Salla Timonen

Detecting Anomalies by Container Testing: A Survey of Company Practices and Typical Tools

Master’s Thesis in Information Technology

May 22, 2023

University of Jyväskylä

Faculty of Information Technology

Author: Salla Timonen

Contact information: `salla.k.timonen@jyu.fi`

Supervisor: Tommi Mikkonen

Title: Detecting Anomalies by Container Testing: A Survey of Company Practices and Typical Tools

Työn nimi: Poikkeavuuksien havaitseminen konttitestauksella: kyselytutkimus yritysten käytäntöihin ja tyypillisiin työkaluihin

Project: Master's Thesis

Study line: Software and Telecommunication Technology

Page count: 51+5

Abstract: Software testing is an integral part of any software process, but with the increasing interest and use of software containers, it is, unfortunately, being overlooked. With the growth and popularity of software containers, more and more security issues and concerns are arising. This thesis is about the process of detecting anomalies by testing containers and was done as a survey of company practices and typical tools. The aim was to get an overview of what the current approaches are and what might be areas that should be improved upon. This research validates some observations from other research and makes a contribution by showing what is the status of how companies currently test software containers. As a result, it can be concluded that the survey answers demonstrate how important testing truly is, the rise of interest and use of containers, security and vulnerability issues in software containers, and the varying ways of testing containers. There is no consensus on how testing is or should be accomplished, with advances being mainly driven within companies instead of academic research. Based on findings from the survey and literature, there is further beneficial research to be done regarding security and vulnerability aspects in software containers and in the development of general good practices on how testing software containers should be accomplished.

Keywords: Master's Theses, software containers, software testing

Suomenkielinen tiivistelmä: Ohjelmistotestaus on olennainen osa mitä tahansa ohjelmistoprosessia, mutta ohjelmistokonttien kasvavan kiinnostuksen ja käytön myötä se valitettavasti unohdetaan. Ohjelmistokonttien kasvun ja suosion myötä syntyy yhä enemmän tietoturvaongelmia ja huolenaiheita. Tämä pro gradu -tutkielma perehtyy poikkeavuuksien havaitsemisprosessiin konttien testauksessa, ja se toteutettiin kyselytutkimuksena yritysten käytäntöihin ja tyypillisiin työkaluihin. Tavoitteena oli saada yleiskuva siitä, mitkä ovat nykyiset lähestymistavat konttien testaukseen ja mitkä voisivat olla alueita, joissa olisi parannettavaa. Tämä tutkimus vahvistaa joitain havaintoja muista tutkimuksista ja antaa panoksensa osoittamalla, mikä on yritysten nykyinen tila konttien testaamisessa. Tuloksina voidaan todeta, että kyselyn vastaukset osoittavat testauksen tärkeyden, konttien kiinnostuksen ja käytön lisääntymisen, konttien turvallisuus- ja haavoittuvuusongelmat, sekä monet erilaiset testaustavat. Ei ole yksimielisyyttä siitä, miten testaus pitäisi suorittaa, edistyksen tapahtuessa pääasiassa yritysten sisällä akateemisen tutkimuksen sijaan. Kyselyn tulosten ja kirjallisuuden perusteella on havaittavissa tarvetta lisätylle tutkimukselle ohjelmistokonttien turvallisuus- ja haavoittuvuusnäkökohtiin ja kehitettävä yleisiä hyviä käytäntöjä siitä, miten ohjelmistokonttien testaus tulisi suorittaa.

Avainsanat: pro gradu -tutkielmat, ohjelmistokontit, ohjelmistotestaus

List of Figures

List of Tables

Contents

1 Introduction

Testing is an integral and necessary part of software creation and is a crucial step to having a working and secure software container. Software containers are having a surge in use and are on a rising trend. Used more and more by most companies, risks and vulnerabilities related to them have continuously surfaced. There does not seem to be a consensus on how testing them should be accomplished, and advances are mainly being driven within companies instead of academic research, and in numerous ways.

The main objective of this thesis is to take a look at the current status of company practices and distinguish tools concerning how anomalies can be detected through testing software containers. This was chosen to be done via a survey, conducted during the spring of 2023 as part of a larger software container project: Containers as the Quantum Leap in Software Development (QLeap).

The topic for this thesis is scientifically pertinent due to the growing interest and use of software containers with more and more connected issues frequently emerging. In addition, it is a practically relevant topic due to being conducted as part of the project QLeap that the University of Jyväskylä and various companies are partaking in.

The research detailed in this thesis provides a glance at the present status of company practices, showing possible aspects that require additional research through the survey. The scope was determined to not only be within the QLeap project and the partaking companies but was open to any relevant information technology company wishing to participate in the survey.

The structure of this thesis is as follows: Chapter 2 delves into background information and what research is currently available on the topic of software containers and testing them. Chapter 3 focuses on the research methodology, including the research hypothesis and how the survey was structured. The results are shown in Chapter 4, and further discussion is in Chapter 5. Finally, Chapter 6 concludes this thesis.

2 Background

This chapter starts first with an overview of software testing and containers in sections 2.1 and 2.2 and then delves deeper into the background information and what research is currently available on the topic of software containers and testing them in section 2.3. Finally, section 2.4 summarizes the key theories found in the literature and comments on what is not found.

2.1 Software Testing

Software testing is an integral and well-known part of the creation of any software. It is an essential step in order to make a working program through verification and validation. Though the process can be costly, avoiding testing is even more costly (Gaur et al. 2016).

The article (Jamil et al. 2016) defines testing as a "process of evaluation that either the specific system meets its originally specified requirements or not", continuing to state that it is predominantly a process encompassing validation and verification in order to determine whether the system being developed meets user-defined requirements.

Another way to think about software testing is to look at the process being done. It is a process of executing programs with the intent of finding errors to secure software with no defects (Gaur et al. 2016).

When it comes to the process of software creation, testing must be implemented and thought of in the pre- and post-development process, as testing is part of the whole software development lifecycle. (Jamil et al. 2016)

An important distinction is also to be made here, and that is the difference between testing and debugging. Software testing is the process done in order to establish whether it meets specified requirements. Test cases are planned, strategies are specified, and results are evaluated. Debugging on the other hand is the process of fixing problems that have been found during testing. (Anwar and Kar May 2019)

There is a plethora of testing standards available and a general consensus on the matter. According to an article from 1988 by David Gelperin and Bill Hetzel (Gelperin and Hetzel 1988) the major testing models are:

- Phase Models Demonstration to make sure that the software satisfies its specification,
- Destruction to detect implementation faults,
- Life Cycle Models Evaluation to detect requirement, design, and implementation faults,
- Prevention to prevent requirement, design, and implementation faults.

In the article, it is also stated that a conflict may arise between the goals of demonstration and fault detection if the strategy is not carefully selected.

The goals for software testing include:

- Verification and validation with verifying the software works as desired and validating whether it fulfils conditions,
- Priority coverage with ensuring efficient and effective testing within budget and schedule limits,
- Balanced with balancing requirements, technical limitations, and expectations,
- Traceability with document preparation of both successes and failures of the testing process,
- Deterministic where what is being done, targets and possible outcomes should be known. (Sawant, Bari, and Chawan June 2012)

Testing techniques can be broadly categorized into static and dynamic testing. Static testing (e.g., walk-through, informal review, technical review, or inspection) refers to the method of testing where code is not executed whereas dynamic testing (e.g., correctness, performance, reliability, and security testing) is a technique in which the dynamic behaviour of the code is analyzed. (Anwar and Kar May 2019; Sawant, Bari, and Chawan June 2012)

The basic software testing strategies are unit testing, integration testing, acceptance/validation testing, and system testing. Unit is the smallest testable module and the benefits of unit testing include cost-effectiveness, providing greater reliability, and being able to test various parts separately and simultaneously. It also simplifies the debugging due to being limited to a small unit. Examples of unit testing techniques include functional testing, structural testing, and heuristic or intuitive testing. Integration testing is used for constructing the program structure and performing tests to uncover errors related to the interfaces. This is done by integrating the unit-tested components and testing them as a group. Some strategies include top-down and bottom-up integration testing. Acceptance testing is an approach to authenticating whether the product is made per standards and specified criteria. Examples of this are user acceptance testing, alpha and beta testing, operational acceptance testing, and contact and regulation acceptance testing. Lastly, system testing is testing conducted on a complete system in order to evaluate its compliance with its specified requirements. Examples of this are recovery testing, security testing, graphical user interface testing, and compatibility testing. (Anwar and Kar May 2019; Sawant, Bari, and Chawan June 2012).

Software vulnerabilities are security flaws, glitches, or weaknesses found in software code that could be exploited by a threat source, as defined by the Computer Security Resource Center (CSRC, no date). These vulnerabilities need to be evaluated and tested critically in software because the exploitation of these can hinder confidentiality, integrity, availability, and more. Vulnerability management can include steps like detection, evaluation, prioritization, and rectification. (Haque and Babar 2021)

Anomalies are inconsistent patterns in data that do not conform to normal behaviour. The importance of detecting anomalies is shown in many fields. An example in software systems is the connection between abnormal computer traffic patterns and potentially hacked computers sending out sensitive data. (Chandola, Banerjee, and Kumar July 2009)

These anomalies can be found through thorough software testing of the vulnerable software. Many approaches can be taken to detect anomalies and possible attacks. The three major categories of intrusion detection methodologies are signature-based detection (SD), anomaly-based detection (AD), and stateful protocol analysis (SPA). SD is a knowledge-based take on intrusion detection, which relies on signatures (or patterns) that correspond to recognized threats. SD compares these patterns against captured events, which is effective for known attacks, but ineffective for unknown ones. It is also time-consuming and hard to keep signatures up to date. AD on the other hand looks into any deviations from what is deemed to be "normal" behaviour such as activities, connections, hosts, or users. Unlike SD, AD is effective in detecting new vulnerabilities, but since observed events are constantly changing it may have a weaker accuracy. The third category of intrusion detection is SPA, which is similar to AD but depends on vendor-developed generic profiles. These are usually based on protocol standards from international standard organizations and focus on unknown attacks. The downside is though, that it won't be able to see attacks that seem like harmless protocol behaviours, and it might be harder to find compatible SPAs for dedicated operating systems. (Liao et al. 2013)

The importance of software testing is largely shown through projects with insufficient testing and sometimes catastrophic results. Though testing is crucial and a fundamental part, there is a balance to finding the optimal test effort since costs rise with testing, but the quality does not necessarily continue growing (Anwar and Kar May 2019).

