The benefits of virtualization across the software development pipeline

University of Oulu
Department of Information Processing Science
Bachelor's Thesis
Andrea Peltokorpi
13.6.2021

Abstract

The emergence of cloud computing and the evolution into service-based solutions across the software industry have influenced many changes in software development paradigms and methods. As a result, various forms of virtualization and container-based solutions have become more and more commonplace throughout the field, with technologies and frameworks such as Docker and Kubernetes becoming industry standard solutions to virtualization.

This thesis is a literature review into existing research on virtualization and containers, and their use in various categories of the software industry. The aim of the thesis is to look at the reasons for the proliferation of virtual machines and containers, along with their benefits for the software development process, the continuous integration and delivery pipeline, and the different cloud platforms and providers.

The benefits of virtualization are clearest in the cloud infrastructure, as cloud services are inherently built to utilize virtual machines. Containers and container orchestration systems allow container management and dynamic resource allocation, improving efficiency and reducing costs. In software development and testing, the modular and self-contained nature of containers allows for faster iteration and more problem-averse development. And finally, in the continuous integration and delivery pipelines, containers and container management tools allows automation, and lower overhead and complexity, enabling lower-threshold software deployment. Along with enabling cloud infrastructure as it exists today, the evolution of virtualization and containers in the software industry provide benefits across the board.

Keywords
containers, virtualization, Docker, cloud

Supervisor
PhD, university lecturer Mikko Rajanen

Contents

1. Introduction

Over the past decades the rise and popularization of the internet has led to the development of a wide range of standards and practices. Many paradigm shifts have evolved how online software is created and operated. Perhaps the most notable development is the focus and reliance on cloud computing, and the ever-increasing amount of data-intensive software systems has paved way for distributed, scalable software. This has transformed the way online platforms work, and it has created whole new cloud-based industries (Gorton and Klein, 2014).

Different forms of distributed computing and the rapid mass servicification of cloud-based architecture has made way for compartmentalized, container-based frameworks. This has led to various procedures for software development and delivery that utilize these new methods (Marathe, Gandhi, & Shah, 2019). In order to improve effectiveness of modern software development and deployment, platform virtualization using virtual machines and containers has become prevalent. Technologies such as Docker, Kubernetes, Vagrant and VirtualBox have become highly common parts of the toolset of the integration and delivery pipeline for online software (Koskinen, Mikkonen, & Abrahamsson, 2019).

The research question of this thesis is how virtual machines and particularly containers are used as a part of software development, deployment pipeline and cloud infrastructure. I am interested in the modularity allowed by virtualization and containers, and the various ways this can benefit software development and cloud platforms. There are multiple components to containers and virtualization in this thesis: their benefit to the software development process, their benefit for cloud platforms, the simplification of the software deployment pipeline, security, modularity and others. However, the focus of this thesis will be on the overall factor of using containers throughout the software development and deployment landscape – the scope will remain fairly general instead of focusing deeply into any single aspect. Though individual prolific container technologies (i.e., Docker and Kubernetes) will be taken into account, their technical specifics are outside the scope of this thesis.

The thesis is a literature review of existing research into virtualization, virtual machines, and containers. As the technologies in question are comparatively novel, there is plenty of relevant and fairly recent research into the qualities and benefits of a container-based approach to software and cloud architecture. According to Koskinen et al. (2019) containers are a new research area, with a vast majority of research having been released after 2017. As a result, the topic is quite fast evolving, with many different approaches for research – some more architectural and technical, some more practical. For example, Wan, Guan, Wang, Bai and Choi (2018) write about Docker containers with a focus on deployment and operation cost. Liaqat, Naveed, Ali, Shuja and Ko (2019) focus on the performance benefits of virtual machines for cloud environments, and Combe, Martin and Di Pietro (2016) look at Docker from the perspective of security.

Structurally this thesis will begin by looking at the modern, emergent cloud platforms. The focus is on the rise of infrastructure, platforms and software as services, and the space of software development in this new paradigm. Next, the perspective will shift to virtualization and containers, with research describing motivations for their

development, and their attributes. Examples of research into particularly prevalent technologies will be looked at as well. Next, container clusters and their role in continuous integration and continuous delivery will be looked at. Finally, the thesis will focus on the usage of containers as a part of the software development process and deployment pipeline. This section will look at the container-based frameworks and their role in software development, and as they coexist in the array of different cloud platforms.

2. Method

This thesis approaches the research question from a fairly general point of view. The aim is not to create new research into the topic, but to go through and collate existing research on the subject instead. As such, the method chosen for this thesis is a literature review on research into the various aspects of the topic in question. Container-based architecture is still fairly new in the software landscape, but the various technologies have become increasingly popular – an argument could be made, that some form of virtualization as a part of software development and delivery has become the industry standard. Due to this fairly speedy spread of use, there is plenty information on the topic in various online media, but much of it is non-scientific and descriptive in nature or intended as a tutorial. A literature review is therefore appropriate in order to gauge the varying scientific research into the qualities and benefits of virtualization and containers.

