Tuomas Vase

ADVANTAGES OF DOCKER

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2015

ABSTRACT

Vase, Tuomas
Advantages of Docker
Jyväskylä: University of Jyväskylä, 2015, 24 p.
Information Systems Science, Bachelor's Thesis
Supervisor: Seppänen, Ville

Docker is an open platform product that can package an application and its dependencies inside a virtual container and this technology is called as container technology. Those packages can be build, ship and run inside distributed environments by developers or system administrators with ease. This technology offers huge advantages for enterprises with new kind of portability, scalability, speed, delivery and maintenance.

This thesis focused on the advantage side of Docker container technology. Differences between virtualization and container technology were examined from several levels which included usability, benefits, disadvantages, risks and performance. The aim of the study was to investigate has Docker all those advantages it is hyped for and is there negative sides in this technology.
The most important observation of this study is that Docker has many potential advantages in the fields of usability, performance and security against traditional virtualization. The research was executed as a literature review with some own conclusions.

Keywords: docker, virtualization, container, container-technology, performance, advantages, security

TIIVISTELMÄ

Vase, Tuomas
Advantages of Docker
Jyväskylä: Jyväskylän yliopisto, 2015, 24 s.
Tietojärjestelmätiede, kandidaatintutkielma
Ohjaaja: Seppänen, Ville

Docker on avoimen alustan sovellus, joka pystyy pakkaamaan sovelluksen kaikkien tarvittavien riippuvuuksien kanssa yhteen virtuaaliseen pakettiin, eli konttiin, ja tätä teknologiaa kutsutaan konttiteknologiaksi. Näitä kontteja ohjelmistokehittäjät ja järjestelmäasiantuntijat voivat rakentaa, lähettää ja ajaa helposti hajauttettujen ympäristöjen sisällä. Tämä teknologia antaa valtavia etuja yrityksille uudenlaisen siirrettävyyden, skaalautuvuuden, nopeuden jakamisen ja ylläpidon muodossa.

Tämä tutkielma keskittyi Docker konttiteknologian etuihin. Nykyisen virtualisoinnin ja konttiteknologian eroja tutkittiin monella tasolla, jotka sisälsivät käytettävyyden, hyötyjen, haittojen, riskien ja suorituskyvyn näkökulmia. Tutkimuksen tarkoituksena oli selvittää onko Dockerilla kaikki ne hyvät ominaisuudet, joista sitä on kehuttu ja mitä huonoja puolia tästä teknologiassa löytyy. Tärkein löytö tässä tutkimksessa oli, että Dockerilla on monia potentiaalisia hyötyjä käytettävyyden, tehokkuuden ja turvallisuuden saralla verrattaen tavanomaiseen virtualisointiin. Tutkimus tehtiin kirjallisuuskatsauksena, jossa on mukana myös omia päätelmiä.

Asiasanat: docker, virtualisointi, kontti, pilvilaskenta, suorituskyky, hyödyt, turvallisuus

FIGURES

TABLES

TABLE OF CONTENTS

1 Introduction

The use of virtualization technologies has increased drastically over the last few years, as cloud computing has evolved and the needs of users and enterprises have exceedingly increased. All of these changes make a huge demand for highly performing and efficient solutions for virtualization, thus new methods of virtualization have born for the needs of users and enterprises. (Bui, 2014).

Virtualization is a term that refers to the abstraction of computer resources. Virtualization's purpose is to improve resource utilization by providing a unified and integratable platform for users and applications. (Luo, Lin, Chen, Yang, & Chen, 2011). New kind of virtualization is called container-technology which is a tool for delivering software. In other words containers have a platform-as-a-service (PaaS) or software-as-a-service (SaaS) focus in a portable way. This provides a greater interoperability while still utilizing operating system (OS) virtualization principles (Pahl, 2015). Although containers and virtual machines are both virtualization techniques, they fix different problems (Pahl, 2015). Although virtualization actually happened as early as the 1960s with virtual machines in IBM System/360 machines, it was not until 2001 when VMWare introduced its x86 virtualization software, which skyrocketed virtualization of Linux environments. (Fink, 2014).

In this thesis the focus will be in Docker[1] containers, as it is the most common container technology and also most of the published scientific materials are related to it. Docker is an open source project that offers a consistent way to automate the faster deployment of applications inside portable containers (Bernstein, 2014). With containers, applications share an operating system and whenever possible, also binaries and libraries. The result is that these deployments will be drastically smaller in size than traditional virtualizations deployments, making it possible to store hundreds of containers on a physical host (Bernstein, 2014).

