two HTTP POST requests to Grafana. These requests setup a connection between InfluxDB and pre-configured dashboard (see Figure 43). Appendix 4 presents the deployment chart of this stage.

```
#!/bin/bash

sleep 20

curl 'http://admin:hyva_grafana_passu@'$(cat /var/jenkins_home/contriIP.txt)':3000/api/datasources' \
-X POST -H 'Content-Type: application/json;charset=UTF-8' --data-binary \
'{"name":"jarmo","type":"influxdb","url":"http://'$(cat /var/jenkins_home/contriIP.txt) \
':8086","access":"proxy","isDefault":true,"database":"cb_mon","user":"jarmo", \
"password":"jarmo_monitoring_system_on_paras_ja_influxdb"}'

curl -XPOST -i http://admin:hyva_grafana_passu@$(cat /var/jenkins_home/contriIP.txt):3000/api/dashboards/db \
--data-binary @/var/jenkins_home/contriboard-config/test.json -H "Content-Type: application/json"
```

Figure 43. Grafana setup

If Setup SUT job was successful, Jenkins launches Robot Framework job. Jenkins job executes the following steps: Creates new folder for results if it does not exist, Deploys the Robot Framework container using defined ENV variables (explained in chapter 5.2.3) and saved IP address to the same server, where it is running himself (see Figure 44). Appendix 5 presents the deployment chart of this stage.

```
#!/bin/bash
mkdir -p /var/jenkins_home/RobotResults || true
docker run --rm --name=robot -e GIT_TEST_REPO=https://gitlab.com/jamkit/contriboard-test.git \
-e GIT_REPO_NAME=contriboard-test/robot-framework -e RESOURCE_FILE=resource.txt \
-e TEST_FILES_FOLDER=contriboard-test/robot-framework/ContriboardTesting -e JENKINS_FOLDER=/var/jenkins_home/RobotResults \
-e SUT_IP_ADDRESS=$(cat /var/jenkins_home/contriIP.txt) \
-e TEST_SCRIPT=small_test.sh --net host \
--volumes-from r-"$RANCHER_STACK_NAME"_jenkinsmaster_jenkinsdata_1 registry.gitlab.com/jamkit/robot-framework:latest

# ENV VARIABLES:
# GIT_TEST_REPO
# GIT_REPO_NAME
# RESOURCE_FILE
# TEST_FILES_FOLDER
# JENKINS_FOLDER
# SUT_IP_ADDRESS
# TEST_SCRIPT
```

Figure 44. Deployment of the Robot Framework container

When Robot Framework container has finished the test suite it exits, and the results are analyzed using Jenkins Robot Framework plugin. Users can change the threshold

limits in the job view, however, on default the job is passed even if the job is unstable. The plugin also offers simple trends about earlier build failure rates and durations.

If Robot Framework job was not a failure, Jenkins launches Locust job. Locust job uses the same principles as the Robot Framework job. The biggest difference is that users can change Locust test parameters in the execute shell (see Figure 45). These parameters are: number of requests, number of users and hatch rate. After Locust tests are done, the container exits and Jenkins parses the results in order to figure the failure rate of the test. On default, if the failure rate is under 5%, the job status is a success, otherwise a failure. Users can change this variable easily by editing grep command. Appendix 6 presents deployment chart of this stage.

```
#!/bin/bash
mkdir -p /var/jenkins_home/locustReports || true
docker run --rm --name=Locust -e LOCUST_TARGET_HOST=http://$(cat /var/jenkins_home/contriIP.txt)/ \
-e GIT_TEST_REPO=https://gitlab.com/JAMKIT/Locust-tests.git -e GIT_REPO_NAME=Locust-tests -e LOCUST_LOCUSTFILE=locust-tests-new.py \
-e LOCUST_NUM_REQUEST=50 -e LOCUST_USERS=10 -e LOCUST_HATCH_RATE=1 \
--net host --volumes-from r-"$RANCHER_STACK_NAME"_jenkinsmaster_jenkinsdata_1 registry.gitlab.com/jamkit/locustio:latest

grep Total | awk '$3 > 0 {print $3;}' | grep -Po 'd+\.\d+' | echo $(($1)) | awk '{ if ($1 > 5) exit 1 }'

# ENC variables:
# GIT_TEST_REPO
# GIT_REPO_NAME
# LOCUST_LOCUSTFILE
# LOCUST_TARGET_HOST
# LOCUST_NUM_REQUEST
# LOCUST_USERS
# LOCUST_HATCH_RATE
```