Enhancement in the testing process can be done through test automation, testing frameworks in agile, and test-driven development. There is also a plethora of testing metrics that can be used like prioritization metrics and process quality metrics (Jamil et al. 2016).

2.2 Software Containers

Unlike the process of software testing, software containers are more recent and relatively unfamiliar to many. In the technological definition, containers are simply something that can hold applications, as can be concluded from their name. Providing applications with an environment to carry out their dependencies and simultaneously isolating them from unconnected programs and the underlying operating system (OS). In other words, it can be said to be an application packaging unit containing only its dependencies, and therefore comparable to the containers in the shipping industry where pre-built containers are used to store and ship products. (Siddiqui, Siddiqui, and Khan 2019)

With the growth of technology in a relatively brief period, the industry has evolved quite a bit. First from native to virtualized, then to the cloud, and now to software container technology. The evolution is focused on creating software that is increasingly lean and agile (Siddiqui, Siddiqui, and Khan 2019). Software containers allow the user to package applications with all their required dependencies (for instance software, configurations, libraries, frameworks,

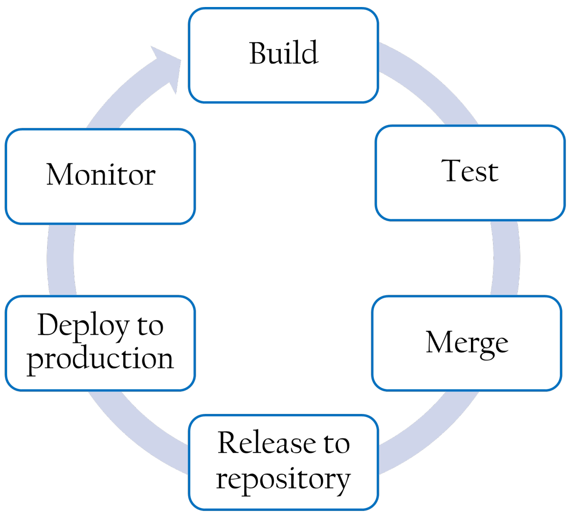

Figure 1. CI/CD cycle.

and binaries) (C.-C. Chen et al. 2022). Software containers are sometimes also referred to as operating system-level virtualization or lightweight virtualization (Sultan, Ahmad, and Dimitriou 2019).

The need for containers surfaced due to developers having the sometimes-discouraging task of shifting applications between different environments, like from development to testing and ultimately to production environments. Migration issues can easily arise during these shifts since the environments can be dissimilar in both hardware and software. In order to overcome these obstacles and bring further efficiency, software containers are in use. (Siddiqui, Siddiqui, and Khan 2019)

Common hindrance developers face is how to release new code quickly yet safely (Leszko 2022). The practice of Continuous Integration and Continuous Delivery (CI/CD) has become well-known in DevOps, ensuring the fast delivery of new code, by automatically testing and releasing software versions (Rangnau et al. 2020). This produces many advantages like fast delivery with faster release cycles, fast feedback, low-risk release, flexible release options, early discovery, and increased productivity (Zampetti et al. 2021; Leszko 2022). The CI/CD cycle is shown in figure 1.

Continuous Integration is usually the starting phase, ensuring that code from all developers is integrated. This brings the first step of feedback by compiling the new code, running automated tests on it, and verifying its quality. The Continuous Delivery part, sometimes worded as Continuous Deployment, refers to the stages that come after CI in the CI/CD pipeline. This is when the now-checked code is merged into a shared repository and then deployed into production. (Leszko 2022)

Docker is one of the key components for deployment using CI/CD pipelines (Abhishek and Rajeswara Rao 2021). Containers can be deployed using Docker as a part of the CI/CD pipeline using Jenkin Server (Abhishek, Rao, and Subrahmanyam 2022) and for example, scanning containers for known vulnerabilities is a crucial activity of the CI/CD process (Berkovich, Kam, and Wurster August 2020).

The CI/CD practice works as a kind of bridge between testing and containers, ensuring that everything works seamlessly. This has resulted in a type of new operating model in which testing has to be rethought and largely automated. While the more abstract concept is relatively well understood, CI/CD implementations differ widely (Mahboob and Coffman 2021).

Software containers are also commonly used to support microservices which are an architectural approach, that allows an application to encompass many independently operating components (Jamshidi et al. 2018). In addition to container management systems like Docker, orchestration systems like Kubernetes can be used to control applications and provision resources, resulting in scalable, reliable, and reactive systems (Douglis and Nieh 2019).

When talking about software containers, the contrast between them and traditional virtual machines should be pointed out. IEEE Internet Computing published a special issue on virtualization in 2013, at which point virtual machines had already become popular. They provide a virtual as opposed to a physical version of a resource, and a convenient way to encapsulate state and deploy services predictably. This helped in plenty of issues found, for example, efficiency, security, high availability, elasticity, mobility, and scalability. The key differences between containers and virtual machines are shown in figure 2. (Douglis and Krieger 2013)

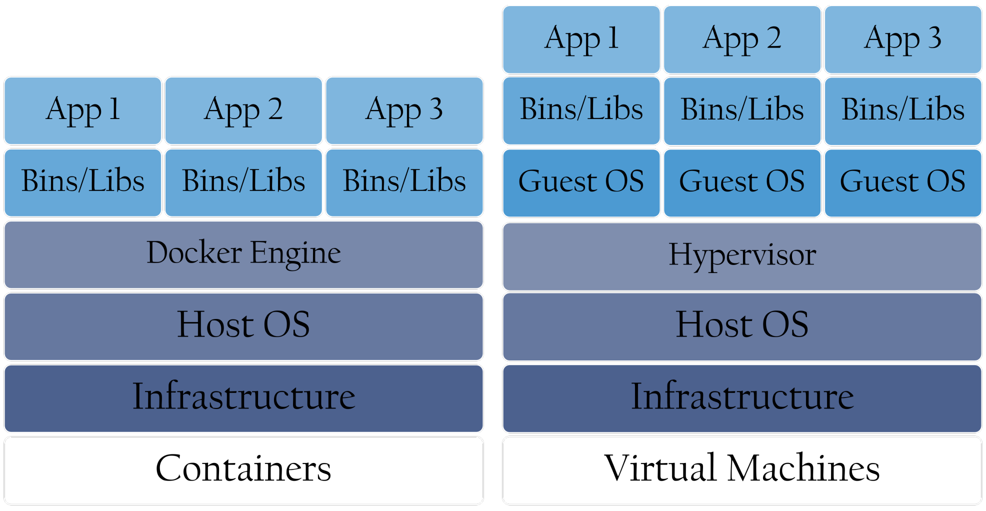

Figure 2. Containers vs Virtual Machines.

The key level of virtualization in a virtual machine is the virtualization of the underlying hardware. Software containers, on the other hand, aim to virtualize and isolate the operating system itself, therefore giving each containerized application a separate area of execution within the operating system. Containers allow multiple applications to run in one operating system, without the need to run multiple operating system instances and can therefore be more lightweight and potentially more easily managed. (Siddiqui, Siddiqui, and Khan 2019; Douglis and Nieh 2019)

The idea of software containers is in no way new or recent but can be traced to 1979 and the Unix V7 operating system and the introduction of the chroot system call. Various isolated implementations have come about of the software container logic, but a more comprehensive implementation was LXC by Linux in 2008. Nowadays Docker is the most widely spread and leading container technology provider. Docker was founded in 2013 and started by using LXC but has now replaced it with its own library and offers a complete ecosystem around containers, including but not limited to managers and orchestrators. (Siddiqui, Siddiqui, and Khan 2019)

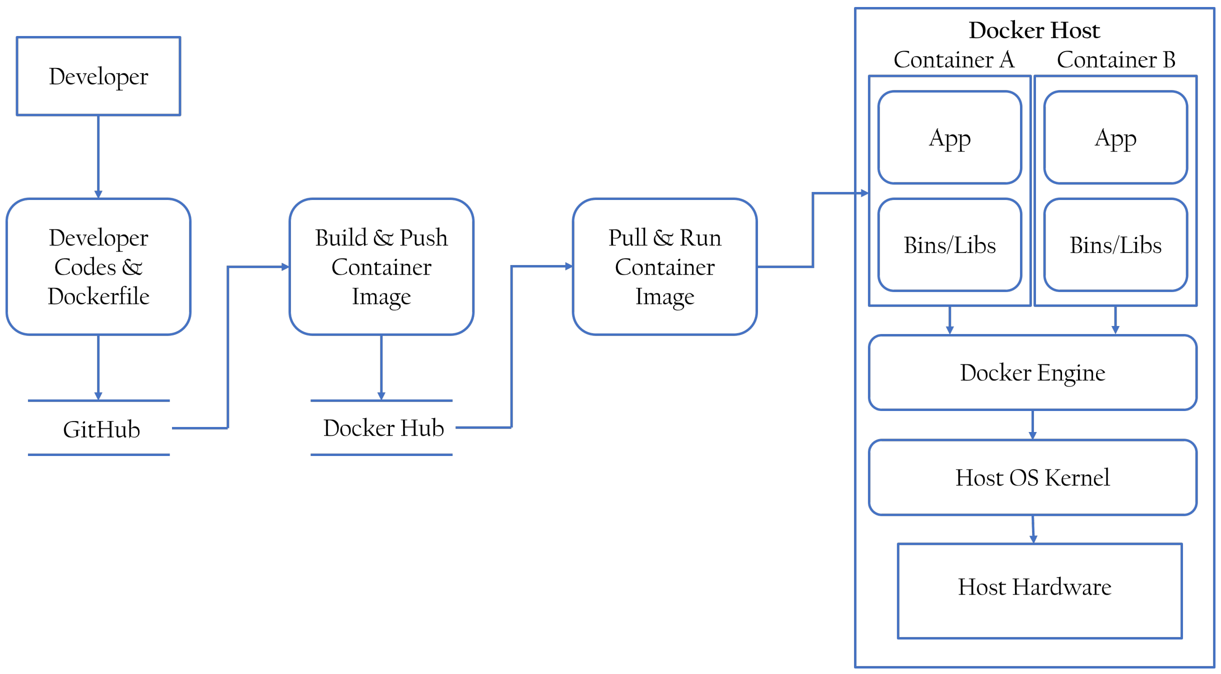

Figure 3. Overview of the container process (adapted from Wong et al. 2021).