[1] http://www.docker.com

The motivation to make this study were that software industry and cloud providers are rapidly changing to container environments and thus how the container-based virtualization is even working and what are the benefits of this kind of virtualization is an interesting subject. Aim of the study was to find answer that has Docker all those advantages that it is hyped for and is it simple to execute and use. This thesis had some challenges in terms of founding relevant enough scientific material, as Docker containers is only two years old technology and thus thorough examinations of this technology does not exist yet.

This study revealed that Docker has some serious advantages in fields of performance, usability and security when reviewed against traditional virtualization.

1.1 Research problem and research questions

In the deepest core Docker is a virtualization framework focused around running applications and not for emulating hardware, which seems simple at first but underlines the critical difference between operating system level virtualization software like Docker and machine level virtualization (Fink, 2014). However, using Docker is not yet an easy procedure, as Docker wants to run things in foreground what necessitates a need for conversion of common programs (Fink, 2014). Docker's focus on one application per container might also be problematic, although there are a lot of advantages. As Dua, Raja and Kakadia (2014) state in their study, that containers have a congenital advantage over virtual machines because of improvements in performance and reduced startup time. Docker is a lightweight solution that launch in a sub-second with hypervisor availability on top of the operating system, which allows quite a lot of scalability (Anderson, 2015). Regarding to these statements, this thesis will first aim at answering the following research question:

- What are the advantages for enterprises in container technology?

Critical infrastructure is controlled more increasingly by software (Bradley, Fehnker, & Huuck, 2011). Securing the chain of software production is therefore an increasing concern. All stages of software deployment have chance to be corrupted during the software deployment pipeline and other vulnerabilities may occur when integrating the software with other infrastructures (Bass, Holz, Rimba, Tran, & Zhu, 2015). Attackers are increasingly trying to figure out new ways to exploit any shown weaknesses and other vulnerabilities in software systems in order to gain financially benefits or only for doing harm (Axelrod, 2014). However, there might be a solution for these security concerns in the field of software development and container technology. Therefore, second research question is:

- What are the security risks and safety benefits of container technology?

1.2 Research method and data acquisition

This research was made in a form of literature review. Literature is mostly acquired from Google Scholar and the Institute of Electrical and Electronics Engineers (IEEE) databases. Relevance of the used references was weighted by the score of referred amounts and latest articles were considered more valuable than older articles. Thesis is mostly using articles which are published in the year 2014 or 2015. References were searched by relevant keywords and combinations of such keywords. These keyword combinations were executed from the following words: docker, virtualization, container, container-technology, performance, advantages and security.

1.3 Research structure

This research is divided to four main chapters. First chapter is Introduction, where main points of virtualization and container technology are briefly demonstrated, followed by research problem, questions, method, and structure with data acquisition.

Next chapter, Containers and virtualization, will explain main concepts of virtualization and how containers belong to it. Definitions of used terminology and other available containers are also explained in this chapter. Docker containers architecture and comparison to other containers will follow after mentioned definitions.

Third chapter, Advantages of Docker as container technology, is the main chapter of this thesis. This chapter is divided to three main points, which are Dockers usability, performance comparison and security advantages. Each point of view has pros and cons towards Docker.

Last chapter is Conclusion, where main advantage points are repeated and the core content of this thesis is presented. In addition, future studies in container technology are considered.

2 Containers and virtualization

The popularity of Cloud computing due increasing amount of customers has led providers to use resource-sharing solutions to meet the needs of infrastructures resources (Xavier et al., 2015). Virtual machines have been the backbone for cloud computing at the infrastructure layer, as virtual machines are providing virtualized operating systems. Containers are a similar, but lighter solution for virtualization, as containers use a far less resources and time as traditional virtualization (Figure 1) technologies (Pahl, 2015).

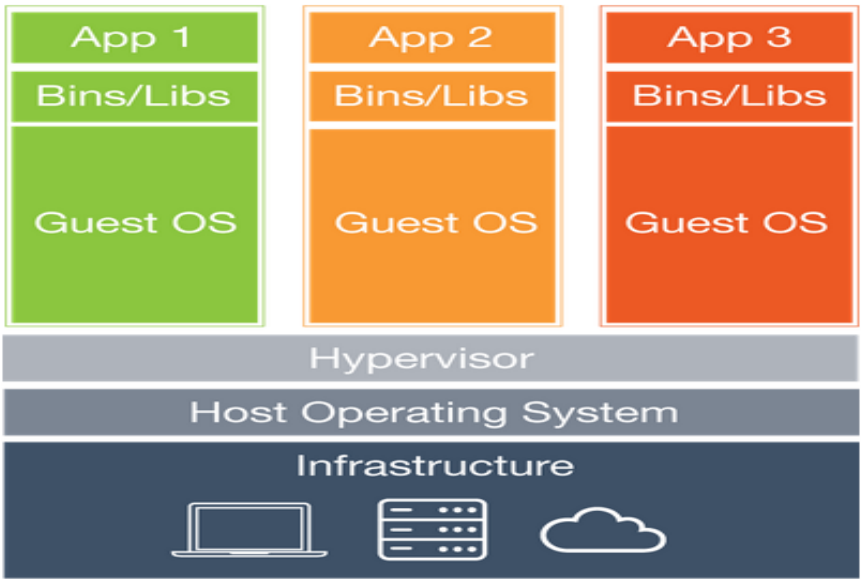

Figure 1 - Traditional virtualization[2]