Figure 45. Deployment of the Locust container

Now all functional and performance test are done and Jenkins launches the job which removes SUT. The job uses Rancher API to get the correct id of the SUT stack. When the correct id is received, the job sends DELETE request with the correct id to Rancher API and SUT is removed (see Figure 46). After the job is done, Jenkins deploys Contriboard and monitoring stack by using the first job again.

```
ID=$(curl -s -u "$RANCHER_API_USER:$RANCHER_API_PASSWORD" -X Get \
    -H 'Accept: application/json' \
    -H 'Content-Type: application/json' \
    "http://$RANCHER_HOST:8080/v2-beta/projects/1a5/stacks?name=contriboard" | grep -Po '"id".{0,10}' | grep -Pom1 '(?<="id":")[^"]*')

echo $ID
echo removing SUT

curl -u "$RANCHER_API_USER:$RANCHER_API_PASSWORD" \
-X DELETE \
"http://$RANCHER_HOST:8080/v2-beta/projects/1a5/stacks/$ID"
```

Figure 46. SUT removal

6 CONCLUSION

All requirements were fulfilled and the development environment is fully functional. It can be used with free services and it prefers software that is studied during the courses at school, so it can be used for teaching in the future. For that reason, it does not offer a shortest possible pipeline or the best tools, however, the development environment tools can be replaced rather easily.

The development environment is done for training usage so it is not recommended for production. Even though it is fully functional, it is not perfect and could be better implemented. For example, there are few credentials visible in the source code and some containers do not have a defined user, which are security threats. In further development, these problems are rather easy to fix by using GitLab secret variables and simply adding users in Dockerfiles. Other future improvements are: Usage of dynamic IP addresses and servers, Changing Jarmo's data format from JSON to something else.

Implementation of the development environment took about 400 hours. The development workflow can be divided roughly into the three phases: research, implementation of the container and integration of the environment. Figure 47 presents the time allocation between the development phases.

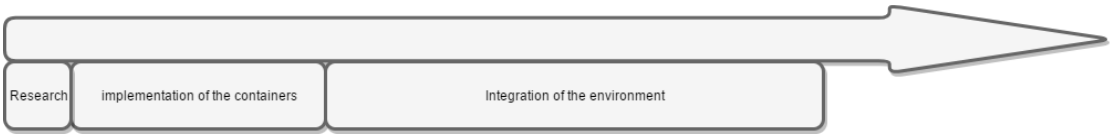


Figure 47. Time allocation between the development phases

The lecturers have noticed that students require multiple hours of time to set up the needed test benches and environments before starting the actual work. Because of that, both test bench containers are also modified to standalone containers, which can be used without the development environment. This way the students do not

need to worry about tools and environments and they can focus on the test cases. Standalone test benches are available at JAMKIT GitLab: https://gitlab.com/JAMKIT

7 ACKNOWLEDGEMENT

This work was supported by TEKES (the Finnish Funding Agency for Innovation) as a part of the DIMECC's 'Need for Speed' program (http://www.n4s.fi/en/). This work has been done in co-operation with research project (N4S@JAMK) at JAMK University of Applied Sciences.

References

A Short History of Agile. N.d. The Agile Alliance article about Agile. Accessed on 19 May 2017. Retrieved from https://www.agilealliance.org/agile101/

About AWS. N.d. Wiki page of the Amazon Web Services. Accessed on 4 March 2017. Retrieved from https://aws.amazon.com/about-aws/

About technologies. 2014. Contriboard technologies on N4sJAMK GitHub page. Accessed on 5 March 2017. Retrieved from https://github.com/N4SJAMK/teamboard-meta/wiki/about-technologies

Agile. N.d. Image from tutorialspoint webpage. Accessed on 20 May 2017. Retrieved from https://www.tutorialspoint.com/agile/agile_primer.htm

Azure IaaS. N.d. Image from Sysfore Technologies webpage. Accessed on 20 May 2017. Retrieved from http://www.sysfore.com/azure-iaas.aspx

Benefits of Agile Software Development. 2015. SEGUE Technologies article about agile benefits. Accessed on 20 May 2017. Retrieved from http://www.seguetech.com/8-benefits-of-agile-software-development/

Black, R. Evans D. Graham, D & van Veenendaal, E. 2008. Foundations of Software Testing, ISTQB Certification, Revised Edition. London: Cengage Learning EMEA

Bond, J. 2015. The Enterprise Cloud. California: O'Reilly Media

Branching and Merging. N.d. Git documentation. Accessed on 14 April 2017. Retrieved from https://git-scm.com/about/branching-and-merging

cgroup-v1.txt. N.d. Linux kernel documentation on Linux git. Accessed on 13 April 2017. Retrieved from https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/diff/Documentation/cgroup-v1?id=v4.5&id2=v4.4