The current view of the software container ecosystem includes building blocks like the container platform, orchestrator, repositories, and container as a service. Container platforms help in packaging applications with pre-requisite dependencies, with current provider examples being Docker, Kubernetes, Mesos, and Vagrant. The container orchestrator helps in the lifecycle management of the software within containers, like Kubernetes, Docker Swarm, Mesosphere Marathon, and Nomad. Repositories have collections of container images that can be found, like Docker Hub and Amazon ECR. Lastly, container as a service (CaaS) is a service providing ready-to-use container technology, with current parties being Amazon Elastic Container Service, Microsoft Azure Container Instances, Google Kubernetes Engine, and IBM Cloud Kubernetes Service. (Siddiqui, Siddiqui, and Khan 2019; Cito et al. 2017)

When using software containers, the ecosystem typically includes a repository (for building container images from Docker files consisting of the code and libraries) and an image registry (where the image is pushed for successive deployment as containers). An overview of the container system and the container process is shown in figure 3.

This ecosystem can lead to ample possible security breaches (like stealing data, conducting denial of service attacks, and gaining root access for misuse to name a few). Research shows that threats to containers include spoofing, tampering, repudiation, information disclosure, denial of service (DoS), and elevation of privilege. (Wong et al. 2021)

Docker hub is one of the most popular Docker image repositories, distributing both official and community images. There have been multiple studies into the state of security vulnerabilities in these images, largely due to some reported high-profile attacks done to this channel of image distribution. One such study was conducted in 2017 (Shu, Gu, and Enck 2017) where a Docker image vulnerability analysis framework (DIVA) was created. DIVA automatically discovered, downloaded, and analyzed container images found from Docker Hub. The study went through 356,218 images and concluded that: there is a great extent of vulnerabilities on average; images were not being updated for hundreds of days; and these vulnerabilities commonly spread to the child image from the parent. They also call attention to the need for more automated and systematic procedures to apply security updates to Docker images.

Another paper containing an in-depth security analysis of Docker images on Docker Hub was done in 2020 also showing the need for vast improvement (Liu et al. 2020). It identified the three major sources of security risks to be: sensitive parameters in run commands, malicious Docker images, and vulnerabilities in contained software that were left unpatched.

One of the first studies into Docker image security was done by inspecting images from Docker Hub with an open-source tool *Banyan Collector* they created (Gummaraju, Desikan, and Turner 2015). This study highlights that over 30 percent of the images in official repositories contained vulnerabilities.

According to a 2018 article measuring Linux container security (Lin et al. 2018), in which a container security measurement study was conducted using an attack dataset that contained 223 exploits, containers "[do] not provide much security enhancement for the programs inside". They continue to conclude that while the security of a software container depends on the security of the kernel, the interdependence and give-and-take relationship demands meticulous configurations to successfully thwart privilege escalation attacks. Another more recent article from 2020 (Kwon and Lee 2020) states that Docker does not provide security assurances for recognized security vulnerabilities inside of the container images.

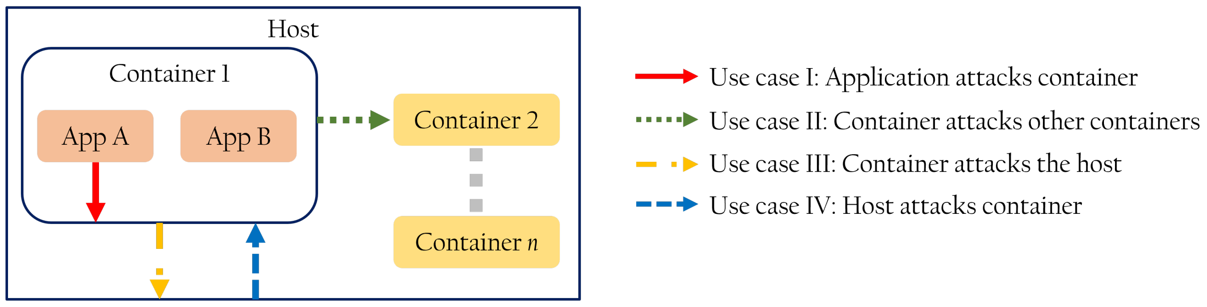
Host
Container 1
App A
App B
Container 2
Container *n*
Use case I: Application attacks container
Use case II: Container attacks other containers
Use case III: Container attacks the host
Use case IV: Host attacks container

Figure 4. Overview of security protection requirements in containers (adapted from Sultan, Ahmad, and Dimitriou 2019).

To combat these growing issues, an article from 2019 (Sultan, Ahmad, and Dimitriou 2019) derived four generalized use cases for the host-container level in order to identify threats, suggesting also viable solutions. These can be seen in figure 4.

The first use case is protecting a container from its interior applications, the second was inter-container protection, the third protecting the host itself from containers, and lastly protecting the containers from the host. The article delves into a plethora of possible attack scenarios for these use cases and also suggests workable solutions. The attack scenarios for use-case I include ones based on image vulnerabilities, configuration defects, embedded malware or secrets, using untrusted images, privilege escalation, and DoS. Attack scenarios in use-case II relate to untrusted images, application vulnerabilities, poorly separated inter-container traffic, insecure runtime, and unbounded network access from containers. The third use-case with threats being host OS attack surface, containers sharing host OS kernels, resources accountability, and host filesystems being tampered with include possible attack scenarios like container escape attacks, DoS and tampering.

2.3 Securing Software Containers

The article from 2019 (Sultan, Ahmad, and Dimitriou 2019) also mentions possible solutions to the various attack scenarios shown in figure 4. Use-case I solutions revolve around monitoring, scanning, running the applications with the least privilege needed, storing secrets outside the image, and updating periodically. Solutions for use-case II involve additional scanning, restricting communication, separation of containers into virtual networks, and updates. Use-case III includes solutions like using a container-specific OS, scanning, and running containers with minimal sets of permissions.

Existing mitigation strategies, and the limitations they may have, are also discussed in a study from 2021 (Wong et al. 2021). The first mitigation strategy they overview is multi-factor authentication systems. They state multiple disadvantages like increased time and cost, and the ability of attackers to perform a Short Message Service (SMS) attack on a compromised phone to get verification codes that are not encrypted. The next mitigation tactic looked through was those relating to image security, for example reducing attack surfaces, signing images, and vulnerability scanning. Limitations mentioned for these include the possibility of stolen private keys, issues in the quality of container scanning, and the need to use various tools to perform all needed scans. For example, scanners depend on Common Vulnerabilities and Exposures (CVE) data from public databases and can only detect publicly disclosed flaws. The accuracy of available container scanning tools was investigated in another article (Javed and Toor 2021) that also concludes that the OS is the more vulnerable part of the container.

Security patching is another mitigation strategy introduced in the study by Wong et al, though the author also states that a reliable and fast patching framework is a current gap in research that should be addressed. Minimizing administrative privileges is also a way to better secure a software container, and there are a few developed methods made for this that raise alerts when detecting anomalies (Lin, Tunde-Onadele, and Gu 2020; Kang, Fuller, and Honavar 2005). This can be done by limiting docker run options, hardening host configurations, limiting file permissions, configuring Transport Layer Security (TLS), or even running a container "rootless". (Wong et al. 2021)

Proper isolation can also be done as a mitigation strategy, including Linux kernel Cgroup which controls and limits the host resources for the container (Wong et al. 2021). Though another study states that kernel security mechanisms (for example Capability, Seccomp, and MAC) have a supreme role than that of container isolation (for example Cgroup and Namespace) in preventing privilege escalation attacks (Lin et al. 2018). An approach to sandbox mining and sandbox enforcing was presented in another article where after exploring containers through test cases, the mined sandbox confines and restricts a container's access to system calls (Wan et al. 2019).

To prevent DNS spoofing attacks the implementation of network controls is additionally important. For example, it is recommended that the use of Docker's default bridge docker0 should be stopped, and instead, the use of a user-defined network is preferred. Robust log monitoring is also beneficial, like the use of MACs (message authentication codes). It is also a good tactic to prevent confidential data leaks by not storing unencrypted secrets in repositories, though not impenetrable. (Wong et al. 2021)

Other mitigation strategies and precautions to take include implementing powerful access controls and keeping containers lightweight (Efe, Aslan, and Kara 2020).

Although all these mentioned mitigation strategies are important stepping stones to a more secure software container, other than scanning and monitoring, they are all mostly preventive approaches. Furthermore, they can be done wrong, possibly giving a false sense of security. Additionally, to these mitigation strategies, software container security frameworks and testing approaches are looked into in the next section.

2.4 Testing Software Containers

An example of anomaly-based detection in software containers was already briefly mentioned in section 2.3 when discussing the minimization of administrative privileges. This was a reference to an article from 2020 (Lin, Tunde-Onadele, and Gu 2020) where Classified Distributed Learning (CDL) was implemented as a prototype to detect security attacks in containerized applications. CDL carries out a distributed learning framework in order to overcome challenges due to insufficient training data in ephemeral container environments, aiming to create sturdy normal behaviour models. Evaluations were made in real-world vulnerability attacks in frequently used server applications. These experimental results show not only a sizeable reduction in false positives but also an improvement in attack detection.

A different approach to testing software containers was introduced in an article from 2020 which presents a proposal for Docker Image Vulnerability Diagnostics System (DIVDS)

(Kwon and Lee 2020). The intended use of the DIVDS is to diagnose Docker images when one is uploaded or downloaded from an image repository. This is because Docker images are currently distributed without any vulnerability diagnostics, which in turn can result in collapsed environments due to polluted images.