In traditional virtualization, hypervisor is controlling host machines resources. Hypervisor-based virtualization can share, for example, memory resources across memory limits, as most processes (or virtual machines in this case) do

[2] https://www.docker.com/what-docker

not consume all of their allocated memory. Therefore, users can have more computing resources than is physically available in cloud based environments. Operation level virtualization, like container-based virtualization, can even do better, as it has improved resource sharing. Container technology grants multiple isolated instances with wanted properties for user and eases the managing and generating processes. Thus, container-based virtualization is ahead of traditional virtualization in terms of usability, but it also extends the resource utilization and therefore reduces the overhead of creating new virtualization processes. (Adufu, Jieun, & Yoonhee, 2015.)

Almost every virtualization technology can be placed in two main categories of virtualization: hypervisor-based virtualization and container-based virtualization. Containers provide operating system level virtualization, as Hypervisor-based virtualization is in the hardware level. In other words, container-based virtualization (Figure 2) is running multiple virtual environments on top of host kernel, whereas hypervisor is simulating computers resources as a whole. (Bui, 2014). Mostly each of these virtualization environments, which are running in container-based virtualization, is referred to containers. Containers are isolated from each other and other processes and they are also sharing only those resources, which are dictated to them. (Boettiger, 2015).

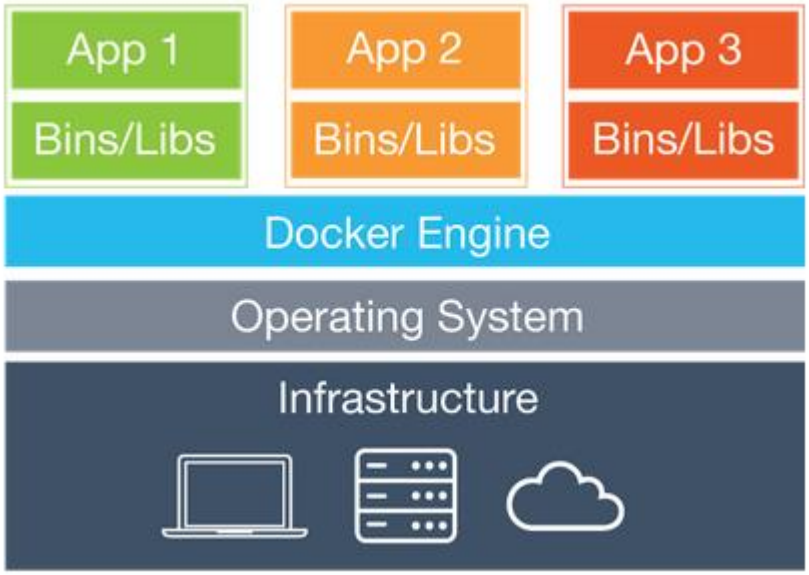

Figure 2 - Docker containers[3]

Operating system (OS) containers run multiple processes, whereas application containers run a single service that is packaged. As Linux Containers (LXC) is a pure OS container technology, Docker and Rocket[4] (rkt) are examples of application level containerization. However, application level containers also need

[3] https://www.docker.com/what-docker
[4] https://github.com/coreos/rkt

OS level container under it. Although all of them share kernel and same kind of resources, they are delicately different. In application container approach containers are created for each of the components of designed application and this approach suite extremely well with multi-component systems, which are distributed. (Karle, 2015.)

2.1 Containerization, Docker and PaaS

DotCloud, which is a platform-as-a-service vendor, firstly created an open source application containerization project that we nowadays know as Docker. Containerization project has reached enormous popularity, as many cloud computing giants, like Amazon Web Services (AWS), have announced that they are using and supporting applications, which are containerized by Docker. (Linthicum, 2014).