The literature review in this thesis aims to collate a wide variety of research with different approaches and points of focus. The research was largely gathered from various databases with a focus on peer-reviewed research in order to maintain a scientific focus. Various scientific databases were used in order to scope and find the literature to be collated. Particularly IEEE Xplore, Scopus, Ebsco and Google Scholar were used to collect the initial set of articles on the topic. The search for peer-reviewed articles was particularly focused on the topics of containers, Docker, virtualization, and virtual machines, and additionally on cloud, cloud architecture, and software deployment. Even though the topics are so new, a large amount of research could be found, with many different points of focus.

Once the initial set of articles had been gathered, and the structure and general outline of this thesis was beginning to form, additional relevant research was gathered in order to complement the literature gathered so far. This was done in two ways: first by going through the initial articles' references for relevant topics, and then looking at research referencing the initial articles. The aim was to create a fairly complete picture of container-based paradigms and how they relate to cloud computing, so a variety of relevant, scientific research was necessary and useful.

Like the topic itself, the selection of research for this literature review is quite new. A majority of the articles are published no earlier than 2015, with a focus on research published after 2017. Some outliers are also included, published earlier in the 2000s, with this research focusing on software virtualization before container-based ecosystems became more prevalent. These articles look at how virtualization in and of itself is a valuable resource in software development and deployment, and how various pre-container virtualization tools integrate into different models and stages of software development.

3. Containers in software deployment

With the rise of cloud computing, the modern software field has been transformed in many ways. The business of creating and upkeeping software has evolved to host various service-based paradigms, such as Infrastructure as a Service (IaaS), Software as a Service (SaaS) and Platform as a Service (PaaS), with various effects to the whole industry. This has made platform virtualization and containers quite ubiquitous in the software field and a major part of everyday operation in the software industry (Syed & Fernandez, 2015). As a result, virtualization and containerization as concepts have become increasingly important.

3.1 Continuous software delivery and cloud computing

Over the past years, the traditional models for software development, delivery and use have evolved. The classic process of software deployment has consisted of creating and building software, packaging it, and delivering it to the user or target machine to be installed. Software has been seen as a complete product, a ready-made package that can be sold and bought (Dolstra, 2001). Whereas previously the customer would purchase a complete product with specific features, and further development is user support at best, nowadays people expect a continuous line of development and evolution for the software they use. This has led to a requirement for continuous software development, support and delivery (Rodríguez et al., 2017).

The changing expectations and usage behaviors have affected the software development process as well. The methods and practices have evolved to match the expectation of continuous development; the focus has increasingly become to constantly and proactively deliver value to the user, instead of a more methodological and plan-driven development paradigm. This has led to the proliferation of various agile and lean development and deployment processes (Pikkarainen, Salo, Kuusela, & Abrahamsson, 2012).

At the same time, the size and complexity of software has increased, making development more expensive and prone to various technical problems and errors. As a result, more modular solutions in development, architecture and delivery pipelines have become necessary to increase interoperability and problem-aversion (Rodríguez et al., 2017). More efficient automation through continuous integration (CI) and continuous delivery (CD) processes, along with structured version control systems have also become important parts of the software workflow (Sailer & Petrič, 2019).

On one hand, the expectations of the modern software customer and user, and on the other, the experience of the software provider has shifted the industry into more service-based paradigms; software is increasingly developed and delivered as a service, instead of a cleanly packaged product. This shift has coincided with the increasing move of software to cloud-based architecture, and the servicification of cloud infrastructure and platforms as well (Syed & Fernandez, 2015).

The process of developing software and deploying it into the cloud has facilitated a change in the landscape on multiple levels. The shift into cloud-based Platforms as a Service has provided an environment and mechanisms for a multitude of software

operations in the cloud: design for a cloud environment, deployment into the cloud, services, database migrations and build integrations. New workflows to take full advantage of cloud-based platforms have been developed, and more and more focus is put into the interoperability of PaaS clouds, and the light weight and modularity of cloud-based software solutions (Pahl, 2015).

At the same time, as the amount of data moving in the cloud has increased significantly, the requirements for cloud platforms have become more rigid. The various emergent cloud providers, such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud are expected to be able to store massive amounts of data securely, with high availability and low failure rates. If these Quality of Service (QoS) requirements are unable to be met, the cloud service provider must pay penalties to their users (Yousefipour, Rahmani, & Jahanshahi, 2018).

Providing all this cost-effectively has also necessitated the development of increasingly efficient and sufficiently high-performance tools and infrastructure solutions. As such, the evolution of the standards, methods and technologies of cloud-based platforms can have significant financial implications for the cloud platform provider (L. Zhao, Lu, Jin, & Yu, 2015).

3.2 Virtualization

The evolving realities of software use, development and delivery have led to a paradigm of abstraction and virtualization. In the process of virtualization, virtual machines are created to run software in specifically pre-set isolation – this is done in order to minimize the effect of varying development and running environments. To accomplish this, the hosting machine's attributes are hidden from the virtual machine upon creation by a hypervisor, and any software can be run in the virtual machine without fear of cross-contamination with the hosting machine (Shiraz, Abolfazli, Sanaei, & Gani, 2013).