Another proposal was introduced in 2022 with the SEAF framework (L. Chen et al. 2022). SEAF stands for Scalable, Efficient, and Application-independent Framework which is to be used for container security detection. They propose this static detection method, which inspects various security defects and evaluates their impacts. It utilizes a Global Relationship Tree-based analysis and open-source modules for anomaly detection. The article concludes by evaluating popular repositories and finding more than 35 000 security defects in categories of misconfigurations, unauthorized access, CoinMiner malware, sensitive information leakage, and software vulnerabilities. It is also stated that since the detection rules are based on known threat models, SEAF cannot reveal threats before they are disclosed.

There are a plethora of other similar frameworks and patterns for securing software containers like the *Secure Container Manager Pattern* (Syed and Fernandez 2020) which focuses on the use of container managers for orchestrating container clusters, and the *Framework to Secure Docker Containers* (Abhishek and Rajeswara Rao 2021) which looks into container image security in cloud computing.

Another notable framework is *CONSERVE: a framework for the selection techniques for monitoring container security* (Jolak et al. 2022). After reviewing different container monitoring techniques, it gives a "multi-dimensional decision support framework for an informed and optimal selection of a suitable set of container monitoring techniques to be implemented in different application domains".

A conference paper from 2021 (Sidqui and Siddiqui 2021) points out that testers are at the receiving end in which no global standards yet exist for the non-functional part of applications. The same authors later wrote the *Non-Functional Testing Framework for Container-Based Application* (Siddiqui and Siddiqui 2021). It takes a deeper dive into this with the objective of having an inclusive framework that could assist in defining the process and an eco-system which could be used to test the non-functional behaviour of container applications.

According to the authors (Siddiqui and Siddiqui 2021) deciding factors for an application's success come from testing results. They state that the evolution of containers has mainly been driven by industry adaptation rather than academic research. This is especially true on the testing side because testing frameworks can be found for containerized platforms, rather than for container-based applications. And specifically to these container-based applications the need for non-functional testing grows critical due to the application being oblivious to the underlying hardware and middleware. Though one of the same authors did a review for testing approaches, it was restricted to cloud-based applications (Siddiqui and Ahmad 2016).

A blog post was written concerning the differences between testing cloud-native and on-premise testing (Patrizio 2020). It articulates that testing a collection of resources (like dozens of containerized applications) means the need for testing each application against every resource as well as each other. For example in a case where multiple containers use the same database, it is not evident that a problem doesn't occur when containers 1 and 2 simultaneously use it, even if containers 1 and 3 work together without performance problems. To lessen the impact of all these unknowns, Siddiqui et. al. define four non-functional attributes characterizing the container application's surrounding environment and its behaviour: capacity, scalability, stability, and high availability (Siddiqui and Siddiqui 2021).

Other than issue mitigation strategies as mentioned in section 2.3, and further monitoring and diagnostics systems and frameworks introduced in this section, information on testing specifically software containers are scarce. That, in particular, is precisely what should be scrutinized, and what this thesis delves into.

3 Research Methodology

This chapter starts first with an overview of the research background and context in section 3.1, then the research question and hypothesis in section 3.2, after which the research method is explained in section 3.3. The construction of the survey 3.4, response collection 3.5 and the method for data evaluation 3.6 are then described in the stated sections.

3.1 Research Background and Context

This thesis is conducted as part of a larger software container project: Containers as the Quantum Leap in Software Development (QLeap). The project QLeap is funded by Business Finland and led by the University of Jyväskylä aiming to investigate the full lifecycle of containers as the quantum leap in software development.

The goals of the project are to bring together researchers and industry experts from different organizations to facilitate industry-academia collaboration, and to research problems that emerge from use cases and development approaches from industrial partners' business needs.

The topic for this thesis is not only scientifically pertinent due to the growing interest and use of software containers with more and more connected issues frequently emerging, but in addition, it is a practically relevant topic due to being conducted as part of the project QLeap.

The specific topic of testing software containers arose from the partaking companies' needs and distinguished use cases.

3.2 Research Question and Hypothesis

The main objective of this thesis is to take a look at the current status of company practices and distinguishing tools concerning how anomalies can be detected through testing software containers.

The encapsulated research question for this thesis is the following:

RQ: What is the status of how companies test software containers?

Based on this research question, the aim is to delve into different areas of testing software containers, which are also represented in the survey conducted:

- the first section is about tools used in testing,
- the second section is the process and practices of testing,
- the third section is the results and challenges of testing,
- the fourth and last section is about goals and future implementation plans.

Based on the research findings described in chapter 2, there does not seem to be a consensus on how testing software containers should be accomplished, and advances are mainly being driven within companies instead of academic research, and in numerous diverse ways. The hypothesis is that the survey answers between companies will be vastly different, with varying levels of testing, and that there might not even be a clear method for specifying testing to the software containers.

The research detailed in this thesis is hoped to provide a glance at the present status of company practices, and additionally show via the survey results possible aspects that require additional research.

3.3 Research Method

The research process started with reviewing literature that is available on the topic of testing software containers. A survey was then created in order to ask the prevalent questions to answer the research question and the varying topics described in the previous section.

A survey is a structured data collection method based on the creation of a form with questions to which respondents provide answers based on their knowledge, expertise, and experience. It is a common data-gathering process allowing access to information from a group of respondents during research. Surveys are relatively flexible and are therefore used in many diverse types of research, like planning and evaluation. It can be either qualitative or quantitative depending on the type of research and wanted data.

The idea of conducting the research as a survey first came from the companies involved and was decided upon as the research strategy with the distribution being completely online after some thought. Though a survey is a fundamental research strategy used to gather information, it nevertheless still has its difficulties, and it is easier to create a poor-quality survey than one with real value (Kelley et al. May 2003).

There are many steps to think through before starting the creation of a survey. Firstly, there must be a clear research goal. After this, the method for collecting data and the type of questions asked need to be examined. The questions themselves need to be designed so that they answer the research question, in a clear, concise, and unbiased way. After the survey is created, it can be distributed, and responses analyzed. (Jones, Baxter, and Khanduja 2013)

The main pros to a survey, in this case, were the opinion of the companies involved, giving time to find the appropriate respondents, giving them time to look through and think about the questions before answering them, not limiting time for answering, and the easy online distribution.

On the other hand, the main cons for a survey, in this case, were possible interpretations of the questions, creating questions that were not directing the respondent but still being able to get clear answers, and mainly getting people to go through and answer the survey.

Most of these cons were addressed with thorough consideration of the survey questions that peers also went through to mitigate interpretations, and providing an email address through which questions could be asked. Additionally asking questions from different approaches and reaching out to multiple people for responses and sending reminders for answering the survey.

3.4 Survey Structure and Construction

The survey was created with Webropol 3.0, a survey and reporting tool that the university offers use of licenses for students and staff. Webropol makes online survey creation simple and offers help and analysis tools. It also handles how the material is collected, processed, stored, and archived/destroyed. Other helpful tools Webropol offers are emails that can be sent based on triggers (e.g., receiving an answer to a survey), adding a contact form to the survey, and establishing rules for some questions (e.g., having some questions be mandatory, or giving preset options).

Though Webropol provides a service through which it is possible to make, send, collect, analyze, and delete data in a controlled manner, the questions asked, and their formatting needed a lot of attention. A balance had to be obtained between a precise enough question to get relevant results, but not too broad or biased which might result in interpretations based on the respondents' understanding.

The survey structure has five main sections, prefaced by an introduction, preceded by a contact form, and followed by a place to add any additional comments or questions the respondent may have. With a total of twenty-three questions, including the contact form and additional comments, the contact form was the only mandatory question that required an answer before being able to submit the form, which asked for the respondent's email address and company name (used for sorting the answers).

The first section of the survey is the *Demographics section* with a total of four questions, two of which are multiple choice and two open-answer questions. The importance of this section is to understand the respondents' roles and experiences in the companies, regarding containers. These questions are listed in table 1.

Table 1. Survey questions: Demographics section.

No.	**Question**	**Type of answer**
2.	What sort of software/system is being containerized in the company?	Open answer
3.	What are your major responsibilities in the company?	Open answer
4.	How many years have you been working in the IT industry?	0-2 / 3-5 / 6-10 / 11-15 / >15 years
5.	How many years have you been working with software containers?	0-2 / 3-5 / 6-10 / >10 years

Table 2. Survey questions: Tools used in the company.

No.	Question	Type of answer
6.	What containerization platform(s) is in use in the company?	Selection from list
7.	What tools to test container-based applications are used in the company?	Open answer

The second section is *Tools used in the company* with just two questions about the tools used in the company, the first of which is answered by choosing answers from a list, with the possibility of also adding one's own answer so as not to miss anything. These questions are listed in table 2.

The third section *Testing software containers* goes through the process, practices, and plans for anomaly detection through testing software containers via eight questions, three of which are multiple choice questions, one broader open-answer question, and five more specified ones. This is one of the most important sections of the survey since it directly answers topics related to the research question. The motivation behind the questions for this section is to hone in on and encapsulate the process of testing containers, in order to not only gauge how it is done but also see how it compares between various companies. The questions for this section are listed in table 3.

Table 3. Survey questions: Testing software containers.

No.	Question	Type of answer
8.	What is the process of testing containers in the company?	Open answer
9.	Is there testing done before containerization of the software?	Yes / No / Not sure
10.	Is there testing done after containerization of the software?	Yes / No / Not sure
11.	Is the testing specified to the container itself or the containerized content?	Container itself / Containerized content / Both / Not sure
12.	What practices and techniques do you use to make containerized software more testable?	Open answer
13.	What type(s) of testing is done to containers and containerized software?	Open answer
14.	Are there plans to implement other types of testing, and if so, what?	Open answer
15.	What automated testing strategies do you use to test containers?	Open answer

The fourth section *Results of testing* focuses on results, and challenges, and specifically aims to find out whether containerization itself affects test results. This section has three open-answer questions, listed in table 4.

The fifth and last section *Examples* asks for more concrete findings and potential future implementation plans with four open-answer questions. These questions are listed in table 5.

Lastly, the survey has an *Additional comments* part where any comments, questions or clarifications can be added if wanted.

Table 4. Survey questions: Results of testing.