Clarke, G. 2016. The Evolution of Software Development Practices. Accessed on 18 May 2017. Retrieved from http://garyclarke.us/technology/the-evolution-of-software-development-practices/

Cloud Service Models. N.d. IBM article about cloud service models. Accessed on 17 May 2017. Retrieved from https://www.ibm.com/cloud-computing/learn-more/iaas-paas-saas/

Company Overview of Digital Ocean. 2017. DigitalOcean company overview in Bloomberg's stock page. Accessed on 9 May 2017. Retrieved from https://www.bloomberg.com/research/stocks/private/snap-shot.asp?privcapid=243910980

Compute. N.d. DigitalOcean article about its products. Accessed on 6 May 2017. Retrieved from https://www.digitalocean.com/products/compute/

Continuous delivery. N.d. Amazon web services article about continuous delivery. Accessed on 10 April 2017. Retrieved from https://aws.amazon.com/devops/continuous-delivery/

Continuous integration. N.d. Amazon web services article about continuous integration. Accessed on 10 April 2017. Retrieved from https://aws.amazon.com/devops/continuous-integration/

Contriboard. N.d. Image from Contriboard web page. Accessed on 5 March 2017. Retrieved from http://n4sjamk.github.io/contriboard/

Cooper production environment. 2017. Documentation about Cooper production environment in JAMK-IT GitHub pages. Accessed in 30 May 2017. Retrieved from https://github.com/JAMK-IT/DOC10-example-project/wiki/Cooper-product-line

Daniels, K. & Davis J. 2016. Effective DevOps. California: O'Reilly Media

Deep customer insight. N.d. N4S work package introduction page. Accessed on 5 March 2017. Retrieved from http://www.n4s.fi/en/work-packages/deep-customer-insight/

DigitalOcean becomes the second largest hosting company in the world. 2015. Netcraft article about DigitalOcean. Accessed on 9 May 2017. Retrieved from https://news.netcraft.com/archives/2015/05/01/digitalocean-becomes-the-second-largest-hosting-company-in-the-world.html

Distributed builds. 2016. Jenkins documentation about distributed builds. Accessed on 18 April 2018. Retrieved from https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds

Docker architecture. N.d. Docker documentation. Accessed in 31 May 2017. Retrieved from https://docs.docker.com/engine/docker-overview/#docker-architecture

EC2. N.d. EC2 page of the Amazon Web Services. Accessed on 4 March 2017. Retrieved from https://aws.amazon.com/ec2/?nc2=h_m1

Ferguson Smart, J. 2011. Jenkins: The Definitive Guide. California: O'Reilly Media

Fowler, M & Lewis, J. 2014. Article about microservices. Accessed on 5 May 2017. Accessed on https://martinfowler.com/articles/microservices.html

Fowler, M. 2013. Continuous Delivery. Accessed on 18 November 2014. Retrieved from https://martinfowler.com/bliki/ContinuousDelivery.html

Functional testing. N.d. Article about functional testing on Guru99 website. Accessed on 24 May 2017. Retrieved from http://www.guru99.com/functional-testing.html

Getting Started - Git Basics. N.d. Git documentation. Accessed on 14 April 2017. Retrieved from https://git-scm.com/book/en/v1/Getting-Started-Git-Basics

Ghahrai, A. 2015. Software Development Life Cycle - SDLC Phases. Accessed on 18 May 2017. Retrieved from http://www.testingexcellence.com/software-development-life-cycle-sdlc-phases/

GitLab Continuous Integration. N.d. GitLab-CI document in GitLab webpage. Accessed on 6 May 2017. Retrieved from https://docs.gitlab.com/ee/ci/

Grafana Documentation. N.d. Grafana documentation on their webpage. Accessed in 28 May 2017. Retrieved from http://docs.grafana.org/

Guckenheimer, S. N.d. Article about DevOps in Visual Studio webpage. Accessed on 22 May 2017. Retrieved from https://www.visualstudio.com/learn/what-is-devops/

Hartikainen, T. 2015. Jarmo, simple StatsD -like server. Accessed in 28 May 2017. Retrieved from https://github.com/N4SJAMK/jarmo

Hildred, T. 2015. The History of Containers article in RedHat's webpage. Accessed on 10 April 2017. Retrieved from http://rhelblog.redhat.com/2015/08/28/the-history-of-containers/

History of the product. 2014. Teamboard History on N4SJAMK GitHub page. Accessed on 5 March 2017. Retrieved from https://github.com/N4SJAMK/teamboard-meta/wiki/about-product

Images and layers. N.d. Docker documentation about images. Accessed on 26 May 2017. Retrieved from https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/

