Anomaly Detection Through Container Testing: A Survey of Company Practices

Salla Timonen[0009−0000−4839−1686], Maha Sroor[0000−0001−6998−7358], Rahul Mohanani[0000−0001−7018−8836], and Tommi Mikkonen[0000−0002−8540−9918]

University of Jyväskylä, Mattilanniemi 2, Jyväskylä, Finland
{salla.k.timonen,maha.m.sroor,rahul.p.mohanani,tommi.j.mikkonen}@jyu.fi

Abstract. Background: Containers are a commonly used solution for deploying software applications. Therefore, container functionality and security is a concern of practitioners and researchers. Testing is essential to ensure the quality of the container environment component and the software product and plays a crucial role in using containers.
Objective: In light of the increasing role of software containers and the lack of research on testing them, we study container testing practices. In this paper, we investigate the current approaches for testing containers. Moreover, we aim to identify areas for improvement and emphasize the importance of testing in securing the container environment and the final software product.
Method: We conducted a survey to collect primary data from companies implementing container testing practices and the commonly used tools in container testing. There were 14 respondents from a total of 10 different companies with experience using containers and varying work responsibilities.
Findings: The survey findings illustrate the significance of testing, the growing interest in and utilization of containers, and the emerging security and vulnerability concerns. The research reveals variations in testing approaches between companies and the lack of consensus on how testing should be carried out, with advancements primarily driven by industry practices rather than academic research.
Conclusion: In this study, we show the importance of testing software containers. It lays out the current testing approaches, challenges, and the need for standardized container testing practices. We also provide recommendations on how to develop these practices further.

Keywords: Software containers · Testing · Survey

1 Introduction

Containers have become an integral part of modern software engineering (SE) processes, offering numerous benefits such as increased portability, scalability, and flexibility. Like any emerging new technology, containers potentially introduce new risks and vulnerabilities in the development process. With this in

mind, testing is fundamental to finding bugs that could impede the progress of containerized software development [1,2].

Containers built using pre-existing images and components can inherit vulnerabilities from dependencies, making testing vital to identify and mitigate risks. The dynamic nature of containers adds complexity, requiring testing to ensure security. Additionally, testing container interactions in distributed systems and their impact on overall security is crucial to prevent breaches [3]. Testing plays a vital role in creating functional and secure software containers.

With the increasing popularity of containers, there is a growing need to address the risks and vulnerabilities associated with their use [4,5]. By addressing these issues, organizations can safeguard sensitive data, prevent unauthorized access, and maintain the integrity of their software environment. However, there is no scientific, empirical consensus on the best practices for testing containers, as advancements in this area are predominantly driven by industry practices rather than academic research [6].

In this paper, we investigate the current status of company practices and tools for detecting anomalies through software container testing. The research focuses on two main aspects – (i) the current approaches employed by companies for testing software containers and (ii) how these approaches can be further developed. As an instrument to study the topic, a survey was conducted, with participation from relevant information technology companies. Hence, the results provide insights into current company practices and identify areas for further research, addressing the gap in the existing literature on testing software containers. Moreover, the results contribute to developing more effective and secure containers by exploring current practices and seeking ways to enhance them.

The structure of the paper is as follows. Section 2 provides some background information regarding what research is currently available on the topic of containers and their testing. Section 3 outlines how the survey was designed and conducted in addition to the tools we used for the analysis. Section 4 presents the results. Section 5 follows with further discussion. Finally, Section 6 draws some final conclusions.

2 Background

2.1 Software Containers

Software containers are a critical ingredient in the modern software development context. They enable rapid software updates in cloud context and practices such as DevOps by allowing the packaging of applications and all their necessary dependencies, such as software, configurations, libraries, frameworks, and binaries [7]. Containers isolate and virtualize the operating system, giving each containerized application a separate area of execution within the operating system. This enables multiple applications to run on a single operating system instance, resulting in more lightweight and manageable setups compared to running multiple operating system instances [1,2].

Containers have emerged as a solution to the challenges developers face when migrating applications between different environments, such as from development to testing and production. These migrations often encounter issues due to differences in both hardware and software configurations. To overcome these obstacles and improve flexibility, containers are a commonly used solution [1].

Containers are also widely used to support microservices [8]. Container management systems like Docker and orchestration systems like Kubernetes provide control of applications and provision resources, leading to the development of scalable, reliable, and reactive systems [2].