No.	Question	Type of answer
16.	What differences, if any, have been found through testing software before and after containerization?	Open answer
17.	Have you found that the act of containerization affects test results, if so, how?	Open answer
18.	What challenges do you face when testing containers?	Open answer

Table 5. Survey questions: Examples.

No.	Question	Type of answer
19.	What anomalies are you looking to find through testing containers and containerized software?	Open answer
20.	What sort of anomalies have been detected through testing containers and containerized software?	Open answer
21.	How are you planning to improve anomaly detection and accuracy?	Open answer
22.	What are aspects that you have not found through testing/are not testing for, but hope to implement in the future?	Open answer

3.5 Participants and Data Collection

As mentioned in the previous section, the survey was constructed by utilizing the tool Webropol. In addition to providing a way to create surveys, it also allows the user to easily distribute the survey. The way it was done for this thesis, was a public link which was then emailed to the various companies part of the QLeap project as well as posted on LinkedIn to reach further interested companies. On the back end of Webropol, it was also possible to see the number of people opening the survey, starting to respond, and completing the survey.

The aim for survey answers was to get answers from multiple companies, hopefully from people with varying work profiles and particularly those who work closely with containers: testers, software developers, project managers etc. In some cases, multiple people from the same company responded giving a more complete picture of that company, which could then be compared to the testing approaches of other companies. This information was also provided to the contact people from each company, to whom the survey link was sent, who in turn would proceed to identify the people to respond within their companies.

In total there were six companies taking part in the QLeap project that were creating software and utilizing containers, and thus potential companies to answer the survey. In addition to these six, other companies were reached via a LinkedIn post asking for respondents interested in the topic. The participating companies that answered the survey ranged in size, the software they make and the extent of container usage.

Limiting when the survey could not be answered or seen anymore, and the deletion of data was also possible through Webropol, to manage the process well.

3.6 Data Evaluation Process

The responses for surveys done with Webropol are visible to the creator through their online website, where the survey is also created. The data for each question is displayed individually, with some ready-made graphs provided for multiple-choice questions. Webropol also allows for the creation of your own graphs, which helps comb through and summarize the results in a readable way.

The option of filtering answers was also necessary for the data evaluation, due to there being multiple answers from each company. Answers could be filtered to show only those answers where the answer for the company name in the contact form matched each other, therefore allowing the results to be evaluated per company instead of per individual respondent.

Some of the answers were multiple choice, where a more quantitative data evaluation approach could be taken. This includes bar graphs and comparisons of numbers. On the other hand, most questions in the survey were open-answer questions and a qualitative approach was taken, due to the nature of the more descriptive and conceptual answers. The answers were all read and summarized, and similarities were found. These were then charted in several ways.

4 Results

This chapter goes through all the results of the survey, divided into subsections the same way the survey was with section 4.1 going through the demographics section, sections 4.2 and 4.3 looking at the tools and the process of testing containers, section 4.4 focusing on the results of testing, and lastly section 4.5 showing the more concrete findings.

There were respondents from nine different companies with thirteen respondents in total ranging between one to three answers per company. The survey itself was opened by a bit over one hundred people according to the Webropol back-end statistics.

4.1 Demographics Section

Question 2. What sort of software is being containerized in the company?

The software being containerized ranged greatly between the companies. Mentioned projects related to things like Research and Development (R&D), cloud software, machine learning, value stream metrics, web applications, testing, back-end services, and development environments.

Question 3. What are your major responsibilities in the company?

The respondents also had various types of roles within the companies ranging from solution/enterprise/software architects, product owners, development/R&D team leads, scrum masters, software consultants, software testers, and back-end developers.

Question 4. How many years have you been working in the IT industry?

The majority of respondents had been working in the IT industry for over 15 years as illustrated in Figure 5.

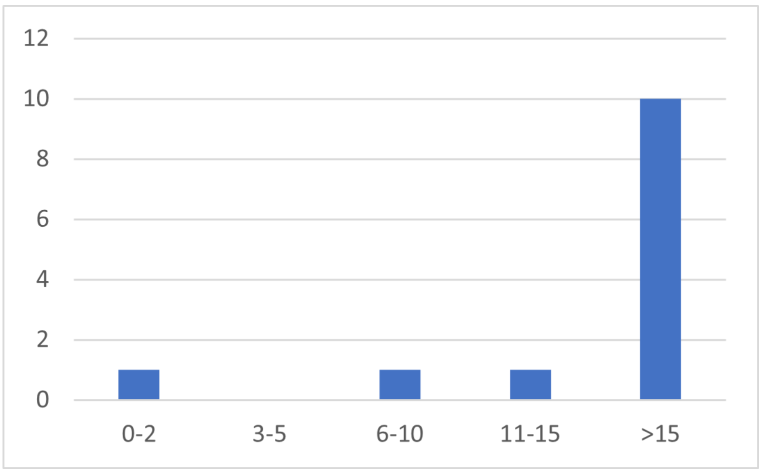

Figure 5. Answers to question 4: years worked in the IT industry.

Question 5. How many years have you been working with software containers?

Most respondents had been working with software containers for 6-10 years, though the varying amounts can be seen in Figure 6.

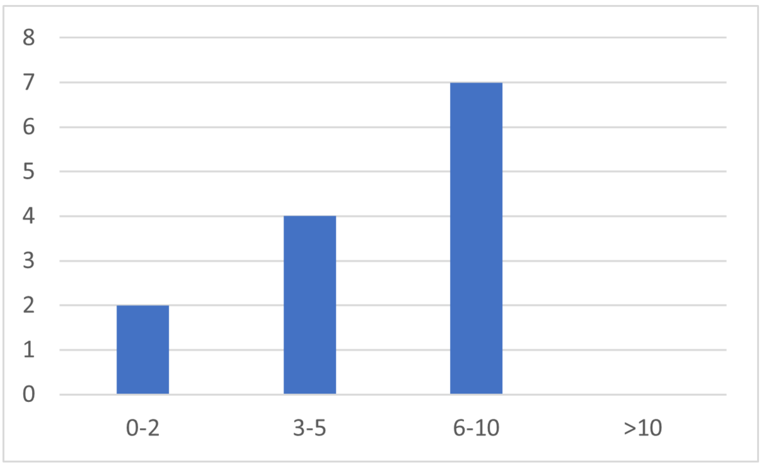

Figure 6. Answers to question 5: years worked with software containers.

4.2 Tools Used in the Company

Question 6. What containerization platform(s) is in use in the company?

Docker was the most used container platform used by the different companies with eight out of nine companies having used it as shown in Figure 7.

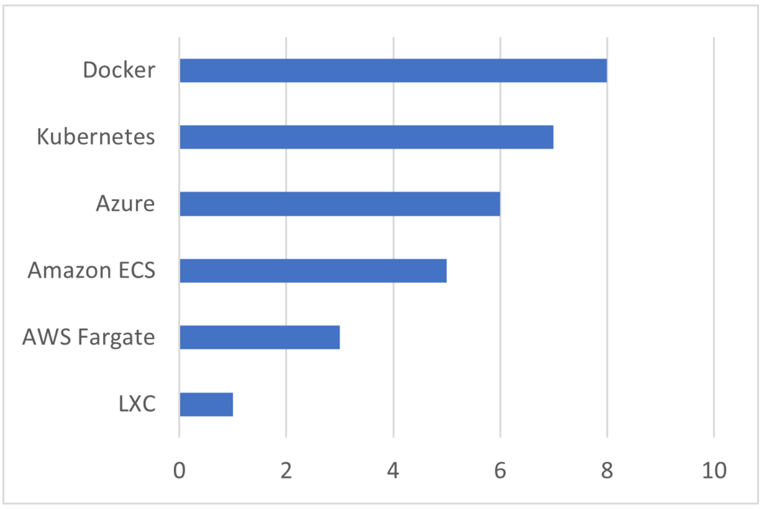

Figure 7. Answers to question 6: containerization platforms used in the company.

Other containerization platforms mentioned were Nomad, OCI, Rancher (RKE), open shift, k3s, and Podman.

Question 7. What tools do you use to test container-based applications?

Tools used to test container-based applications varied, with specific tools mentioned being JUnit, TestContainers (Java), Docker Desktop and tools, minikube, Robot framework, Spira, sonarqube, trivy, selenium, protractor, and jest. Specific test frameworks mentioned were pact.io and cucumber. There was also mention of in-house test platforms, automated Continuous Integration and Continuous Delivery (CI/CD) pipeline tests, and container checker tools.

In addition to specific tools used for testing, what also came across from the answers was how dependent the tools used are on the software language, the technology stack used, and even just which team in the company is being asked.

4.3 Testing Software Containers

Question 8. What is the process of testing containers in the company?

The answers to question eight had some commonalities but mostly varied, also within the companies depending on the respondent and their role.

Listed here are some examples of the testing processes mentioned:

- Github test cases, test containers for Java, CI/CD tools, and robot framework.
- Code changes run through with robot tests in an environment that replicates a real container environment. Afterwards, when tests pass and a release candidate (rc) version is created the rc is deployed on the development environment to be tested in an environment replicating the production environment. The tested container is deployed to the staging environment simulating deployment to production and verification testing is done before the container is cleared for release to production.
- During development testing occurs locally on the developer machines, and then the work is reviewed and automatically tested in the company's CI/CD platform.
- Many different release trains and CI/CD pipelines with shifting maturity depending on the offering technology stack. Unit tests, Application Programming Interface (API) tests, and integration tests are a development responsibility and should be done regardless of whether containers are used or not. The same goes for static code analysis. All of these quality-assuring steps can be achieved either manually or automatically as steps in a CI/CD pipeline. What differs with containers is that we usually include automatic steps in the CI/CD pipeline for continuous vulnerability scans, one of our more modern solutions use harbor and trivy which continuously and automatically scan all pushed container images for known CSV:s. A CI/CD pipeline can also automatically spin up, execute and close down a composition of multiple containers, which is very useful when you have many microservices and want to test a full deployment of your services.
- Containers are used for running full development environments and allow to run complete environments for integration tests.
- Unit tests, functional tests of inputs/outputs of containers, per container and in multi-

container end-to-end setups. Reliability tests of running without restarts and monitoring. Security and open-source license scans.