Containerization term is obtained from shipping containers as it has same kind of principle: to ship or store all kinds of "cargo". This kind of containerization provides a generic way to isolate processes from each other and rest of the system. (Dua et al., 2014)

Developer productivity has drastically increased with PaaS approach, as deployment of services can be done rapidly and easily. Use of containers with PaaS is nowadays popular as major cloud providers are already using containers to provide their services. (Dua et al., 2014). However, there are still few issues that need to be solved before full utilization of PaaS platforms in containerization. According to Dua et al. (2014), the following features are still needed:

- Standardization: There is not a standard container file format and for full interoperability this is a necessity to have.
- Security: From perspective of isolation in networking and memory, containers need to secure both.
- Independence of OS: Working feature of abstraction is needed, as containers should not be tied to userspace or specific kernel.

2.2 Other available containers

There are many container technologies available, but in this thesis the two alternatives introduced are Rkt (or Rocket) and LXC. Rkt is predicted to be one of the toughest rivals for Docker in the future and LXC is the predecessor of Docker.

2.2.1 Rocket – project rkt

Project "rkt" was launched after Docker changed their plans about standardized containers and focused on building tools for full platform products. Docker even removed the container manifesto that they published in 2013 and this led the team of CoreOS to develop their own standardized container system. (Polvi, 2014). CoreOS is nowadays a big Docker container platform, which has container hosting services in the cloud. CoreOS is trying to develop a simplified container solution with built-in security from the start and support for socket activation, as these changes would provide standardized solution without the flaws of Docker. (Janmyr, 2015). Rkt also have severe production and security requirements with compatibility with other available containers. However, while rkt is progressing rapidly and is promising, it is not near a stable solution, as it is still in alpha phase. (Polvi, 2014). In December 2015 the latest version was v.0.13.0.

2.2.2 LXC – Linux containers

LXC or Linux containers are the predecessor of Docker. LXC uses the namespaces and control groups (cgroups) in the same way as Docker do to guarantee isolation of containers. LXC containers used the process identification (PID) and network namespacing for the first time. LXC also developed the method of resource sharing and management via cgroups. (Xavier, Neves, & Rose, 2014). The below table shows how identical Docker and LXC are, as Dua et al. (2014) have compared these two:

Table 1 - Comparing containers (Dua et al., 2014)

Parameter	**LXC**	**Docker**
Process Isolation	Uses pid namespace	Uses pid namespace
Resource Isolation	Uses cgroups	Uses cgroups
Network Isolation	Uses net namespace	Uses net namespace
Filesystem Isolation	Using chroot	Using chroot
Container Lifecycle	Tools: lxc-create, lxc-stop, lxc-start to create, start, stop a container	Uses Docker daemon and a client to manage the containers

2.3 Docker containers

Docker is an open source product that has many capabilities from previous technologies such as LXC containers, Git type of version control and operation system virtualization. (Boettiger, 2015) With containers, applications share an operating system and whenever possible, also binaries and libraries. The result is that these deployments will be drastically smaller in size than hypervisor de-

ployments, making it possible to store hundreds of containers on a single physical host (Bernstein, 2014).

2.3.1 Docker in practice

Basically, Docker extends LXC with a kernel- and application-level API that both run processes in isolation (Bernstein, 2014). Docker relies heavily on two pieces of Linux kernel technology which are namespaces and control groups (Anderson, 2015). Docker uses namespaces to completely segregate an application's view of the underlying operating environment, including networking, user IDs, file systems and all other processes (Bernstein, 2014). Control groups are designed for managing available resources to a container. These resources can be restrictions of access to other processes or bare resources of given hardware (Anderson, 2015).

Docker containers are created using base images. A Docker image can have just the minimum requirements of the operating systems, or it can consist of a sophisticated prebuilt application stack ready for launch. (Bernstein, 2014) Docker can cache things as it will not install or change environments if it is not really a must (Anderson, 2015). When building images with Docker, each action taken, for example "apt-get install", forms a new layer on top of the previous one. These commands can be used manually or executed automatically by using Dockerfiles (Bernstein, 2014).

Docker images share Linux kernel with host machine which is the largest difference between other virtualization technologies and Docker images (Boettiger, 2015). This technology allows that Docker is not as heavy to run as other virtual machines. This feature, that there can be hundreds of containers in one machine with higher performance, has made Docker really attractive for industry and is also the reason for its enormous popularity (Boettiger, 2015). Docker hub is the place for central repository for private and public Docker images. In the Docker hub users can search and share their images and download them with the Docker client. (Bui, 2014)

Dockerfiles automate the process of building images (Anderson, 2015). Dockerfiles have a straightforward script which has same kind of syntax as Makefile. These files have exact commands how to build current image with wanted values (Boettiger, 2015). Those whom write Dockerfiles need a bit familiarity of Linux control commands, such as "apt-get install", before executing written scripts (Boettiger, 2015). At the moment Dockerfile has 13 commands available. (Janmyr, 2015) Another example is that any "RUN" commands in Dockerfile will create a new layer for the container and from these layers Docker combines in the build process a container, which does not have duplicates and can re-use wanted processes. (Karle, 2015)

Docker containers have many already described features, but one of the core ones is Docker engine, which builds and runs Docker containers. Docker containers run on Docker engine, which controls all containers. Docker client provides the user interface (UI) for user and also the interaction to containers.