Keyword-driven-testing. N.d. Article about Keyword-driven testing on Guru99 webpage. Accessed on 14 May 2017. Retrieved from http://www.guru99.com/keyword-driven-testing.html

Lewis, W. 2008. Software Testing and Continuous Quality Improvement, Third Edition. Florida: Auerbach Publications

Lianping, C. 2015. Toward Architecting for Continuous Delivery. Accessed on 13 April 2017. Retrieved from http://ieeexplore.ieee.org/document/7158514/

Manifesto for Agile Software Development. 2001. Agile for Agile Software Development. Accessed on 20 May 2017. Retrieved from http://agilemanifesto.org/

Marschall, M. 2012. Lean, Agile and DevOps relations. Accessed on 22 May 2017. Retrieved from http://www.agileweboperations.com/lean-agile-devops-related

Mathias, K. & Kane, S. 2015. Docker: Up and Running. 2015. California: O'Reilly Media

Mercury Business. N.d. N4S work package introduction page. Accessed on 5 March 2017. Retrieved from http://www.n4s.fi/en/work-packages/mercury-business/

Need4Speed brochure. N.d. Broschure about N4S project. Accessed in 5 March 2017. Retrieved from http://www.n4s.fi/wordpress/wp-content/uploads/2014/04/Need4Speed_esite_web.pdf

Newman, S. 2015. Building Microservices. California: O'Reilly Media

NIST Special Publication 800-145. 2012. United States National Institute of Standards and Technology definition of cloud computing. Accessed on 16 May 2017. Retrieved from http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

Overview of Rancher. N.d. Overview of Rancher in Rancher documentation. Accessed on 13 May 2017. Retrieved from http://docs.rancher.com/rancher/v1.6/en/

Paradigm change – delivering value in real time. N.d. N4S work package introduction page. Accessed on 5 March 2017. Retrieved from http://www.n4s.fi/en/work-packages/paradigm-change-delivering-value-in-real-time/

Parent image. N.d. Figure from Docker's old documentation. Accessed in 29 May 2017. Retrieved from http://docs.master.dockerproject.org/terms/image/

Products. N.d. GitLab's product page. Accessed 6 May 2016. Retrieved from https://about.gitlab.com/products/

Rafaels, R. 2015. Cloud Computing: From Beginning to End. Amazon CreateSpace Publishing.

Rancher Catalog. N.d. Catalog article in Rancher documentation. Accessed on 13 May 2017. Retrieved from http://docs.rancher.com/rancher/v1.6/en/catalog/

Rancher Compose. N.d. Rancher Compose manual in Rancher documents. Accessed on 14 May 2017. Retrieved from http://docs.rancher.com/rancher/v1.6/en/cattle/rancher-compose/

Reddy, J. M. & Prasad, S. V. A. V. 2016. The Role of Verification and Validation in Software Testing. Accessed on 3 April 2017. Retrieved from http://ieeexplore.ieee.org/document/7724475/

RightScale 2017 - State of the Cloud Report. 2017. Rightscale report. Accessed on 17 May 2017. Retrieved from http://assets.rightscale.com/uploads/pdfs/RightScale-2017-State-of-the-Cloud-Report.pdf

Robot Framework – Test automation the smart way. N.d. Article about Robot Framework. Accessed on 14 May 2017. Retrieved from http://quintagroup.com/cms/python/robot-framework

Robot Framework Quick Start Guide. 2017. Robot Framework's GitHub documentation. Accessed in 14 May 2017. Retrieved from https://github.com/robotframework/QuickStartGuide/blob/master/QuickStart.rst

Robot Framework. N.d. Robot Framework homepage. Accessed on 14 May 2017. Retrieved from http://robotframework.org/#documentation

Software Testing. N.d. Tutorialspoint article about software testing. Accessed on 22 May 2017. Retrieved from https://www.tutorialspoint.com/software_testing/software_testing_overview.htm

Structural testing. N.d. Article about functional testing on Guru99 website. Accessed on 24 May 2017. Retrieved from http://www.guru99.com/structural-testing.html

The Treacherous 12. 2016. Cloud Security Alliance report about cloud computing treats. Accessed on 17 May 2017. Retrieved from https://downloads.cloudsecurityalliance.org/assets/research/top-threats/Treacherous-12_Cloud-Computing_Top-Threats.pdf

Treasure Chest – How to make software business in real-time. N.d. N4S work package introduction. Accessed on 4 April 2017. Retrieved from http://www.n4s.fi/en/work-packages/