- Most of the testing is done as integration testing (back-end and database) before containerization. Front-ends and suitable parts of back-ends are tested with some additional unit tests. Some end-to-end (E2E) tests may be used in some projects, which test some limited functionality against a relatively complete test environment, which typically runs on containerized software (SW) in the cloud, but they are relatively rare.

Other things mentioned were specifically using docker unittest, container structure tests, and manual functional testing (for software inside the container).

Questions 9-10. Is there testing done before/after containerization of the software?

For both questions nine and ten, eight out of the nine companies chose "yes", with one answer being "not sure".

Question 11. Is the testing specified to the container itself or the containerized content?)

The responses for question 11 are illustrated in Figure 8 with the answers being per respondent instead of per company since the answers varied also inside some of the companies.

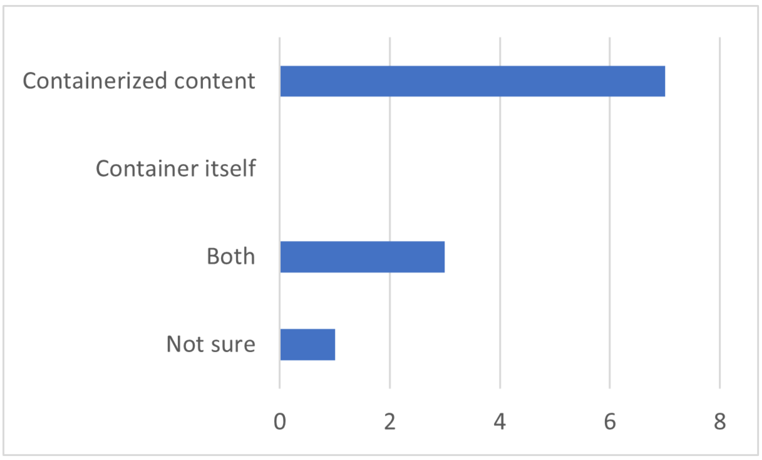

Figure 8. Answers to question 11: testing specified to the container itself versus the containerized content.

Most answers state that the testing is specified to the containerized content and not the container itself.

Question 12. What practices and techniques do you use to make containerized software more testable?

Answers for question 12 included:

- Having containers run in a local development environment for easier root cause analysis and deployment of testing,
- Loose coupling,
- Abstraction,
- Unit testing,
- Docker exec commands to examine the output,
- Allowing software to run in test mode that allows test versions of dependencies where you can set predefined or parameterized states (fulfilling preconditions),
- Clear network interfaces between containers,
- Container hardening,
- Log design.

Mentions back to question 8 were also made.

Question 13. What type(s) of testing is done to containers and containerized software?

Types of testing done to containers and containerized software specified were:

- User Interface (UI) and User Experience (UX) tests,
- Database tests,
- Logic tests,
- Security tests,
- Penetration tests,
- Testing for any misconfigured containers,
- Manual functional testing for the software,
- Unit tests,
- Functional testing,
- Exploratory testing (in development environment replicating production),
- Test automation checking the whole development environment and its User Interfaces and back-end services work together,
- Scalability and high availability (tested to be efficient and reliable),
- Deployment of new application versions (without affecting the user experience),
- Container structure tests,
- Integration tests,
- End-to-end testing,
- Reliability,
- Performance,
- Legal.

The role of the CI/CD pipeline was also commented on and as in question 12, some mentions back to question 8 were made here.

Question 14. Are there plans to implement other types of testing, and if so, what?

Plans for implementing further types of testing mentioned mostly related to automated tests (unit testing, security testing, integration tests, and Quality Assurance (QA) steps (static code analysis, vulnerability scans)). More "toolized" tests were also mentioned for vulnerability testing or testing supply chain attacks, as were the addition of more good practice scanning tools and running tests in a CI/CD in a container.

Question 15. What automated testing strategies do you use to test containers?

The answers to question 15 were mostly about the research of automated testing for containers at this stage, though some mentions were made to unit tests, integration tests, release testing, CI/CD pipelines for code changes, and MS Azure structure for staging and deployment.

4.4 Results of Testing

Question 16. What differences, if any, have been found through testing software before and after containerization?

There were two types of answers for question 16, the others being positive things and on the other hand negative aspects were also brought in. Some positives differences mentioned about testing software before versus after containerization were:

- Containers make automation easier,
- Consistency of the different software layers (stack), even driver level is easy to manage and handle,
- It makes it easier to run the full system and run end-to-end tests,
- Environments are more repeatable,
- Production-like environments on development machines are more likely,
- Each branch can have its own isolated environment making change testing more targeted.

As for the negative aspects, it was mentioned that error situation analysis requires more work if the logging of the systems is not thought out fully before testing.

There were also mentions of not seeing differences or not knowing in cases where the software has always been containerized since the beginning.

Question 17. Have you found that the act of containerization affects test results, if so, how?

The answers for question 17 varied, also among respondents within the same company. Multiple respondents stated that they have not found the act of containerization affected test results. There was mention of containerized testing providing more consistent results and that they are available faster with less painful maintenance. Multiple answers also mentioned containers enabling easier automation.

Another aspect introduced was the need to use different configurations for testing, and that the software environment might change how the software operates: if tested before containerization the software execution environment is different than in a container. If tested inside the container, then the test environment must be part of the container and it's not very clever to have the test environment in a real operation environment - so full testing in a real container is "impossible".

Question 18. What challenges do you face when testing containers?

Question 18 got quite a few answers, with most respondents choosing to answer it, many giving multiple answers and only two leaving it blank. Listed next are the answers:

- Test environment must be in the container,
- Compilation times are longer,
- If, e.g., a container is configured wrong, the network connections of the SW do not work,
- Creating an environment for them to be testable is sometimes challenging,
- The infrastructure to run containerized tests is not always easy to set up,
- Depending on the application architecture, especially with microservices you might

have dependencies to additional services to be able to run integration or E2E tests - but this is related to the microservice pattern rather than the software running in containers,
- Dynamic visualization of the results especially in multi-container environments,
- Synchronization between different containers was an issue at the start,
- Ramping up a system for tests is sometimes slow and requires extra focus on designing software so that you won't depend on the slow initialization of containers that one part depends on,
- Testers have had more trouble contributing as it adds a layer of operating their own environments,
- Testing containerized software is one thing and testing from a container is sometimes hard for people to understand as "more test environments at your fingertips",
- Heavily containerized SW may be difficult to test using test runners which itself run in containers, such as GitLab CI. Docker-in-docker helps, but it is itself a relatively complex piece of SW and makes it hard to understand how tests actually are run,
- Debugging tests are difficult if they only run in a complex environment with multiple containers.

4.5 Examples

Question 19. What anomalies are you looking to find through testing containers and containerized software?

Like question 18, this question got quite a few answers. It was communicated through multiple answers that any anomalies and errors are looked for and that the answer is the same whether the system is containerized or not. More specific anomalies brought up are listed:

- Regression issues,
- Performance down-grades,
- Load-related anomalies such as issues with session replication, bottlenecks, memory leaks,
- Vulnerabilities or increased technical dept through static code analysis,
- Long-term stable execution on different hardware with similar capabilities,

- Smooth dynamic scalability,
- Broken APIs between dependencies,
- Reconnect/disconnect logic issues,
- Software crash/hang discovery,
- Impact on functionality, reliability, performance, and supportability.

Question 20. What sort of anomalies have been detected through testing containers and containerized software?

Multiple answers to question 20 explained that there has not been much in particular, but those mentioned are listed here, many of which echo the answers to question 19:

- Errors,
- Regression issues,
- Performance down-grade,
- Session replication issues,
- For example the very critical log4j bug in 2021. (This was further explained to have been caught by automatic vulnerability scans and stating that achieving the same thing with manual work would have taken weeks with a high risk of missing out),
- Low-level drivers have created challenges example nvidia cuda driver, mysql driver, network drive access and network resource access,
- Broken APIs between dependencies,
- Reconnect/disconnect logic issues,
- Failed crash/hang discovery,
- A lot of the interface definitions and security hardening stuff are more in our control in containers and misconfigurations are more typical because of it. Or misconfigurations we get to fix and address ourselves.

Question 21. How are you planning to improve anomaly detection and accuracy?

A few answers for question 21 stated that there are no plans to improve anomaly detection and accuracy, but there were also a plethora of plans mentioned:

- Static code analysis,
- Vulnerability scanning,
- Putting test framework inside a container when testing,
- Writing better testing components that gather more data and are modular enough to be used in more than one test,
- By learning more about containerized testing and automation,
- Daily scans of your containers against csv-databases are a great way to automatically find out if your software is subject to new vulnerabilities,
- More testing in development and staging,
- Improve health check logic,
- Isolating changes is helping pinpoint the time when things broke down with changes.

Question 22. What are aspects that you have not found through testing/are not testing for but hope to implement in the future?

Unlike some of the previous questions, question 22 was only answered by less than half the respondents. The comments from those who answered are listed:

- The container as a one-test target is not very valuable, instead the integration test of the whole architecture in the real-run environment is very valuable,
- Automation of high-load of usage,
- We might further develop our ability to compare resource needs and performance through the comparison of metrics between releases in CI/CD execution,
- More static scans of container definitions,
- Dynamic behaviour in terms of efficient scalability,
- Stress testing.

5 Discussion

This chapter goes into more detail and discusses interpretations of the survey results in section 5.1, implications of the study in section 5.2, and finally limitations, weaknesses and suggestions for more research are discussed in section 5.3.

5.1 Interpretation of Results

The first section of the survey is the *Demographics section.* The importance of this section is to understand the respondents' roles and experiences in the companies, regarding containers. The answers showed the variety of the respondents' backgrounds and the software being containerized in the companies. It was also shown that the respondents have been working in the IT industry for a longer time, with most answers stating more than 15 years. The respondents' work with containers, on the other hand, is more varied with around half having worked with them for less than six years and a half for six or more.

The second section is *Tools used in the company* with just two questions about the tools used in the company. Docker was the most used, with eight out of nine companies having used it. This is unsurprising since Docker is the leading container technology provider and was founded in 2013 (Siddiqui, Siddiqui, and Khan 2019). Kubernetes and Azure were not far behind with seven and six companies utilizing them, respectively.