Therefore the Docker daemon communicates with client and thus sends commands to run, ship or build containers. (Bui, 2014)

2.3.2 Namespaces and control groups

As mentioned before, namespaces and control groups are the core isolation of container technology. Namespaces provide the certainty that container can only see its own environment and thus it will not affect other containers. It will give restricted access to file systems as change root (chroot) and does not give any rights above containers own level of access. Namespaces also give own network devices to containers, so each container will have unique hostname and IP address for independency. (Joy, 2015)

Almost all containers use control groups for securing the resource sharing. Control groups ensure that each and every container has enough resources to work properly, but even importantly they prevent over exhaustion of those resources. (Dua et al., 2014).

3 Advantages of Docker as container technology

Docker has many capabilities and implementations for user friendliness, performance and security. These crucial components for success are LXC based OS virtualization, re-use of components, portable container deployments, versioning of images, using of image archives and sharing those images (Boettiger, 2015). At the moment Docker still needs Linux-compatible software under it. However, Docker is also included in Windows Server 2016 edition, which is a huge accomplishment.

3.1 Dockers usability

Containers outperform traditional virtualization in many ways, but intensely in performance and scalability. Most of the cloud based systems need an eased scalability option, where Docker containers play a great solution. Instead of running multiple virtual machines, which consume precious resources with Guest OS instances, Docker can launch very rapidly a lot more container instances without any overhead. (Joy, 2015.) Another key issue in software development is "dependency hell", which can also be solved with Docker, as Docker containers have all dependencies needed inside the container and if the release version of container is not broken, it will work in every instance of Docker elsewhere (Boettiger, 2015). Docker also has tools against code-rot with image versioning. Any changes in the deployment pipeline, for example as new features or dependency changes can cause serious breaking in otherwise working code (Boettiger, 2015). These problems can be drastically reduced with Docker, as software environment is dictated to specific operation system and its dependencies with ease and this also provides increased software security (Boettiger, 2015).

According to Joy (2015), Docker's advantages come from five key values in terms of usability:

- Portable deployments
  Applications built inside containers are exceedingly portable, as these kinds of bundles are moved around as a single unit and this movement does not affect performance or the container at all.
- Rapid delivery
  Because of Dockers standardized container format, software teams do not need to worry about each other's tasks, as developers are only taking care of applications inside the container and administrators are only taking care of the server deployments with containers. As containers or packets are tested and have all the needed dependencies, they will work in every instance.
- Scalability
  Docker containers can run in every Linux system as it can also be deployed to various cloud environments, datacenters and physical servers etc. User can easily move container from cloud to desktop and back to cloud in rapid pace. Scaling up and down is also a walk in the park, as user can adjust the scale from one to thousands and back to one if it is not needed.
- Faster build times
  Containers are really small and thus build times are rapid. This allows reduced times in testing, development and deployment. After container has been built, it can be pushed to test environments and from there to production environment.
- Higher density with better performance
  As Docker containers do not use hypervisor, available resources can be used more efficiently. This means that there can be more containers in single machine than there can be virtual machines and with that higher density and lack of wasted resources to overhead, Docker containers can have higher performance as well.

Docker is also one kind of "contract" between Developers and Operations. Many software teams have confronted problems with choosing the used technology, as developers want to use the newest technologies and operators want to use technologies that really work or have been used before. In this kind of common situation Docker is very useful, as developers can use all new technologies and built them inside to container and operations can deploy these containers with ease. (Janmyr, 2015.)

3.2 Performance comparison versus traditional Virtualization

Virtualization is not going anywhere soon. All public cloud giants, such as Amazon EC2, Google Compute Engine and Microsoft Azure, are relying to virtualization technologies to keep their systems powerful and scalable. However, tra-

ditional virtualization has its own expenses as mentioned before, so for the tremendous needs of computing power, cloud companies are already moving towards better resource utilization and faster deployments. (Joy, 2015). Google, IBM and Joyent are all successful public cloud providers, which have been using containers instead of virtual machines. (Bernstein, 2014).

Below Dua et al. (2014) have compared virtual machines against containers. As we can see, containers are very different than virtual machines.