Types of System Test. 2017. Article about System testing in SoftwareTestingHelp website. Accessed on 21 April 2017. Retrieved from http://www.softwaretestinghelp.com/system-testing/

Types of Testing. N.d. Tutorialpoint article about testing types. Accessed on 22 May 2017. Retrieved from https://www.tutorialspoint.com/software_testing/software_testing_types.htm

Understand microservices. 2016. Microsoft Azure article about microservices. Accessed on 5 May 2017. Retrieved from https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview-microservices

Unit Testing. N.d. Software Testing Fundamentals article. Accessed on 4 April 2017. Retrieved from http://softwaretestingfundamentals.com/unit-testing/

V-model. N.d. Tutorialspoint article about V-model. Accessed on 10 April 2017. Retrieved from https://www.tutorialspoint.com/software_testing_dictionary/v_model.htm

What are Software Development Models. N.d. ISTQB article about Development models. Accessed on 18 May 2017. Retrieved from http://istqbexamcertification.com/what-are-the-software-development-models/

What are Software Testing Levels. N.d. ISTQB article about software testing levels. Accessed on 22 May 2017. Retrieved from http://istqbexamcertification.com/what-are-software-testing-levels/

What are the SDLC phases. N.d. ISTQB article about SDLC phases. Accessed on 18 May 2017. Retrieved from http://istqbexamcertification.com/what-are-the-software-development-life-cycle-sdlc-phases/

What is a Container. N.d. Docker article about containers. Accessed on 13 April 2017. Retrieved from https://www.docker.com/what-container

What is Agile Software Development. N.d. Article about Agile in VersionOne webpage. Accessed on 20 May 2017. Retrieved from https://www.versionone.com/agile-101/

What is Agile? 2013. AgileInSights blog post about agile. Accessed on 20 May 2017. Retrieved from https://agileinsights.wordpress.com/tag/agile-development/

What is DevOps. N.d. Article about DevOps in Amazon webpage. Accessed on 21 May 2017. Retrieved from https://aws.amazon.com/devops/what-is-devops/

What is Dynamic testing. N.d. Software Testing Class article about dynamic testing. Accessed on 24 May 2017. Retrieved from http://www.softwaretestingclass.com/difference-between-static-testing-and-dynamic-testing/

What is Locust. 2015. Locust.io documentation. Accessed on 18 May 2017. Retrieved from http://docs.locust.io/en/latest/what-is-locust.html

What is regression testing in software. N.d. ISTQB exam certification article about regression testing. Accessed on 7 April 2017. Retrieved from http://istqbexamcertification.com/what-is-regression-testing-in-software/

What is System Testing N.d. Software Testing Class article about System testing. Accessed on 5 April 2017. Retrieved from http://www.softwaretestingclass.com/system-testing-what-why-how/

What is User Acceptance Testing. N.d. Software Testing Class article about user acceptance testing. Accessed in 23 May 2017. Retrieved from http://www.softwaretestingclass.com/user-acceptance-testing-what-why-how/

What is Waterfall Model. N.d. ISTQB article about Waterfall model. Accessed on 18 May 2017. Retrieved from http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/

What is version control. N.d. Tutorial about version control in Atlassian webpage. Accessed on 14 April 2017. Retrieved from https://www.atlassian.com/git/tutorials/what-is-version-control

What is V-model. N.d. Software Testing Help's article about V-model. Accessed on 19 May 2017. Retrieved from http://www.softwaretestinghelp.com/what-is-stlc-v-model/

Why move to the cloud. 2015. Blog post about cloud benefits in Salesforce page. Accessed on 17 May 2017. Retrieved from https://www.salesforce.com/uk/blog/2015/11/why-move-to-the-cloud-10-benefits-of-cloud-computing.html

Why Operational Testing. N.d. Guru99 article about operational testing. Accessed on 25 May 2016. Retrieved from http://www.guru99.com/operational-testing.html

V-model advantages and disadvantages. N.d. ISTQB article about V-model. Accessed on 19 May 2017. Retrieved from http://istqbexamcertification.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/

Vserver 0.0 changes log. N.d. Vserver first changes log. Accessed on 12 April 2017. Retrieved from http://www.solucorp.qc.ca/changes.hc?projet=vserver&version=0.0