With the expectation for agile workflows and the resulting rapid iterations, the efficiency and speed of software development, testing and delivery have become increasingly important. Optimally, developers and testers would each have an environment matching the production server. Different types of servers for preproduction and testing, mirroring the production server, have been used to get around this problem. But this creates an expensive and complicated overhead, as setting up physical servers is costly and inefficient (Duenas, Ruiz, Cuadrado, Garcia, & Parada G, 2009). Virtualization and virtual machines have created a solution for this; as the software in a production environment runs on a virtual machine, identical and independent virtual machines can be created as needed on each individual developer's and tester's machine. Along with more efficient development, the isolated nature of virtual machines reduces situations where a problem exists on one machine but not the other, simplifying troubleshooting and reducing IT costs (Buyya, Yeo, Venugopal, Broberg, & Brandic, 2009).

Along with the benefits to development and testing, virtualization enables more efficient and problem-averse deployment from development to production. As the parameters of individual developers' virtual machines are identical to the virtual machine running on the production server, the modularity allows for the changes to be deployed with improved convenience and automation. At the same time synchronizing configurations and content between development, testing and production environments (Kimovski et al., 2018).

With the rise of cloud computing and the Platform as a Service paradigm, modern software infrastructure has been needed as well, and the resulting Infrastructure as a Service (IaaS) model has evolved alongside cloud platforms. Virtualization is the backbone of modern cloud computing, and the IaaS model provides virtual machines that are hosted on cloud platforms, with pre-set virtual resources and an operating system. These virtual machines are then configured to contain specific software, such as the Software as a Service being developed (Rad, Bhatti, & Ahmadi, 2017). Though virtualization and virtual machines themselves are not a new phenomenon, the proliferation of cloud computing has been enabled by virtualization. According to Bernstein (2014) all cloud computing uses virtual machines in one form or another as a core of its functionality by definition, providing more efficient resource usage and elasticity, the ability to dynamically and rapidly allocate capabilities and resources with demand.

Virtualization has a clear financial effect on cloud platforms and cloud providers as well. The technical and financial overhead of running large server farms is notable and providing a platform for countless users is expensive. Virtualization and running virtual machines on top of the physical servers allows much more efficient resource usage; the virtual machines can be provisioned, shut down, and managed per usage and as needed. A single server can run multiple, completely independent and isolated virtual machines at the same time, each customized per the user's requirements and needs, maximizing efficient server resource utilization and providing necessary security (Kimovski et al., 2018; Liaqat et al., 2019). Inadequate resources and machine failures are frequent causes of Quality of Service violations by cloud service providers, and an additional key issue with large-scale cloud environments becomes energy efficiency. As a result, it is particularly important to pay attention to resource utilization as a factor (Yousefipour et al., 2018).

3.3 Containers and Docker

With the proliferation and ever increasing saturation of the cloud ecosystem, and its use of virtualization, much has happened in the field of virtual machines. While virtualization has been a solution to many of the software field's and cloud infrastructure's problems, there are still inherent weaknesses in traditional virtual machines (Pahl, 2015). Virtual machines based on hypervisors are still relatively weighty, as they are heavy stacks of a hypervisor, a full operating system, and relevant software and applications. In an agile environment this causes speed and performance challenges (Anderson, 2015).

Over the past years, the popularity of a container based solution to virtual machines has become more and more popular. Container-based virtualization enables even more light-weight, convenient and secure operation, with even less technical overhead than traditional virtual machines. Coinciding with the move to agile and lean software project frameworks, containers have proved a highly modular and portable solution to performance questions in traditional virtual machines (Syed & Fernandez, 2015).

As the name implies, traditional, hypervisor-based virtual machines operate as full, independent systems, with an installed operating system and all the necessary software running on top of the hardware and the hypervisor. Containers on the other hand increase the abstraction, holding packaged, completely self-contained and ready-to-be-deployed parts of applications. While both forms of virtualization are based on and created by virtual machine images, traditional virtual machines require a heavy image of a full operating system, slow to install, boot and operate. Due to their highly specific

nature, containers are based on light-weight, specific base images instead – only the specific micro-service's requirements are installed (Pahl, 2015).

Like traditional virtual machines, containers are almost fully insular, but they can be configured to communicate to each other by a container engine. This configuration allows for a highly modular and portable, but still interconnected framework, where each container can be run, developed, tested and deployed on its own (Syed & Fernandez, 2015). Because the containers share the host operating system instead of running on an operating system of their own, container operations such as rebooting are much faster and more efficient, because the whole virtual operating system does not have to be shut down and restarted, like with traditional virtual machines (Pahl, 2015).

Different patterns of container management can be recognized, constituting a varied set of container combinations. These patterns can range from singular containers to multiple interconnected containers on one node to multiple-container, multiple-node patterns. These different container design patterns can be compared to the design patterns that emerged after the emergence of object oriented programming, with similar benefits: separation of concerns, the capability to re-use components, and with the case of distributed clusters of containers, the ability to upgrade each container independently (Burns & Oppenheimer, 2016).

With the increasing containerization of virtual machines, various container solutions have become prevalent. Over the past few years, the development and adoption of containers has been driven in large part by Docker, an open source container packaging and delivery tool (N. Zhao et al., 2020). Docker functions as a framework for containers, allowing them to share libraries as needed, becoming even more light-weight, scalable and specialized. At the same time, Docker-based containers are exceedingly portable, making them particularly effective from a continuous integration and continuous delivery standpoint (Anderson, 2015).