Table 2 - VM and container comparison (Dua et al., 2014)

Parameter	Virtual Machines	Containers
Guest OS	Each VM runs on virtual hardware and kernel is loaded into in its own memory region.	All the guests share same OS and Kernel. Kernel image is loaded into the physical memory.
Communication	Will be through Ethernet Devices.	Standard IPC mechanism like signals, pipes, sockets etc.
Security	Depends on the implementation of Hypervisor	Mandatory access control can be leveraged.
Performance	Virtual Machines suffer from a small overhead as the Machine instructions are translated from Guest to Host OS.	Containers provide near native performance as compared to the underlying Host OS.
Isolation	Sharing libraries, files, etc. between guests and between guests hosts not possible.	Subdirectories can be transparently mounted and can be shared.
Startup time	VMs take a few minutes to boot up.	Containers can be booted up in a few seconds as compared to VMs.
Storage	VMs take much more storage as the whole OS kernel and its associated programs have to be installed and run.	Containers take lower amount of storage as the base OS is shared.

Docker is also shining in higher data volumes. High Throughput Computing (HTC) and Many Task Computing (MTC) both have billions of tasks, which are intensive in aspect of computing power or data management. These experiments require reliable and fast access to memory and also management of high data volumes. In recent study container-based systems as Docker were significantly more efficient in execution times and also in memory management than hypervisor-based virtualization systems. (Adufu et al., 2015).

In terms of application deployment, those who want to use the least of infrastructure will use the simple container-based approach. This is the core rea-

son for cloud vendors to have improved performance in container-based systems, as many people tend to use these simplified and cheaper solutions. A recent study in apples-to-apples benchmark test with "fast data" had five times better performance in container-based IBM Softlayer system than compared against hypervisor-based Amazon AWS. (Bernstein, 2014). Kernel-based Virtual Machines (KVM) is a mature technology, which is widely used. When KVM and Docker were both tuned to highest performance settings, Docker exceeded or equaled KVM in all of the test cases. (Felter, Ferreira, Rajamony, & Rubio, 2015). These tests cases give an impression of Docker containers superiority in every case compared against traditional virtualization. However, a hypervisor-based virtualization is still useful when applications require different operating systems with various versions on the same cloud (Bernstein, 2014).

3.3 Security advantages of Docker

Security is a major challenge when used services are located in virtual environments. It has been said that hypervisor virtualization is more secure than container-based virtualization, as container-technology can communicate directly to host kernel whereas hypervisor prevents it. Docker is a container-based virtualization, thus it raises security concerns against it. (Bui, 2014) However, there are many ways to provide reliable security in Docker, when new innovations and old architecture, as namespaces and control groups, are properly used. As Docker Inc. mentions in their Website that the key focus for Docker in software security is to make the highest level of security possible without sacrificing any usability (Diogo, 2015).

When following question was asked from audience in IT seminar: "What is your greatest concern?" the response was "someone subverting our deployment pipeline" (Bass et al., 2015). Software is the heart of many missions and critical operations and the deportment is always relevant. However, software development is advancing more rapidly than cautious development and security verification can handle. Thus, cyber security needs to be included in every level of software development process in every way possible (Bradley et al., 2011). Introduction of standardized image format has rocketed the interest of using Docker containers in enterprises. These containers facilitate software distribution and allow greater use of resources. Docker containers are currently sharing the same host kernel, thus it is possible to gain access over the system via container if developers are not aware enough. This leads us to conclusion: Docker's security is mostly about limiting and monitoring the possible attack surface (Mouat, 2015).

Namespaces are providing the first form of security with isolation. Those processes, which are running inside a container, cannot see or affect any other process in another container. Control groups are the second most valuable form of first line security, as it is sharing and limiting resources. For example, control groups ensure that each process has enough CPU power or memory, but it is

also ensuring that a Distributed Denial-of-Service (DDoS) attack inside a container cannot exhaust other resources. (Petazzoni, 2013)

As Bui (2014) stated in his research, while Docker is fairly secure even with default configurations, there still was a minor security issues regarding to default network model, as the virtual Ethernet Bridge was vulnerable to ARP spoofing and MAC flooding attacks. However, these security problems can be solved easily, if the network administrator is aware of these issues and adds correct filtering. There are also new security features published after Bui's (2014) article, as in autumn 2015 Docker introduced next-level security in version 1.8 with Docker Content Trust (DCT). This feature provides a public key infrastructure (PKI) for Docker. PKI approach has two different keys, which are root key (which is offline) and a tagging (per-repository) key, which is generated and stored when Docker image has been published for the first time. DCT adds digital signature to each data sent or received and therefore verifies safe Docker images from remote registries. Each of these repositories has their own unique keys, thus other containers security won't be breached when one key is revealed (Vaughan-Nichols, 2015). Docker Content Trust is built-in feature in 1.8 and later versions, thus it is recommended to use for stronger security.