Appendices

Appendix 1. Jenkins stack docker-compose.yml file

```
jenkinsmaster:
  labels:
    io.rancher.sidekicks: jenkinsdata,jenkinsslave
    io.rancher.scheduler.affinity:host_label: jenkins=true
  image: registry.gitlab.com/jamkit/jenkins-master-2.3:latest
  links:
    - jenkinsslave:jenkinsslave
  volumes_from:
  - jenkinsdata
jenkinsdata:
  labels:
    io.rancher.container.start_once: 'true'
    io.rancher.scheduler.affinity:host_label: jenkins=true
  image: registry.gitlab.com/jamkit/jenkins-data-2.3:latest
jenkinsslave:
  labels:
    io.rancher.scheduler.affinity:host_label: jenkins=true
  environment:
      - RANCHER_HOST=0.0.0.0
      - RANCHER_API_USER=xxxxx
      - RANCHER_API_PASSWORD=xxxxx
      - RANCHER_STACK_NAME=xxxxxx
  image: registry.gitlab.com/jamkit/jenkins-slave-2.3:latest
  volumes:
  - /var/run/docker.sock:/var/run/docker.sock
  volumes_from:
  - jenkinsdata
jenkinsnginx:
  ports:
  - "8088:8088"
  labels:
    io.rancher.scheduler.affinity:host_label: jenkins=true
  image: registry.gitlab.com/jamkit/jenkins-nginx:latest
  links:
  - jenkinsmaster:jenkins-master
```

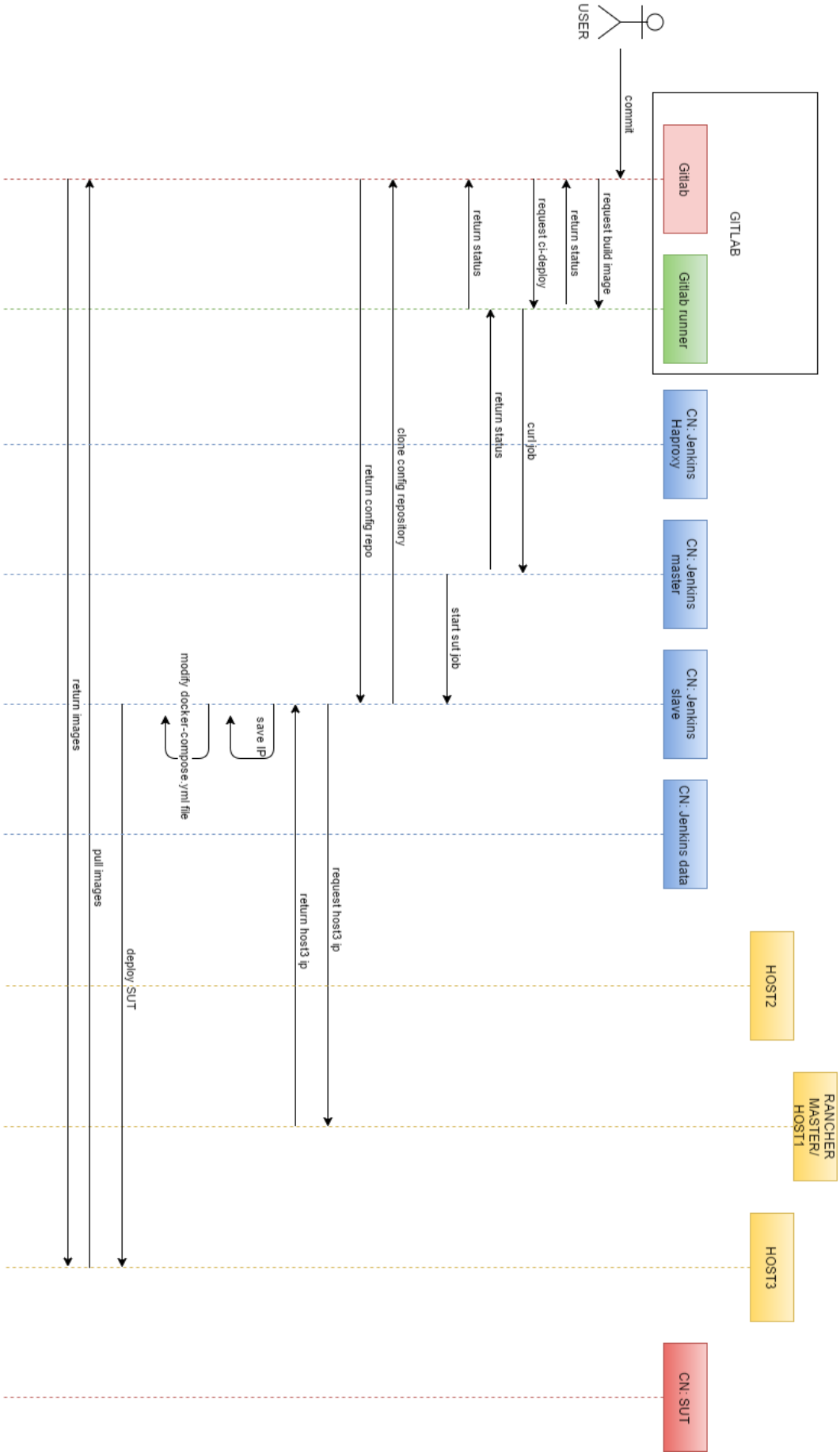
Appendix 2. Dockerfile of the Jenkins-slave container

```
FROM ubuntu:14.04
MAINTAINER JAMKIT

# Preparations for sshd
run locale-gen en_US.UTF-8 &&\
    apt-get -q update &&\
    DEBIAN_FRONTEND="noninteractive" apt-get -q upgrade -y -o Dpkg::Options::="--force-confnew" --no-install-recommends &&\
    DEBIAN_FRONTEND="noninteractive" apt-get -q install -y -o Dpkg::Options::="--force-confnew"  --no-install-recommends openssh-server &&\
    apt-get -q autoremove &&\