The third section *Testing software containers* goes through the process, practices, and plans for anomaly detection through testing software containers. Brought up in the 2021 article was how the evolution of containers has mainly been driven by industry adaptation rather than academic research (Siddiqui and Siddiqui 2021) discussed in section 2.4. This is shown in the answers to question eight, with the processes of testing containers between the companies varying greatly. There seems to not be any consensus on how testing software containers is or should be accomplished, correlating to the fact that one could not be found in the literature either. The hypothesis of the survey answers between companies being vastly different, with varying levels of testing, and there not being a clear method for specifying testing to the software containers held true.

There was a consensus, however, between respondents for questions nine through eleven, with most companies having testing done both before and after containerization and having that testing be specified to the containerized content.

The answers to questions 12 and 13 also brought some insight into the plethora of testing done on containerized software and techniques to make them more testable. Though software container security and vulnerabilities were not originally part of the scope of this thesis, they were mentioned in these survey answers and deemed important and relevant enough and therefore also brought into this thesis.

Question 14 was one of the most answered open answer questions with eleven out of thirteen choosing to answer. This seems telling as the question is about plans of implementing other types of testing. Many of the answers related to automated tests, which seem to be the next step for many companies. Question 15 expands on automated testing and shows that some companies have that in place, though statements were also made about the need for expansion.

The fourth section *Results of testing* focuses on results, and challenges, and specifically aims to find out whether containerization itself affects test results. The answers unveil differences found between testing software before and after containerization, with both positive and negative aspects brought up. Question 18 was a more crucial question with the answers displaying challenges faced when testing containers. This was one of the most answered open answer questions with eleven out of thirteen respondents choosing to answer. One of the main takeaways from the answers was the challenges in setting up an effective way to test containers. This can be linked again to the fact that there is no consensus on how testing software containers is or should be accomplished.

The fifth and last section *Examples* asks for more concrete findings and potential future implementation plans. Software container security and vulnerabilities were mentioned again, and emphasis was put on the need to test for all kinds of errors and anomalies, with examples of the ones found in the answers to question 20. A key point to bring up concerning the fifth section is the mention of plans to learn more about containers, testing them, and automation regarding them. The use of containers is on the rise and the development of good practices is still ongoing as clearly seen from the survey answers.

5.2 Implications of the Study

The main objective of this thesis is to take a look at the current status of company practices and distinguishing tools concerning how anomalies can be detected through testing software containers. The encapsulated research question for this thesis stated in section 3.2 is the following:

RQ: What is the status of how companies test software containers?

Based on previous research described, and the survey results, one of the key findings is that there does not seem to be a consensus on how testing software containers should be accomplished, and advances are mainly being driven within companies instead of academic research, and in numerous diverse ways. This is also something written about in the article *Non-Functional Testing Framework for Container-Based Applications* (Siddiqui and Siddiqui 2021).

The hypothesis of the survey answers between companies being vastly different, with varying levels of testing, and there not being a clear method for specifying testing to the software containers held true. An essential point to bring up again is that the use of containers is on the rise and the development of good practices is still ongoing as clearly seen from the survey answers, and the mention of plans to learn more about containers, testing them, and automation regarding them.

Another key point is that although software container security and vulnerabilities were not originally part of the scope of this thesis, they were mentioned in the survey answers and deemed important and relevant enough and therefore also brought into this thesis.

The research detailed in this thesis is hoped to provide a glance at the present status of company practices, and additionally show via the survey results possible aspects that require additional research. Analyzing the survey results discussed in this thesis shows both the importance of software testing and the potential gaps in research on security and vulnerability aspects in software containers and in the development of general good practices on how testing software containers should be accomplished.

5.3 Limitations and Weaknesses

The limitations and weaknesses of this survey and its results are predominantly related to the small size of respondents. It should also be remembered that the answers examined are the opinions of these respondents when thinking about the reliability of these results.

The very last part of the survey has an *Additional comments* section where any comments, questions or clarifications can be added if wanted. The answers mainly focused on the interest in the topic and results, but comments related to possible improvements included:

- The need for specifying where testing containers and where testing the software in the container is meant,
- The issue of asking for a large base of experience from people who often have single product experience and therefore answers depending greatly on even the team inside the company,
- Mention of the respondent themselves not working with containers too much or in a limited way.

These weaknesses, especially the first one, could be improved by further research such as structured thematic interviews, to collect and analyze the sought-after qualitative data. For this thesis, a survey was chosen over interviews due to the apprehension and recognition that it might result in even fewer respondents.

It is true that there is a notable imbalance in the respondents' answers due to the roles and in some cases limited use of containers. Despite this, the survey answers demonstrate uniformly how important testing truly is, the rise of interest and use of containers, security and vulnerability issues in software containers, the varying ways of testing containers and the gaps in current research. This research validates some observations from other research and makes a contribution by showing what is the status of how companies currently test software containers.

6 Conclusion

Testing is an integral and necessary part of software creation and is a crucial step to having a working and secure software container. Software containers are having a surge in use and are on a rising trend. Used more and more by most companies, risks and vulnerabilities related to them have continuously surfaced.

The research detailed in this thesis provides a look into the current status of company practices and distinguishing tools concerning how anomalies can be detected through testing software containers.

It can be concluded that the survey answers demonstrate how important testing truly is, the rise of interest and use of containers, security and vulnerability issues in software containers, and the varying ways of testing containers. There is no consensus on how testing is or should be accomplished, with advances being mainly driven within companies instead of academic research.

Further beneficial research in light of this thesis is the seen gaps in current research about security and vulnerability aspects in software containers and in the development of general good practices on how testing software containers should be accomplished.

In conclusion, this research validates some observations from other research and makes a contribution by showing what is the status of how companies currently test software containers.

Bibliography

Abhishek, Manish Kumar, and D. Rajeswara Rao. 2021. ”Framework to Secure Docker Containers”. In *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4),* 152–156. https://doi.org/10.1109/WorldS451998.2021.9514041.

Abhishek, Manish Kumar, D. Rajeswara Rao, and K. Subrahmanyam. 2022. ”Framework to Deploy Containers using Kubernetes and CI/CD Pipeline”. *International Journal of Advanced Computer Science and Applications* 13 (4). http://dx.doi.org/10.14569/IJACSA.2022.0130460.

Anwar, Nahid, and Susmita Kar. May 2019. ”Review Paper on Various Software Testing Techniques and Strategies”. *Global Journal of Computer Science and Technology* (): 43–49. https://doi.org/10.34257/GJCSTCVOL19IS2PG43.

Berkovich, Shay, Jeffrey Kam, and Glenn Wurster. August 2020. ”UBCIS: Ultimate Benchmark for Container Image Scanning”. In *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20).* USENIX Association. https://www.usenix.org/conference/cset20/presentation/berkovich.

Chandola, Varun, Arindam Banerjee, and Vipin Kumar. July 2009. ”Anomaly Detection: A Survey”. *ACM Comput. Surv.* 41 (). https://doi.org/10.1145/1541880.1541882.

Chen, Chao-Chun, Min-Hsiung Hung, Kuan-Chou Lai, and Yu-Chuan Lin. 2022. ”Docker and Kubernetes”. In *Industry 4.1: Intelligent Manufacturing with Zero Defects,* 169–213. https://doi.org/10.1002/9781119739920.ch5.

Chen, Libo, Yihang Xia, Zhenbang Ma, Ruijie Zhao, Yanhao Wang, Yue Liu, Wenqi Sun, and Zhi Xue. 2022. ”SEAF: A Scalable, Efficient, and Application-independent Framework for container security detection”. *Journal of Information Security and Applications* 71. ISSN: 2214-2126. https://doi.org/10.1016/j.jisa.2022.103351.

Efe, Ahmet, Ulaş Aslan, and Aytekin Mutlu Kara. 2020. Securing Vulnerabilities in Docker Images. *International Journal of Innovative Engineering Applications* 4 (1): 31–39. ISSN: 2587-1943. https://doi.org/10.46460/ijiea.617181.

Cito, Jürgen, Gerald Schermann, John Erik Wittern, Philipp Leitner, Sali Zumberi, and Harald C. Gall. 2017. ”An Empirical Analysis of the Docker Container Ecosystem on GitHub”. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR),* 323–333. https://doi.org/10.1109/MSR.2017.67.

CSRC. No date. ”Glossary”. Visited on March 16, 2023. https://csrc.nist.gov/glossary/term/software_vulnerability.

Douglis, Fred, and Orran Krieger. 2013. ”Virtualization”. *IEEE Internet Computing* (USA) 17 (2): 6–9. ISSN: 1089-7801. https://doi.org/10.1109/MIC.2013.42.

Douglis, Fred, and Jason Nieh. 2019. ”Microservices and Containers”. *IEEE Internet Computing* 23 (6): 5–6. https://doi.org/10.1109/MIC.2019.2955784.

Gaur, Jai, Akshita Goyal, Tanupriya Choudhury, and Sai Sabitha. 2016. ”A walk through of software testing techniques”. In *2016 International Conference System Modeling and Advancement in Research Trends (SMART),* 103–108. https://doi.org/10.1109/SYSMART.2016.7894499.

Gelperin, D., and B. Hetzel. 1988. ”The Growth of Software Testing”. *Commun. ACM* (New York, NY, USA) 31 (6): 687–695. ISSN: 0001-0782. https://doi.org/10.1145/62959.62965.

Gummaraju, J, T Desikan, and Y Turner. 2015. ”Over 30 percent of Official Images in Docker Hub Contain High Priority Security Vulnerabilities”. Visited on January 31, 2023. https://www.banyansecurity.io/blog/over-30-of-official-images-in-docker-hub-contain-high-priority-security-vulnerabilities/.

Haque, Mubin Ul, and Muhammad Ali Babar. 2021. ”Well Begun is Half Done: An Empirical Study of Exploitability and Impact of Base-Image Vulnerabilities”. *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER),* 1066–1077. https://www.semanticscholar.org/paper/Well-Begun-is-Half-Done%3A-An-Empirical-Study-of-%26-of-Haque-Babar/a8e1fd78284439f82d85f8119a638b458356b214.