According to Mouat (2015) there are a few possible exploits in Docker:

- Kernel exploit
  In container-technology the kernel is shared with all of the containers, which vastly increases the importance of secure kernel. With a kernel exploit a single container could crash the whole host.
- Denial-of-service attacks
  Kernel resources are shared to all Docker containers. If control groups have been configured poorly, one noxious container can starve other containers in the host
- Escapes from Container
  Attacker shouldn't be able to gain access to another containers or the host itself.
- Poisoned images
  Without a trusted Docker image provider it is almost impossible to know is the shared image safe to use or what it has inside of it.
- Secret compromises
  For example when Docker container is using a database it will most likely require a secret inside of it, such as a password.

Reading above exploits or articles regarding to Docker security can make an impression of insecurity, which however is not the case. While using container-technologies there is a certain need to be aware of potential security issues, but if used properly, Docker containers can imply more efficient and better security than using virtual machines or bare metal servers alone (Mouat, 2015). Furthermore, even though containers have larger attack surface than virtual machines, it is possible to maintain performance and higher density without sacri-

ficing security. This can be done with correct configuration and awareness or just installing the container service inside a virtual machine. However, running Docker inside a Hypervisor virtualization system isn't the right medicine for some security issues, as it adds only an even more complex system. (Hemphill, 2015)

According to Hemphill (2015), the most important thing in software security is to understand the fact that security breaches happen. In these cases it is vital to figure how the breach occurred and fix it in such a way that it cannot happen again and with Docker software deployment these security patches can be applied very precise and quickly. Docker also adds availability of monitoring and auditing, thus potential attacks can be detected early on. (Mouat, 2015)

4 Conclusion

In this study, the container technology called Docker was examined for its advantages. The study was executed as review of literature from the most recent and reliable sources. The most important observation of this study is that Docker has many potential advantages in the fields of usability, performance and security. These observations were discovered during the progress of this thesis.

Containers have better usability, performance and scalability than traditional virtualization, as containers utilize resources far better for not having unnecessary overhead and containers also provide higher density. However, there are still use cases that fit better for virtual machines. (Bui, 2014; Joy, 2015). Containers have natural advantage over virtual machines, as containers even have reduced startup time, which is dictating better performance. By having additional layers on top of LXC based approach in containerization, Docker is perhaps the most relevant container technology for PaaS authors. (Dua et al., 2014).

Docker containers help the renewing software industry in terms of portability, delivery, scalability, speed and density. Each of these gives more usability and eases the work of developers and system operators, as containers remove many common problems or decreases the time used in each task. (Joy, 2015) One of the biggest problems removed is the "dependency hell" as containers contain all the needed dependencies and therefore properly executed and built containers are guaranteed to work in every other Docker instance (Boettiger, 2015). Another indicator of interest towards container technology and Docker is those promising results from test cases, where containers over performed or equaled traditional virtualization in every case. (Adufu et al., 2015; Bernstein, 2014; Felter et al., 2015)

As this study revealed, that for safe use of Docker, users and developers need to be aware of potential security issues and the most relevant tools and techniques of securing containers (Mouat, 2015). Containers are mostly positive effect in terms of security, as it provides extra level of isolation and better control. A system, which is using containers properly, will be more secure than an identical system without container-technology (Mouat, 2015). However, there are several things to keep in mind when using Docker, as it needs fierce config-

uration. Another great principle is to follow defense-in-depth and least privilege, as then a compromise of one component won't affect the full system. Several levels of security maximize systems safety and one penetration won't do any serious harm (Mouat, 2015). Docker isn't as insecure as it is generally thought and thus fully configured Docker development pipeline eases the actual software development process but even better, it increases the overall safety of the system: from scratch to release (Hemphill, 2015).

Overall Docker is really promising technology, which should be considered as replacement for traditional virtualization. Docker containers perform well with default configurations, but for better performance and security, heavy configuration is needed. Docker has fine features, which guarantee simplified usability and scalability. However, for the needs of cloud giants better clustering system is needed, where for example Docker Swarm[5] can fit. In this thesis the focus was only in Docker containers, so other features and other technologies were omitted. In future studies other container technologies and other container technology features should be evaluated in depth analysis with empirical research, as field of virtualization is revolutionizing. One thing is certain: Virtualization is not going anywhere soon and Docker is here to stay. (Janmyr, 2015; Joy, 2015).