    apt-get -q clean -y && rm -rf /var/lib/apt/lists/* && rm -f /var/cache/apt/*.bin &&\
    sed -i 's|session    required     pam_loginuid.so|session    optional     pam_loginuid.so|g' /etc/pam.d/sshd &&\
    mkdir -p /var/run/sshd

ENV LANG en_US.UTF-8
ENV LANGUAGE en_US:en
ENV LC_ALL en_US.UTF-8
ENV RANCHER_COMPOSE_VERSION v0.8.6

RUN apt-get update -qq && apt-get install -qqy \
    apt-transport-https \
    ca-certificates \
    curl \
    lxc \
    wget \
    iptables

RUN wget -qO- https://github.com/rancher/rancher-compose/releases/download/${RANCHER_COMPOSE_VERSION}/rancher-compose-linux-amd64-${RANCHER_COMPOSE_VERSION}.tar.gz | tar xvz -C /tmp
RUN mv /tmp/rancher-compose-${RANCHER_COMPOSE_VERSION}/rancher-compose /usr/local/bin/rancher-compose
RUN chmod +x /usr/local/bin/rancher-compose

RUN apt-get install -y software-properties-common
RUN add-apt-repository -y ppa:webupd8team/java
RUN apt-get update
RUN yes 'yes' | apt-get install -y oracle-java8-installer

# Install Docker from Docker Inc. repositories.
RUN curl -sSL https://get.docker.com/ | sh

# Set user jenkins to the image
RUN useradd -m -d /home/jenkins -s /bin/sh jenkins &&\
    echo "jenkins:jenkins" | chpasswd
#-ou 0 -g 0
RUN sudo adduser jenkins sudo
RUN usermod -aG docker jenkins
RUN echo 'jenkins   ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers

RUN mkdir -p /var/jenkins_home/RobotResults
RUN mkdir -p /var/jenkins_home/locustReports
RUN chmod +777 /tmp


EXPOSE 22

RUN apt-get -yqq autoremove && \
                apt-get -yqq clean && \
                rm -rf /var/lib/apt/lists/* /var/cache/* /tmp/* /var/tmp/*
# Default command
CMD env | grep _ >> /etc/environment && /usr/sbin/sshd -D
```