Jamil, Muhammad Abid, Muhammad Arif, Normi Sham Awang Abubakar, and Akhlaq Ahmad. 2016. ”Software Testing Techniques: A Literature Review”. In *2016 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M),* 177–182. https://doi.org/10.1109/ICT4M.2016.045.

Jamshidi, Pooyan, Claus Pahl, Nabor C. Mendonça, James Lewis, and Stefan Tilkov. 2018. ”Microservices: The Journey So Far and Challenges Ahead”. *IEEE Software* 35 (3): 24–35. https://doi.org/10.1109/MS.2018.2141039.

Javed, Omar, and Salman Toor. 2021. ”Understanding the Quality of Container Security Vulnerability Detection Tools”. *CoRR* abs/2101.03844. arXiv: 2101.03844.

Jolak, Rodi, Thomas Rosenstatter, Mazen Mohamad, Kim Strandberg, Behrooz Sangchoolie, Nasser Nowdehi, and Riccardo Scandariato. 2022. ”CONSERVE: A framework for the selection of techniques for monitoring containers security”. *Journal of Systems and Software* 186:111158. ISSN: 0164-1212. https://doi.org/10.1016/j.jss.2021.111158.

Jones, TL, Maj Baxter, and V Khanduja. 2013. ”A quick guide to survey research”. *Annals of the Royal College of Surgeons of England* 95 (1): 5–7. https://doi.org/10.1308/003588413X13511609956372.

Kang, Dae-Ki, D. Fuller, and V. Honavar. 2005. ”Learning classifiers for misuse and anomaly detection using a bag of system calls representation”. In *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop,* 118–125. https://doi.org/10.1109/IAW.2005.1495942.

Kelley, Kate, Belinda Clark, Vivienne Brown, and John Sitzia. May 2003. ”Good practice in the conduct and reporting of survey research”. *International Journal for Quality in Health Care* 15, number 3 (): 261–266. ISSN: 1353-4505. https://doi.org/10.1093/intqhc/mzg031.

Kwon, Soonhong, and Jong-Hyouk Lee. 2020. ”DIVDS: Docker Image Vulnerability Diagnostic System”. *IEEE Access* 8:42666–42673. https://doi.org/10.1109/ACCESS.2020.2976874.

Leszko, R. 2022. *Continuous Delivery with Docker and Jenkins: Create secure applications by building complete CI/CD pipelines.* Packt Publishing. ISBN: 9781803245300. https://books.google.fi/books?id=3V5qEAAAQBAJ.

Liao, Hung-Jen, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. 2013. "Intrusion detection system: A comprehensive review". *Journal of Network and Computer Applications* 36 (1): 16–24. ISSN: 1084-8045. https://doi.org/10.1016/j.jnca.2012.09.004.

Lin, Xin, Lingguang Lei, Yuewu Wang, Jiwu Jing, Kun Sun, and Quan Zhou. 2018. "A Measurement Study on Linux Container Security: Attacks and Countermeasures". In *Proceedings of the 34th Annual Computer Security Applications Conference,* 418–429. ACSAC '18. Association for Computing Machinery. ISBN: 9781450365697. https://doi.org/10.1145/3274694.3274720.

Lin, Yuhang, Olufogorehan Tunde-Onadele, and Xiaohui Gu. 2020. "CDL: Classified Distributed Learning for Detecting Security Attacks in Containerized Applications". In *Annual Computer Security Applications Conference,* 179–188. ACSAC '20. Association for Computing Machinery. ISBN: 9781450388580. https://doi.org/10.1145/3427228.3427236.

Liu, Peiyu, Shouling Ji, Lirong Fu, Kangjie Lu, Xuhong Zhang, Wei-Han Lee, Tao Lu, Wenzhi Chen, and Raheem A. Beyah. 2020. "Understanding the Security Risks of Docker Hub". In *European Symposium on Research in Computer Security.* https://doi.org/10.1007/978-3-030-58951-6_13.

Mahboob, Jamal, and Joel Coffman. 2021. "A Kubernetes CI/CD Pipeline with Asylo as a Trusted Execution Environment Abstraction Framework". In *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC),* 0529–0535. https://doi.org/10.1109/CCWC51732.2021.9376148.

Patrizio, Andy. 2020. "5 ways cloud-native application testing is different from testing on-premises software". https://www.functionize.com/blog/5-ways-cloud-native-application-testing-is-different-from-testing-on-premises-software.

Rangnau, Thorsten, Remco v. Buijtenen, Frank Fransen, and Fatih Turkmen. 2020. ”Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines”. In *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC),* 145–154. https://doi.org/10.1109/EDOC49727.2020.00026.

Sawant, Abhijit, Pranit Bari, and Pramila Chawan. June 2012. ”Software Testing Techniques and Strategies”. *International Journal of Engineering Research and Applications(IJERA)* 2 (): 980–986. https://www.researchgate.net/publication/316510706_Software_Testing_Techniques_and_Strategies.

Shu, Rui, Xiaohui Gu, and William Enck. 2017. ”A Study of Security Vulnerabilities on Docker Hub”. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy,* 269–280. CODASPY ’17. Association for Computing Machinery. ISBN: 9781450345231. https://doi.org/10.1145/3029806.3029832.

Siddiqui, Shadab Alam, and Tamanna Siddiqui. 2021. ”Non-Functional Testing Framework for Container-Based Applications”. *Indian Journal of Science and Technology* 14 (47): 343–344. https://doi.org/10.17485/IJST/v14i47.1909.

Siddiqui, Tamanna, and Riaz Ahmad. 2016. ”A review on software testing approaches for cloud applications”. Recent Trends in Engineering and Material Sciences, *Perspectives in Science* 8:689–691. ISSN: 2213-0209. https://doi.org/10.1016/j.pisc.2016.06.060.

Siddiqui, Tamanna, Shadab Alam Siddiqui, and Najeeb Ahmad Khan. 2019. ”Comprehensive Analysis of Container Technology”. In *2019 4th International Conference on Information Systems and Computer Networks (ISCON),* 218–223. https://doi.org/10.1109/ISCON47742.2019.9036238.

Sidqui, Shadab Alam, and Tamanna Siddiqui. 2021. ”Quantitative Data Analysis of Non-Functional Testing in Container Applications”. In *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO),* 1–6. https://doi.org/10.1109/ICRITO51393.2021.9596457.

Sultan, Sari, Imtiaz Ahmad, and Tassos Dimitriou. 2019. ”Container Security: Issues, Challenges, and the Road Ahead”. *IEEE Access* 7:52976–52996. https://doi.org/10.1109/ACCESS.2019.2911732.

Syed, Madiha H., and Eduardo B. Fernandez. 2020. ”The Secure Container Manager Pattern”. PLoP ’18. Portland, Oregon: The Hillside Group. https://dl.acm.org/doi/10.5555/3373669.3373676.

Wan, Zhiyuan, David Lo, Xin Xia, and Liang Cai. 2019. ”Practical and effective sandboxing for Linux containers”. *Empirical Software Engineering* 24 (6): 4034–4070. ISSN: 1382-3256. https://doi.org/10.1007/s10664-019-09737-2.

Wong, Annika, Eyasu Getahun Chekole, Martín Ochoa, and Jianying Zhou. 2021. ”Threat Modeling and Security Analysis of Containers: A Survey”. *ArXiv,* https://doi.org/10.48550/arXiv.2111.11475.

Zampetti, Fiorella, Salvatore Geremia, Gabriele Bavota, and Massimiliano Di Penta. 2021. ”CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study”. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME),* 471–482. https://doi.org/10.1109/ICSME52107.2021.00048.

Appendices

A Survey Questions

Attachment of the survey questions

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Survey on Testing Software Containers

☐ Mandatory questions are marked with a star (*)

This survey is a part of the QLeap project and a Master's thesis in University of Jyväskylä.

There are 23 questions and it will take around an hour to answer.
The questions revolve around the topic of software container testing, and is about company practices and typical tools.

Answer each of the following questions as well as you can as applicable to the company you work at. At the end clarifications and additional comments are hoped to be written, in either English or Finnish.

For any questions or clarifications: salla.k.timonen@jyu.fi

Thank you!

1. Contact form *

Company name	
Email address	

Demographics Section

2. What sort of software is being containerized in the company?

3. What are your major responsibilities in the company?

4. How many years have you been working in the IT industry?

- ○ 0-2 Years
- ○ 3-5 Years
- ○ 6-10 Years
- ○ 11-15 Years
- ○ >15 Years

5. How many years have you been working with software containers?

- ◯ 0-2 Years
- ◯ 3-5 Years
- ◯ 6-10 Years
- ◯ >10 Years

Tools used in the company

6. What containerization platform(s) is in use in the company?

- ☐ Docker
- ☐ Kubernetes
- ☐ AWS Fargate
- ☐ Amazon ECS
- ☐ LXC
- ☐ Azure
- ☐ Other(s) (please specify) ____________

7. What tools to test container-based applications are used in the company?

Testing Software Containers

8. What is the process of testing containers in the company?

9. Is there testing done before containerization of the software?

- ◯ Yes
- ◯ No
- ◯ Not sure

10. Is there testing done after containerization of the software?

- ◯ Yes
- ◯ No
- ◯ Not sure

11. Is the testing specified to the container itself or the containerized content?

- ◯ Container itself
- ◯ Containerized content
- ◯ Both
- ◯ Not sure

12. What practices and techniques do you use to make containerized software more testable?

13. What type(s) of testing is done to containers and containerized software?

14. Are there plans to implement other types of testing, and if so, what?

15. What automated testing strategies do you use to test containers?

Results of Testing

16. What differences, if any, have been found through testing software before and after containerization?

17. Have you found that the act of containerization affects test results, if so, how?

18. What challenges do you face when testing containers?

Examples

19. What anomalies are you looking to find through testing containers and containerized software?

20. What sort of anomalies have been detected through testing containers and containerized software?

21. How are you planning to improve anomaly detection and accuracy?

22. What are aspects that you have not found through testing/are not testing for, but hope to implement in the future?

Additional Comments

23. Is there anything else you would like to add, clarify, or comment?