[5] https://docs.docker.com/swarm/

REFERENCES

Adufu, T., Jieun, C., & Yoonhee, K. (2015). Is container-based technology a winner for high performance scientific applications? *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific,* 507-510. doi:10.1109/APNOMS.2015.7275379

Anderson, C. (2015). Docker [software engineering]. *IEEE Software,* (3), 102-c3.

Axelrod, C. W. (2014). Reducing software assurance risks for security-critical and safety-critical systems. *Systems, Applications and Technology Conference (LISAT), 2014 IEEE Long Island,* 1-6. doi:10.1109/LISAT.2014.6845212

Bass, L., Holz, R., Rimba, P., Tran, A. B., & Zhu, L. (2015). Securing a deployment pipeline. *Release Engineering (RELENG), 2015 IEEE/ACM 3rd International Workshop On,* 4-7. doi:10.1109/RELENG.2015.11

Bernstein, D. (2014). Containers and cloud: From LXC to docker to kubernetes. *Cloud Computing, IEEE, 1*(3), 81-84. doi:10.1109/MCC.2014.51

Boettiger, C. (2015). An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review, 49*(1), 71-79.

Bradley, M., Fehnker, A., & Huuck, R. (2011). Cyber security at software development time. *Defense Science Research Conference and Expo (DSR), 2011,* 1-4. doi:10.1109/DSR.2011.6026847

Bui, T. (2014). Analysis of docker security.*Aalto University School of Science*(Aalto University T-110.5291 Seminar on Network Security)

Diogo, M. (2015, August 12, 2015). Introducing docker content trust. Retrieved 26.11.2015 from http://blog.docker.com/2015/08/content-trust-docker-1-8/

Dua, R., Raja, A. R., & Kakadia, D. (2014). Virtualization vs containerization to support PaaS. *Cloud Engineering (IC2E), 2014 IEEE International Conference On,* 610-614. doi:10.1109/IC2E.2014.41

Felter, W., Ferreira, A., Rajamony, R., & Rubio, J. (2015). An updated performance comparison of virtual machines and linux containers. *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On,* 171-172. doi:10.1109/ISPASS.2015.7095802

Fink, J. (2014). Docker: A software as a service, operating system-level virtualization framework. *Code4Lib Journal, 25*

Hemphill, B. (2015, October 22, 2015). Docker, docker, docker, security, docker. Retrieved 26.11.2015 from http://containerjournal.com/2015/10/22/docker-docker-docker-security-docker/

Janmyr, A. (2015). A not very short introduction to docker. Retrieved 4.12.2015 from http://www.jayway.com/2015/03/21/a-not-very-short-introduction-to-docker/

Joy, A. M. (2015). Performance comparison between linux containers and virtual machines. *Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances In,* 342-346. doi:10.1109/ICACEA.2015.7164727

Karle, A. (2015). Operating system containers vs. application containers. Retrieved 4.12.2015 from https://blog.risingstack.com/operating-system-containers-vs-application-containers/

Linthicum, D. (2014). Cloud app containerization all the rage -- but is it anything new? Retrieved 4.12.2015 from http://searchcloudcomputing.techtarget.com/podcast/Cloud-app-containerization-all-the-rage-but-is-it-anything-new

Luo, S., Lin, Z., Chen, X., Yang, Z., & Chen, J. (2011). Virtualization security for cloud computing service. *Cloud and Service Computing (CSC), 2011 International Conference On,* 174-179. doi:10.1109/CSC.2011.6138516

Mouat, A. (2015). In Anderson B. (Ed.), *Docker security: Using containers safely in production* (First Edition ed.) Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Pahl, C. (2015). Containerization and the PaaS cloud. *Cloud Computing, IEEE, 2*(3), 24-31. doi:10.1109/MCC.2015.51

Petazzoni, J. (2013, August 21, 2013). Containers and docker: How secure are they? Retrieved 26.11.2015 from http://blog.docker.com/2013/08/containers-docker-how-secure-are-they/

Polvi, A. (2014). CoreOS is building a container runtime, rkt. Retrieved 4.12.2015 from https://coreos.com/blog/rocket/

Vaughan-Nichols, S. (2015, August 13, 2015). Docker 1.8 adds serious container security. Retrieved 26.11.2015 from http://www.zdnet.com/article/docker-1-8-adds-serious-container-security/

Xavier, M. G., Neves, M., & Rose, C. (2014). A performance comparison of container-based virtualization systems for MapReduce clusters. *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference On,* 299-306. doi:10.1109/PDP.2014.78

Xavier, M. G., De Oliveira, I. C., Rossi, F. D., Dos Passos, R. D., Matteussi, K. J., & De Rose, C. A. F. (2015). A performance isolation analysis of disk-intensive workloads on container-based clouds. *Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference On,* 253-260. doi:10.1109/PDP.2015.67