When using containers, the infrastructure typically consists of a repository for building container images and an image registry for deployment. The container setup can potentially lead to various security breaches, including data theft, denial of service attacks, and unauthorized access. Research has identified threats to containers such as spoofing, tampering, information disclosure, denial of service, and elevation of privilege [5].

Docker Hub, one of the most popular Docker image repositories, distributes official and community images. Security vulnerabilities in these images have been studied, driven by high-profile attacks reported through distribution channels. One study [3], introduces a Docker image vulnerability analysis framework called DIVA for analyzing vulnerabilities in Docker images. Furthermore, [9] identifies three major sources of security risks – sensitive parameters in run commands, malicious Docker images, and unpatched vulnerabilities in contained software.

In an early study in 2015, Docker image security was studied by inspecting images from Docker Hub using an open-source tool, "Banyan" Collector [10]. A more recent article from 2020 highlights that Docker does not offer assurances for recognized security vulnerabilities within container images [11].

2.2 Container Testing Approaches

To address the above concerns, four generalized use cases at the host-container level were proposed to identify threats and provide potential solutions [4]. These use cases include protecting a container from its internal applications, inter-container protection, host protection from containers, and container protection from the host.

Existing mitigation strategies and their limitations are discussed in [5]. Identified strategies included multi-factor authentication systems, implementing network controls, and security patching, but the need for a reliable and fast patching framework is emphasized as a research gap. Proper isolation is also discussed in the paper. On the other hand, another study suggests that kernel security mechanisms play a more critical role than container isolation methods in preventing privilege escalation attacks [12]. A sandbox mining and enforcing approach was proposed in a study [13] where a mined sandbox confines and restricts a container's access to system calls.

Minimizing administrative privileges is another approach to enhancing container security, with methods like anomaly detection systems [14,15]. Another article from 2021 [16] investigates the accuracy of container scanning tools and

highlights the vulnerability of the container's operating system. Additional mitigation strategies involve implementing strong access controls and ensuring containers remain lightweight [17].

While the mentioned mitigation strategies are valuable for enhancing container security, apart from scanning and monitoring, they focus on preventive measures. Furthermore, if implemented incorrectly, they can create a false sense of security. In addition to these mitigation strategies, there are container security frameworks and testing approaches. One example of anomaly-based detection in containers is from a 2020 article on the implementation of Classical Distributed Learning to detect security attacks in containerized applications [14].

A different approach was outlined in another 2020 article proposing the Docker Image Vulnerability Diagnostics System [11]. It was designed to diagnose Docker images during upload or download, addressing the lack of vulnerability diagnostics in current Docker image distribution. In 2022, the SEAF framework [18] was proposed for scalable, efficient, and application-independent container security detection. Inspecting various security defects and evaluating their impact, it found more than 35 000 security defects from popular repositories.

Other frameworks include the Secure Container Manager Pattern [19] and the Framework to Secure Docker Containers [20], focusing on container management and container security, respectively. CONSERVE is another framework for selecting container monitoring techniques in different application domains [21].

An article dated 2021 [6] highlights the absence of global standards for testing the non-functional part of applications. The authors subsequently developed the Non-Functional Testing Framework for Container-Based Applications [22], emphasizing the crucial role of testing results in determining an application's success. They continue to state that the evolution of containers has been mainly driven by industry adaptation than academic research, particularly in the realm of testing frameworks for container-based applications.

To address the challenges of the unknowns, Siddiqui et al. [22] define four non-functional attributes for container applications, characterizing the surrounding environment and its behaviour: capacity, scalability, stability, and high availability. Although a review of testing approaches was conducted, it focused on cloud-based applications [23].

3 Research Methodology

In this work, we aim to encapsulate current company practices and tools for detecting anomalies in container testing by answering the following research questions:

RQ1: What are the current approaches employed by companies for testing software containers?

RQ2: How can the approaches for testing software containers be improved?

We used an online questionnaire survey instrument to answer the research questions. We followed the standard guidelines for planning and conducting survey studies in SE [24], and empirical standards for SE research [25]. A survey

was chosen as the most suitable study method for this research, enabling convenient online distribution and data collection from respondents. Also, it supported collecting valuable insight from participating companies, allowing respondents ample time to review and contemplate the questions without time constraints. You can access the survey questionnaire used to collect data for our research here: https://zenodo.org/record/8378974.

The survey includes a total of 23 questions. The contact form is the only mandatory question. The questions comprised both—multiple-choice questions and open-ended questions. Out of the total, 16 open-ended questions allowed respondents to provide detailed and in-depth responses without being restricted by predefined options. The survey was created with Webropol 3.0[1]—a survey and reporting tool that simplifies creating and distributing online surveys. It also ensures the secure collection, processing, storage, and archiving/destruction of survey data. Moreover, it enables monitoring survey engagement, including the number of opened links, started responses, and completed surveys.