Docker allows container creation and management with independent images stored in centralized registries, the most popular being Docker Hub. Using these images, Docker supports highly automated and simple creation of containers for a wide variety of microservices (N. Zhao et al., 2020). Essentially, Docker serves as a method to automate the creation and operation of containers. To improve interconnectivity and performance at the same time, Docker adds an additional layer to containers, automating their creation and operation. Additionally, critically, Docker's framework eases deployment into cloud-based platforms (Rad et al., 2017). The increasing shift into Platform as a Service has therefore been favourable to Docker-based containers. The improved performance and automation in both development and deployment drive up the usage and popularity of Docker as the Platform as a Service model becomes more prevalent (Bernstein, 2014).

3.4 Container clustering and Kubernetes

Resulting from the move to cloud-based Platforms as a Service and the widespread adoption of virtualization, using containers has become a very popular way of operating in the modern software field. With Docker, individual processes can be run in separate containers, and moved around across physical and virtual servers at will, without affecting the application's functionality. As a result, tracking and managing containers has become more complicated, and at the same time increasingly important (Moravcik & Kontsek, 2020). Because containers can be set up to communicate with each other, and an application might consist of several interconnected containers – each built to run a single part of the application – the containers can be linked together into a container

cluster running together in a shared virtual environment. In these clusters the internal complexities of the virtual machine can be hidden from the interconnected containers, but they can be managed together as a unit (Pahl, 2015).

While some containers function as singular containers, some of the recognized container design patterns have multiple related containers in a single node, or even spanning multiple nodes. These various multi-container design patterns require coordination and container orchestration (Burns & Oppenheimer, 2016). As a result, various container management systems have been developed to assist with container orchestration. These container management systems, or container orchestration engines provide an additional layer of abstraction to containers, coordinating these various containers and possibly monitoring their resources and status, performing updates, and managing micro-services run within the containers (Moravcik & Kontsek, 2020).

Particularly with the proliferation of Docker containers, different container management systems have been developed for the Docker ecosystem. Docker containers can be configured into a Docker Swarm, which is a mode for Docker containers where Docker containers are clustered as a group, regardless of how many nodes they span. They are then linked to each other within the Docker infrastructure, and they can be controlled as a unit. This allows for more efficient container management (Marathe et al., 2019). Docker containers are inherently designed for individual operation, but Docker Swarms allow for cooperation between large numbers of containers configured as a cluster and distributed over the network (Moravcik & Kontsek, 2020).

Container management systems also serve as a possible solution to potential security shortcomings with Docker containers. Using container-based solutions and Docker instead of hypervisor-based virtual machines has some inherent security implications, due to the containers' reliance on the host machine's own kernel for operation. Increased abstraction through container orchestration is suggested to alleviate these concerns (Combe et al., 2016; Martin, Raponi, Combe, & Di Pietro, 2018).

A noteworthy, high profile example of these Docker container management systems is Kubernetes, an open source framework developed by Google to oversee and manage clusters of Docker containers. After its announcement by Google in 2014, Kubernetes got endorsed by other major operators in the cloud computing field, such as Microsoft and IBM. As a result, both the Docker container ecosystem and the Kubernetes orchestration tool have become critical parts of modern cloud architecture (Bernstein, 2014). At its core, Kubernetes is deeply rooted in Docker technology. It allows to schedule and scale Docker containers in a cluster, making resource management more efficient and helping cloud platforms reach Quality of Service targets in their operation (Khatami, Purwanto, & Ruriawan, 2020).

With Docker and Kubernetes, orchestration is provided as Containers as a Service (CaaS). With the emergence of paradigms like DevOps and the increasing focus on rapid, continuous integration and delivery, Kubernetes offers increased automation, large number of different functionalities, and high usability through a web-based graphical interface. Its high modularity and open source nature along with its symbiotic link with Docker has given it a large community and a position as industry standard in container orchestration (Moravcik & Kontsek, 2020).

3.5 Benefits to software development

Due to the shift of the software industry to a Software as a Service model, the changed realities of software development have necessitated new ways of doing things.

Nowadays the expectation on the development team (responsible for the software development and testing) is to constantly provide value to the users, and the industry standard has become a continuous cycle of rapid iteration, testing, and delivery (Duenas et al., 2009). The paradigm shift to continuous development and agile frameworks has created an increasing need for efficiency in all stages of development (Rodríguez et al., 2017).

The promise of containers is to make software development easier and more hassle-free by introducing modularity, portability, automation and abstraction (Rad et al., 2017). With containers, and particularly portable container technologies such as Docker, developers can replicate the production environment on their own hardware with little manual setup needed. These light-weight container environments allow for rapid iteration and experimentation, as application components and their encapsulating containers can be built and rebuilt quickly. The modular nature of containers ensures that making changes to a single micro-service container does not affect any other containers (Koskinen et al., 2019).