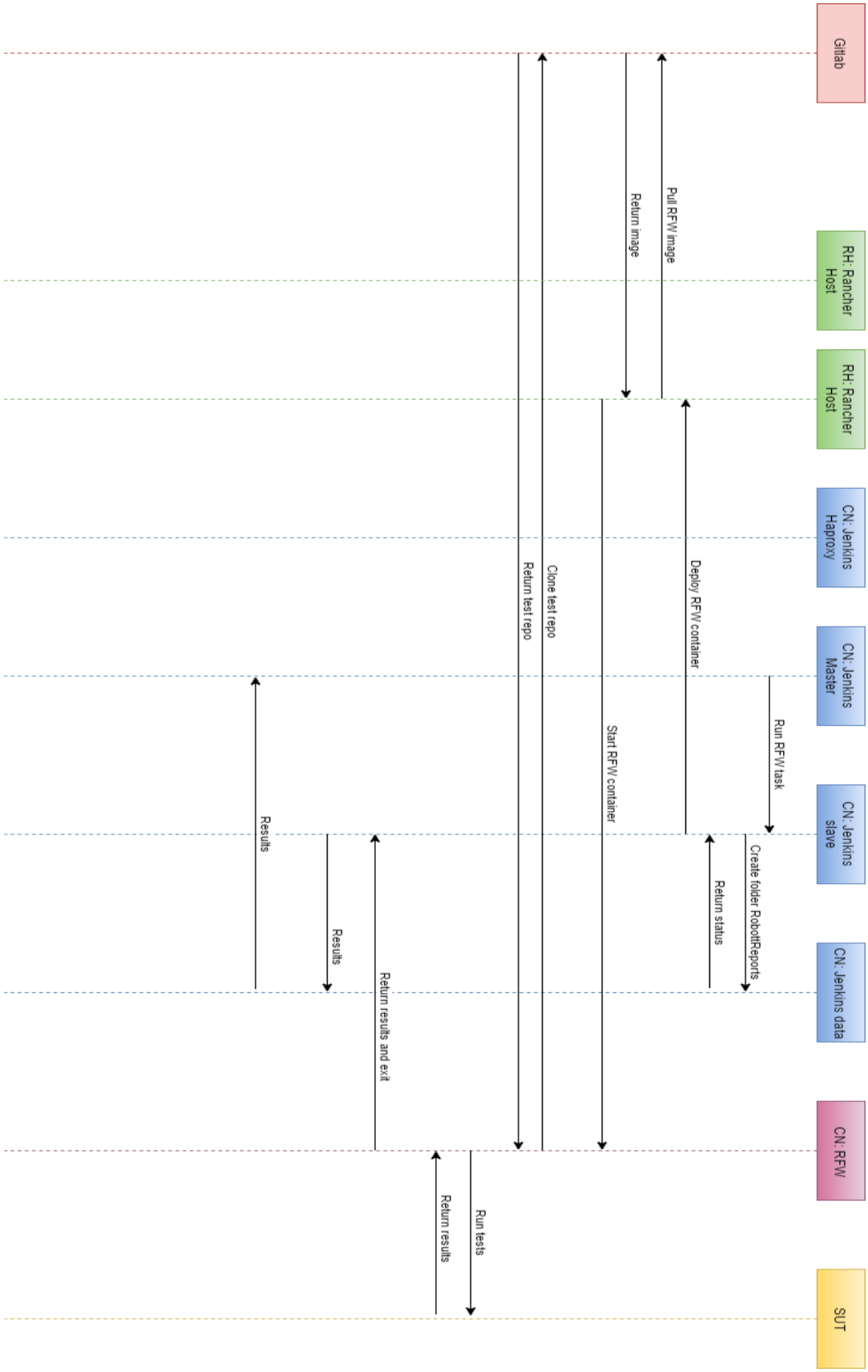
Appendix 3. Dockerfile of the Robot Framework container

```
FROM ubuntu:16.04

# Add Google Chrome repo
RUN apt-get update -qq && \
apt-get install -y wget

RUN echo "deb http://dl.google.com/linux/chrome/deb/ stable main" | tee /etc/apt/sources.list.d/google-chrome.list && \
wget -q -O - https://dl.google.com/linux/linux_signing_key.pub | apt-key add -

RUN \
apt-get update && \
apt-get install -y \
python-pip \
unzip \
git \
curl \
firefox \
ca-certificates \
google-chrome-stable \
xfonts-100dpi \
xfonts-75dpi \
xfonts-scalable \
xfonts-cyrillic \
xfonts-base \
xvfb \
imagemagick \
python-setuptools

# Add chromedriver
RUN wget https://chromedriver.storage.googleapis.com/2.27/chromedriver_linux64.zip && \
unzip chromedriver_linux64.zip && rm -f chromedriver_linux64.zip && \
chmod +777 chromedriver && mv -f chromedriver /usr/bin/chromedriver && \
apt-get remove -y unzip

# Add geckodriver
RUN wget https://github.com/mozilla/geckodriver/releases/download/v0.14.0/geckodriver-v0.14.0-linux64.tar.gz && \
tar xfvz geckodriver-v0.14.0-linux64.tar.gz && rm -f geckodriver-v0.14.0-linux64.tar.gz && \
chmod +777 geckodriver && mv -f geckodriver /usr/bin/geckodriver

#Install robotframework
RUN pip install -U requests urllib3 robotframework robotframework-selenium2library robotframework-xvfb robotframework-requests docutils xvfbwrapper && \
apt-get autoremove -y && \
rm -rf /var/lib/apt/lists/*

#ADD own library
ADD ExtendedSelenium2Library.py /usr/local/lib/python2.7/dist-packages/

# Change chrome launch parameter adn add starting script
COPY google-chrome-launcher /usr/bin/google-chrome

COPY start.sh /usr/local/bin/start.sh
RUN chmod -R 755 /usr/local/bin/start.sh
ENTRYPOINT ["/usr/local/bin/start.sh"]
```

Appendix 4. SUT deployment chart



Appendix 5. Robot Framework deployment chart



Appendix 6. Locust deployment chart