The survey consisted of five sections designed to collect comprehensive information related to the research objectives, as follows:

- *Demographics*: gather data on respondents' roles and experiences.
- *Tools used in the company*: gather data on container-related tools.
- *Testing software containers*: gather data on processes, practices, and future plans for anomaly detection through container testing.
- *Results of testing*: gather data on outcomes, challenges, and assess containerization's influence on testing.
- *Anomaly examples*: gather data on challenges and future implementation plans.

Overall, the survey encompassed a comprehensive examination of respondents' roles, tooling landscape, container testing processes, testing outcomes, and anomaly examples, thereby enabling a thorough exploration of containerization practices and their impact on testing.

The survey gathered a total of fourteen responses from individuals affiliated with ten distinct companies. The work was carried out in industry-academia collaboration, with seven companies contacted directly via email in order to solicit respondents. In addition, other organizations were targeted through a LinkedIn post, seeking respondents interested in the topic of research. The target sample was expected to possess satisfactory knowledge regarding containers and a minimum of one year of experience. Throughout the data collection period, a total of 137 individuals opened the survey (referred to as the "Total"). Out of these participants, 32 initiated the response process (referred to as the "Net Participation"), representing 23.36% of the Total. Of the "net participation" group, 14 individuals successfully completed the survey, accounting for 43.75% of the net participation. The data was collected between January 11th, 2023, and June 13th, 2023.

[1] https://webropol.com/

4 Results

4.1 Demographics

The survey involved a sample of 14 respondents, referred to as P1 through P14. The responses provided insights into a wide array of projects undertaken by these companies, including research and development (R&D), cloud software development, machine learning applications, value stream metrics, web application development, testing activities, back-end services, and development environments. Collectively, the responses reflected the broad spectrum of industries and domains in which containerization practices were implemented.

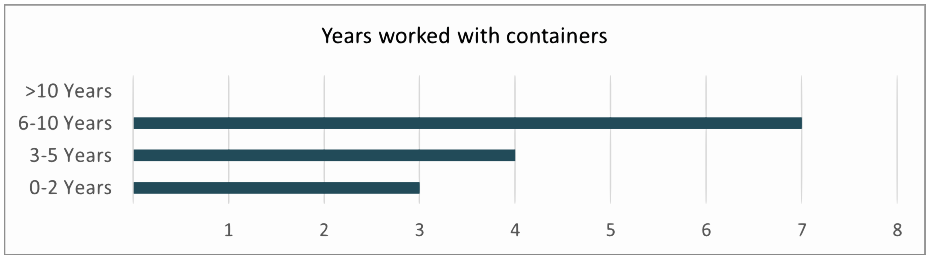

Fig. 1. Respondents' experience with containers

Figure 1 visually presents an overview of the respondents' experience levels. Among the total respondents, seven individuals (50% of the total) reported having 6–10 years of experience with containers. Additionally, four respondents indicated having 3–5 years of experience, while three reported having 0–2 years of experience with containers.

4.2 Tools Used in Companies

Table 1 provides a comprehensive overview of the container platforms the respondents utilize, highlighting the prominent role of various cloud-based platforms. Among the mentioned platforms, Docker emerged as the most widely adopted, with all respondents except one reporting its usage. Kubernetes followed closely with 11 users, while both Azure and Amazon Elastic Container Service (ECS) were utilized by eight respondents each. Another popular platform, AWS Fargate, had a user count of six. Additionally, respondents mentioned other container platforms such as LXC, Nomad, Podman, OCI, Rancher (RKE), OpenShift, and k3s.

The survey respondents mentioned a variety of tools utilized for testing container-based applications. These tools included JUnit, TestContainers (Java), Docker Desktop, Minikube, Robot framework, Spira, Sonarqube, Trivy, Selenium, Protractor, and Jest. Additionally, specific test frameworks like Pact.io

Table 1. Container platforms used by respondents.

Container platform	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	Total
Docker	x	x	x	x	x	x	x	x	x	x	x	x		x	13
Kubernetes		x	x	x	x	x	x	x	x	x	x	x			11
Azure		x	x	x		x	x	x			x	x			8
Amazon ECS				x		x	x	x	x	x	x			x	8
AWS Fargate						x	x	x	x		x			x	6
LXC						x	x	x							3
Other(s)			x	x									x		3

and Cucumber were also mentioned. Regarding security and vulnerability testing, respondents identified tools such as Black Duck, AWS Security Hub, Qualys, and Defensics.