As the software industry has evolved, the complexity of software has grown. The increasing number of dependencies creates technical overhead and high risk for integration problems. If the software works one way on one developer's machine, another way on a testing machine and a third way in production, the process can grind to a halt, wasting time and increasing cost of development (Anderson, 2015). With a Docker-based container framework, the setup and management of containers has been automated further, allowing the developer to iterate with even less overhead. Because Docker containers are created from immutable images pulled from a registry, each container will exist in a predictable state. This reduces the risk for software installation related conflicts and misconfiguration (Di Tommaso et al., 2015).

Another critical part of the cycle of iteration in modern software development is constant testing and test automation (Duenas et al., 2009). Continuous integration and various testing tools can be used to enable low-threshold unit testing and automated code analysis (Sailer & Petrič, 2019). In addition to enabling rapid local testing, Docker containers can be moved to separate testing environments with little difficulty and setup. This can be achieved particularly due to the modularity, and therefore the resulting portability of Docker containers (Rad et al., 2017).

Docker containers are able to communicate with each other as needed; Docker creates namespaces and virtual networks, allowing the containers to see each other, while hiding the complexity of the framework from the container (Anderson, 2015). At the same time, Docker containers are blind to other containers they are not clustered together with. This enables stacking multiple completely separate container clusters on the same system without risk of cross contamination, easing development and testing of separate, unrelated applications even further (Syed & Fernandez, 2015).

Even though various security implications can be found in the use containers and Docker, the effect of these security questions is mostly evident in the usage of containers in a production context. In a local development context however, with proper orchestration, separation, and isolation of micro-services, these security concerns are more minimal (Combe et al., 2016; Martin et al., 2018). As Duenas et al. (2009) note, the development team has much to gain from virtualization in general, and according to Di Tommaso et al. (2015), the use of containers and Docker only stands to amplify these benefits.

3.6 Benefits to cloud platforms and deployment pipelines

As the industry has shifted more and more towards Platforms and Infrastructure as a Service, and as agile methods and rapid, cyclical workflows have increasingly become the standard in software development, similar structures have appeared in the system administration side of the equation. The ever-increasing focus on continuous integration and delivery, and the emergence of paradigms such as DevOps have paved the way for more efficient deployment and delivery pipelines (Pikkarainen et al., 2012; Rodríguez et al., 2017). In this, more agile way of operation, an important benchmark becomes the simplicity of software deployment – the more complicated and cumbersome it is to deploy new versions of software under development, the more difficult it is to uphold rapid iteration and other agile ideals. Automated deployment as a part of continuous integration and continuous delivery has a positive effect on the release schedule and the reliability of both the release procedure and the software being released (Rodríguez et al., 2017; Sailer & Petrič, 2019).

Today, much of the Platform and Infrastructure as a Service ecosystem revolves around containers and container clusters, and this is in no small part due to benefits provided by container orchestration. The portable and robust framework of container clusters allows for increased control in a system administration context; managing containers as clusters in multiple separate nodes is critical for operation in an environment of multiple containers spanning multiple clouds (Pahl, 2015). The Docker ecosystem and Docker Swarm allow increased control over both singular containers and container clusters distributed over the network. Both the tools for more efficient orchestration and the light weight of Docker containers streamline management of the container infrastructure and help simplify and automate deployment processes (Moravcik & Kontsek, 2020).

An additional benefit of the Docker ecosystem in system administration and DevOps is its wide adoption rate. As Docker containers have become the standard infrastructure in cloud computing, orchestration tools such as Kubernetes, based on the Docker ecosystem are becoming more or less universal as well. The benefits provided by the shared technical features between Docker and Kubernetes drive mass adoption, and as a result, further development and integration of these tools (Bernstein, 2014).

Both the cloud platform providers and the system administration on the software development side of the equation have major financial incentive to optimize the deployment processes and the operation of the cloud platforms (Wan et al., 2018; L. Zhao et al., 2015). The tracking and management of container clusters, their operation, and resource optimization are all important benefits provided by Kubernetes. On the system administration side, the management of clusters across the cloud becomes simpler, and the elastic scaling of the clusters provides automatic resource optimization and efficiency for the cloud platforms (He, 2020; Marathe et al., 2019).

4. Findings and discussion

When looking at modern developments in the software industry, a critical factor is the shift into cloud computing and cloud-adjacent technologies (Pahl, 2015). This has led to various advancements in technologies, practices, and workflows in the field. These changes have changed both the opportunities and viable methods of utilizing software, as well as the requirements that come built in (Rodríguez et al., 2017). Notably, the expectation has become the delivery of constant value to the customers and users, instead of a singular, complete, packaged-and-shipped product (Pikkarainen et al., 2012). These paradigm shifts have resulted in the proliferation of various agile and lean workflows, both in software development, and in systems management. These methods focus on rapid iteration and testing in development and the maximum efficiency and automation in software deployment and delivery (Rodríguez et al., 2017).

At the same time, the industry has shifted into a service-based framework: Software as a Service, Platform as a Service and Infrastructure as a Service have increasingly become the standard methods of operation. Software as a Service has been an important factor in transforming the expectations to that of a constant delivery of value (Syed & Fernandez, 2015). The move into cloud has facilitated the emergence of Platforms as a Service, with various cloud platform providers offering both server capacity, and tools for utilizing the platform efficiently. This paradigm of service and the associated Quality of Service requirements give platform providers a financial incentive to further develop process efficiency, leading to the development of various continuous integration and delivery tools (Pahl, 2015). Finally, the Infrastructure as a Service model has become popular due to the proliferation of virtual machines, as cloud computing is inherently based on virtual machines and virtualization. In this paradigm of service, different types of virtual machines are provided as a service to run software, both locally and on more dispersed cloud platforms (Rad et al., 2017).

Due to the emergence of cloud platforms, and the increasing reliance on distributed computing, virtualization and various different types of virtual machine solutions have become standardized (Bernstein, 2014). The evolving requirements and expectations of software development and delivery have also favoured increased abstraction and virtualization throughout the process (Pahl, 2015). Virtual machines are created with varying degrees of separation and isolation from the hardware underneath, and the hosting system's properties are hidden from the virtual machines. The benefits of this are evident in development and testing, in software delivery, and in the operation of cloud platforms (Shiraz et al., 2013).

Virtual machines provide a degree of portability, as they can be created, copied and set up across each individual developer's and tester's personal machine. The identical properties of each virtual machine, and the degree of isolation allow for more problem-averse development, as the software runs on identical and isolated environments (Buyya et al., 2009). This also allows for increased efficiency and automation when deploying the software into the cloud, as these same virtual machines are run on cloud platforms; if it works on one host machine, it should work on any regardless of the host machine's properties (Rad et al., 2017). For the cloud platform provider, the benefits are even clearer, as virtualization is the backbone of cloud computing (Bernstein, 2014). Virtualization allows multiple systems to run on a single host machine at the same time, with each being blind to the host machine's features. Each virtual machine can be set up

with unique properties relevant to the software it is running without risk of cross-contamination with the host machine or the other virtual machines running on the system. This allows more efficient resource utilization in the cloud platform, improving performance, and reducing costs (Kimovski et al., 2018; Liaqat et al., 2019). Additional benefits to energy efficiency can also found with the proper utilization of virtual machines (Yousefipour et al., 2018).

While virtual machines have become the norm over a longer period of time, a more recent development is the increasing proliferation of container-based virtualization (Syed & Fernandez, 2015). As opposed to traditional, hypervisor-based virtual machines, container-based virtual machines increase the abstraction further, by splitting each part of the application into separate micro-services. These micro-services are run in light-weight, self-contained capsules, containing only the micro-service in question and its relevant requirements (Pahl, 2015). Traditional virtual machines provide a whole virtual environment with a complete operating system installed with its adjacent software, and this can cause some operational speed and performance challenges. Operations such as booting and rebooting the virtual machine and making changes to its properties can be a slow process (Anderson, 2015). Because containers use the host operating system to function, managing containers is more light-weight, allowing for improved operation and iterative, fast-paced use (Syed & Fernandez, 2015).

Like traditional virtual machines, containers are based on a paradigm of self-containment and isolation, though they can interface and communicate with other relevant micro-services through pre-set paths (Syed & Fernandez, 2015). Each container and micro-service is its own functional entity. This modularity allows for changes to each individual container and its micro-service without affecting the functionality of the other micro-services, and the application as a whole (Koskinen et al., 2019). The modular nature of containers, and the separation of concerns it provides allows for various container design patterns of varying amounts of containers in a single host machine, or multiple separate nodes. This can be compared to the paradigm shift towards object-oriented programming, and the resulting benefits to programming (Burns & Oppenheimer, 2016).

The movement into container-based virtual machines, and the resulting mass adoption of containers has been driven in large part by Docker, an open source container framework (N. Zhao et al., 2020). Docker allows for packaging and managing containers through a unified framework. It creates an additional layer to containers, increasing interconnectivity and performance. This allows the containers to share libraries and requirements, making each individual container increasingly light-weight on its own (Rad et al., 2017). On the other hand, the increased interconnectivity and the reliance on the host machine's own kernel can lead to security implications. These concerns seem to affect the operation of containers mainly in production environments, and they can possibly be alleviated with container management systems (Combe et al., 2016; Martin et al., 2018).

The shift into Platforms as a Service has favoured the adoption of Docker-based containers (Bernstein, 2014). Like with containers, the rapid iteration in development and testing allowed by Docker reduces overhead and complexity in software development environments (Rad et al., 2017). Docker allows the automatic, fast creation of containers for a wide variety of micro-services and technologies through light-weight images stored in central repositories (N. Zhao et al., 2020). As each container is created from an immutable image in a predictable state, problems stemming from installation and misconfiguration are unlikely (Di Tommaso et al., 2015). Additionally, Docker containers along with the Docker ecosystem improve the

portability of containers further, providing benefits for DevOps, and making continuous integration and delivery easier and more efficient (Anderson, 2015).

Resulting from applications and virtual machines being split into micro-services and containers, the management of these interconnected clusters of containers is important. Particularly as a container cluster can span various nodes in the cloud, orchestrating interconnected containers as a unit has become a key factor in cloud computing (Pahl, 2015). Different container orchestration systems exist to assist with forming related containers into clusters and operating and managing them as a unit when necessary. With these orchestration engines, the containers' status and resources can be monitored, and adjusted dynamically as needed (Moravcik & Kontsek, 2020). Docker's own ecosystem allows Docker containers to be organized into a Docker Swarm, allowing for management of the containers together regardless of how many nodes they span. This increased control helps simplify software development and automate container operation in the cloud (Marathe et al., 2019).