The selection of these tools was influenced by factors such as the programming language, the technology stack employed, and the different teams involved within the respondents' respective companies.

4.3 Testing Software Containers

The processes that were used to test containers varied a lot between companies and respondents' roles. However, some common testing practices and tools were mentioned, listed below:

1. Using Github test cases, test containers, continuous integration and continuous deployment (CI/CD) tools, and Robot framework.
2. Conducting tests in a replicated container environment, starting with robot tests on code changes and progressing to deployment in development, staging, and production environments.
3. Performing local testing during development and automated testing in the company's CI/CD platform.
4. Implementing various release trains and CI/CD pipelines, including unit tests, API tests, integration tests, and static code analysis.
5. Incorporating automatic vulnerability scans using tools like Harbor and Trivy in the CI/CD pipeline.
6. Using containers to run full development environments and perform integration tests.
7. Conducting unit tests, functional tests, reliability tests, and security and open-source license scans.
8. Emphasizing integration testing before containerization and occasionally using end-to-end (E2E) tests in cloud-based containerized environments.

In addition to the above, Docker unit tests, container structure tests, and manual functional testing for software within the container were also mentioned by some respondents.

Thirteen respondents mentioned conducting testing before and after containerization, while one respondent was uncertain. Among them, seven stated

that testing was specified for the containerized content, five mentioned testing for both the container and the content, and one respondent was unsure.

The answers revealed different approaches to enhancing testability of containerized software. These included running containers in a local development environment, employing loose coupling and abstraction, performing unit testing, using Docker exec commands for output examination, enabling test versions of dependencies, ensuring clear network interfaces between containers, implementing container hardening, focusing on log design, and employing automated pipelines.

Our findings also highlight a wide range of testing methodologies applied to containers and containerized software. These included user interface and user experience tests, database tests, logic tests, security tests, penetration tests, checks for misconfigured containers, manual functional testing, unit tests, functional testing, exploratory testing, test automation, scalability and high availability testing, deployment testing, container structure tests, integration tests, E2E testing, reliability testing, performance testing, and legal testing.

Regarding future testing strategies, emphasis was placed on the automation of various test types, such as unit testing, security testing, integration tests, and quality assurance (QA) steps, including static code analysis and vulnerability scans. Suggestions were made to utilize more tools in vulnerability testing and testing supply chain attacks, incorporate additional scanning tools to enforce best practices, and execute tests within a container as part of the CI/CD pipeline.

Furthermore, the utilization of CI/CD pipelines for managing code changes and the adoption of the MS Azure infrastructure for staging and deployment purposes were also mentioned by some respondents.

4.4 Results of Testing

When asked about the differences observed in testing software before and after containerization, some respondents highlighted positive aspects while others raised concerns about negative aspects. Positive aspects mentioned by respondents included easier automation, consistent software layers, improved execution of full system and E2E tests, more repeatable environments, increased likelihood of production-like environments on development machines, and targeted change testing with isolated environments for each branch.

On the negative side, it was noted that if system logging was not properly implemented before testing, the analysis of error situations required additional effort. Additionally, two respondents mentioned that they either did not observe significant differences or were unaware of any differences in cases where the software had been containerized from the beginning. This suggests that the benefits or challenges associated with containerization may vary depending on the specific context and the extent to which containerization has been integrated into the software development and testing processes.

The responses regarding how the act of containerization affects test results exhibited variability, even among respondents from the same company. While opinions varied, a range of perspectives were expressed.

Four respondents indicated that they did not observe any significant impact of containerization on test results and suggested that containerization did not noticeably alter the outcomes of their testing efforts. On the other hand, two respondents highlighted the positive effects of containerized testing. They emphasized that containerization led to more consistent and faster test results, along with easier maintenance of the testing environment. These individuals found that containerization provided benefits such as improved consistency in test outcomes and enhanced efficiency in maintaining the test infrastructure. Furthermore, multiple respondents emphasized automation capabilities enabled by containers. These varied responses demonstrate that the impact of containerization on test results can differ based on the specific context, company, and individual perspectives.

Another aspect brought up was the need for different configurations in testing, considering that the software environment may impact its behaviour. If tested before containerization, the execution environment may differ from that within a container.