An important part of cloud computing and the modern continuous integration and delivery pipeline is Kubernetes, a container orchestration system developed by Google, based on the Docker ecosystem (Bernstein, 2014). Kubernetes allows high scale operation of Docker container clusters across the cloud, with monitoring and dynamic, elastic scaling of resources, and increased automation in deployment and container management. This is all managed through a usable, web-based graphical interface (Moravcik & Kontsek, 2020).

Cloud platform providers have much to gain financially from the increased automation and efficiency provided by Kubernetes (Wan et al., 2018; L. Zhao et al., 2015). With the automated resource management and elastic scaling, cloud platform providers have an easier time reaching Quality of Service targets (Khatami et al., 2020). As a result, Kubernetes has been widely adopted by multiple large cloud platform providers (Bernstein, 2014). And as Kubernetes is inherently tied to the Docker ecosystem, both Kubernetes and the Docker ecosystem have gained large userbases, and they have become industry standards in container-based virtualization and container orchestration solutions (Moravcik & Kontsek, 2020).

Based on these findings, it becomes clear that virtualization and containers are a key factor in various aspects of the modern software industry. On one hand, the cloud infrastructure as it exists today, and the emergent Platform as a Service model are enabled inherently by the use of virtualization. On the other hand, virtualization and containers bring clear benefits to the software development and continuous integration and delivery processes. Specifically, the Docker ecosystem and Kubernetes have become industry standard solutions both in cloud computing and in software development and DevOps by building upon earlier iterations of virtualization. This has been accomplished by increasing abstraction, automation and scalability, and easing container management and operation.

5. Conclusion

This thesis was a literature review with the research question of how virtualization and containers are used in software development, deployment pipelines, and cloud services. The aim was to look at existing research into various stages of virtualization technologies, from earlier, traditional virtual machines to modern container-based virtualization solutions. This was done to get an understanding of most recent developments, as virtualization technology has developed by building upon old technologies and paradigms. At the same time, the plan was to look at the benefits of virtualization and containers on three different aspects of the software field: first the cloud infrastructure and platforms, then the software development and testing workflow, and finally DevOps and the continuous integration and delivery pipelines.

The literature reviewed was varied, ranging along the categories of the research question. Noticeably, in the range of literature reviewed, the most attention was placed on the effects of virtualization, containers, and the Docker ecosystem on cloud platforms. Otherwise, the literature topics covered the different phases of the evolution of virtualization, and the different categories outlined in the research question. Additional topics had to do with shifts in software industry paradigms, such as the evolution of the industry into various service models, and the adoption of iterative and cyclical, agile development methodologies.

Through the literature review it became obvious that the clearest benefit of virtualization has been with the emergence of cloud computing and the Platform as a Service model. The various forms of virtual machines and containers are used extensively in the cloud architecture, and virtualization as a paradigm enables cloud computing as it exists today. The advancements in containers and container orchestration have major financial implications for the cloud platform providers, due to increased efficiency, automation, and resource scaling, along with improved tracking and management of container clusters distributed across the cloud. As the cloud services have formed into the Platform as a Service model, containers and container orchestration systems – Docker and Kubernetes in particular – have formed into an Infrastructure as a Service model alongside the cloud platforms.

Alongside the emergence of cloud computing, software development has gone through a transformation as well. The Software as a Service model has become the standard method of operation, changing customer and user expectations on one hand, and necessitating change in methods and workflows on the other. With the expectation of providing a service and continuously fixing, adding and developing the product further, much of software development has moved to cyclical methods of constant iteration. As the process has become faster, and the software more complicated, virtualization and containers have provided a benefit here as well. The light weight and self-contained, modular nature of containers allows developers and testers to iterate rapidly and make changes with less fear of large-scale effects to unintended parts of the application. Because the container-based environment is created in a predictable state and isolated from the host machine, the environment with its variables is identical between each of the developers' and testers' personal machines and the production environment. This reduces host machine specific problems during installation, configuration, development and testing.

The third category of the research question was the effect of virtualization and containers on software delivery pipelines. As the software development process has become more agile and iterative, the importance of efficient and low-threshold delivery pipelines has increased. The emergence of the DevOps paradigm and the continuous integration and delivery model has aimed to speed up, simplify and automate this process. As with software development, the identical and predictable nature of each virtual environment makes the delivery process easier, as the software's running environment is identical at both ends of the pipeline. At the same time, container orchestration systems have made container cluster management increasingly efficient, further enhancing the containers' self-contained but interconnected nature.

The topic of virtualization is nothing new in and of itself, but through constant progress and evolution in abstraction, efficiency and automation, containers have become an industry standard in the various fields of cloud services, and software development and management. The widescale adoption of the Docker infrastructure and Kubernetes has further solidified the container paradigm at the forefront the virtualization field, providing various benefits to the software development process on one hand, the cloud platforms and providers on the other, and the continuous integration and delivery pipeline in between.

References

Anderson, C. (2015). Docker. *IEEE Software*, *32*(3).

Bernstein, D. (2014). Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Annals of the History of Computing*, *1*(03), 81-84.

Burns, B., & Oppenheimer, D. (2016). Design patterns for container-based distributed systems. In *8th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 16)*.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, *25*(6), 599-616.

Combe, T., Martin, A., & Di Pietro, R. (2016). To docker or not to docker: A security perspective. *IEEE Cloud Computing*, *3*(5), 54-62.

Di Tommaso, P., Palumbo, E., Chatzou, M., Prieto, P., Heuer, M. L., & Notredame, C. (2015). The impact of Docker containers on the performance of genomic pipelines. *PeerJ*, *3*, e1273.

Dolstra, E. (2001). Integrating software construction and software deployment. In *Software Configuration Management* (pp. 102-117). Springer, Berlin, Heidelberg.

Duenas, J. C., Ruiz, J. L., Cuadrado, F., Garcia, B., & Parada G, H. A. (2009). System Virtualization Tools for Software Development. *IEEE Internet Computing*, *13*(5), 52-59.

Gorton, I., & Klein, J. (2014). Distribution, data, deployment: Software architecture convergence in big data systems. *IEEE Software*, *32*(3), 78-85.

He, Z. (2020). Novel Container Cloud Elastic Scaling Strategy based on Kubernetes. In *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)* (pp. 1400-1404). IEEE.

Khatami, A. A., Purwanto, Y., & Ruriawan, M. F. (2020). High Availability Storage Server with Kubernetes. In *2020 International Conference on Information Technology Systems and Innovation (ICITSI)* (pp. 74-78). IEEE.

Kimovski, D., Marosi, A., Gec, S., Saurabh, N., Kertesz, A., Kecskemeti, G., ... & Prodan, R. (2018). Distributed environment for efficient virtual machine image management in federated cloud architectures. *Concurrency and Computation: Practice and Experience*, *30*(20), e4220.

Koskinen, M., Mikkonen, T., & Abrahamsson, P. (2019). Containers in Software Development: A Systematic Mapping Study. In *International Conference on Product-Focused Software Process Improvement* (pp. 176-191). Springer, Cham.

Liaqat, M., Naveed, A., Ali, R. L., Shuja, J., & Ko, K. M. (2019). Characterizing dynamic load balancing in cloud environments using virtual machine deployment models. *IEEE Access*, *7*, 145767-145776.

Marathe, N., Gandhi, A., & Shah, J. M. (2019). Docker swarm and kubernetes in cloud computing environment. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)* (pp. 179-184). IEEE.

Martin, A., Raponi, S., Combe, T., & Di Pietro, R. (2018). Docker ecosystem–vulnerability analysis. *Computer Communications*, *122*, 30-43.

Moravcik, M., & Kontsek, M. (2020). Overview of Docker container orchestration tools. In *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)* (pp. 475-480). IEEE.

Pahl, C. (2015). Containerization and the PaaS cloud. *IEEE Cloud Computing*, *2*(3), 24-31.

Pikkarainen, M., Salo, O., Kuusela, R., & Abrahamsson, P. (2012). Strengths and barriers behind the successful agile deployment—insights from the three software intensive companies in Finland. *Empirical software engineering*, *17*(6), 675-702.

Rad, B. B., Bhatti, H. J., & Ahmadi, M. (2017). An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)*, *17*(3), 228.

Rodríguez, P., Haghighatkhah, A., Lwakatare, L. E., Teppola, S., Suomalainen, T., Eskeli, J., ... & Oivo, M. (2017). Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software*, *123*, 263-291.

Sailer, A., & Petrič, M. (2019). Automation and Testing for Simplified Software Deployment. In *EPJ Web of Conferences* (Vol. 214, p. 05019). EDP Sciences.

Shiraz, M., Abolfazli, S., Sanaei, Z., & Gani, A. (2013). A study on virtual machine deployment for application outsourcing in mobile cloud computing. *The Journal of Supercomputing*, *63*(3), 946-964.

Syed, M. H., & Fernandez, E. B. (2015). The software container pattern. In *Proceedings of the 22nd Conference on Pattern Languages of Programs* (pp. 1-7).

Wan, X., Guan, X., Wang, T., Bai, G., & Choi, B. Y. (2018). Application deployment using Microservice and Docker containers: Framework and optimization. *Journal of Network and Computer Applications*, *119*, 97-109.

Yousefipour, A., Rahmani, A. M., & Jahanshahi, M. (2018). Energy and cost-aware virtual machine consolidation in cloud computing. *Software: Practice and Experience*, *48*(10), 1758-1774.

Zhao, L., Lu, L., Jin, Z., & Yu, C. (2015). Online virtual machine placement for increasing cloud provider's revenue. *IEEE Transactions on Services Computing*, *10*(2), 273-285.

Zhao, N., Tarasov, V., Albahar, H., Anwar, A., Rupprecht, L., Skourtis, D., ... & Butt, A. R. (2020). Large-Scale Analysis of Docker Images and Performance

Implications for Container Storage Systems. *IEEE Transactions on Parallel and Distributed Systems*, *32*(4), 918-930.