A question about challenges faced when testing containers received numerous answers. The challenges mentioned include the need to have the test environment within the container, longer compilation times, network connectivity issues due to misconfigured containers, difficulties in creating testable environments, complexities in setting up infrastructure for containerized tests, dependencies on additional services for integration or E2E tests (particularly in a microservices architecture), dynamic visualization of results in multi-container environments, synchronization between containers, slow system ramp-up for testing, testers facing challenges in contributing due to managing their settings, difficulty in understanding testing from a container perspective, complexities in debugging tests within complex containerized environments, and more.

4.5 Anomaly Examples

Specific anomalies mentioned to be tested for include regression issues, performance downgrades, load-related anomalies such as session replication issues and bottlenecks, memory leaks, vulnerabilities identified through static code analysis, long-term stable execution on different hardware, smooth dynamic scalability, broken APIs between dependencies, logic issues related to reconnecting and disconnecting, software crashes or hangs, and impacts on functionality, reliability, performance, and supportability.

Regarding what sort of anomalies have been detected through testing, several respondents mentioned that they had not encountered many specific anomalies. However, the anomalies mentioned included errors, regression issues, performance downgrades, session replication issues, critical vulnerabilities such as the log4j bug in 2021 (which was caught through automatic vulnerability scans), challenges with low-level drivers, broken APIs between dependencies, reconnect/disconnect logic issues, failed crash/hang detection, and misconfigurations related to interface definitions and security hardening in containers.

While two respondents mentioned that they had no plans to improve anomaly detection and accuracy, others provided various strategies and approaches:

1. Utilizing static code analysis.
2. Conducting vulnerability scanning.
3. Incorporating the test framework within a container during testing.
4. Developing enhanced testing components that gather more data and can be reused.
5. Increasing knowledge about containerized testing and automation.
6. Implementing daily scans of containers against vulnerability databases.
7. Conducting more testing in development and staging environments.
8. Enhancing health check logic.
9. Isolating changes to identify the specific time when issues occurred.

The question on anomalies that are not yet being tested for but are hoped to implement in the future was answered by only 7 respondents (50%). Although fewer respondents provided answers, several aspects were mentioned:

1. Emphasizing integration testing of the entire architecture in the actual runtime environment rather than focusing solely on individual containers.
2. Automating testing for high-load scenarios.
3. Enhancing the ability to compare resource requirements and performance by analyzing metrics between different releases in CI/CD execution.
4. Increasing the frequency of static scans for container definitions.
5. Addressing the dynamic behaviour of containers in terms of scalability.
6. Conducting stress testing to evaluate system performance under extreme conditions.

5 Discussion

The previous literature has presented approaches to anomaly detection through container testing [6,18,22]. This paper, however, investigates real-world anomaly detection practices in companies and offers improvement approaches. Moreover, it narrows the knowledge gap between industry and academia on container anomaly detection. The survey results comprehensively analyze container testing processes and challenges by leveraging insights from individuals with diverse experiences and roles within our sample. The sample encompasses participants with junior expertise, less than two years of container experience, and those with senior experience, 3 to 10 years of container-related experience. We also ensured variability in the roles and responsibilities of the respondents, incorporating individuals from technical positions such as developers, software architects, testers, and security analysts, as well as managerial positions like heads of R&D, development managers, and team leaders. The diverse composition ensures that our findings encompass technical and managerial perspectives on container testing.

Our findings revealed that Docker and Kubernetes are the most prevalent container platforms. Docker emerged as the leading choice, which corroborates

the findings reported in the literature [1]. While Docker and Kubernetes dominated the landscape, we also observed other popular tools, such as Azure, Amazon ECS, and AWS Fargate.

Our survey also uncovered several tools with relatively lower adoption rates. By discerning the prevalence of these diverse software tools, our research contributes valuable insights into the prevailing trends in the domain of container technology. The prominence of Docker and Kubernetes underscores their pivotal roles in shaping contemporary containerization practices.

The survey results further highlight that the selection of container tools is contingent upon several factors, including the software language, technology stack, and the team's preferences within the organization. This finding underscores the importance of tailoring tool choices to align with specific project requirements and team dynamics, reflecting a pragmatic approach to containerization. Participants reported employing multiple testing tools to cater to their respective testing needs. Alongside commercial testing tools, respondents incorporated in-house test platforms, automated CI/CD pipeline tests, and container checker tools into their testing processes. This amalgamation of testing tools underlines the flexibility and adaptability of testing strategies to suit various use cases and testing objectives. By incorporating this comprehensive range of container-related testing tools, our research illuminates the dynamic nature of container testing practices in contemporary software development. This diversity of tooling options underscores the significance of a nuanced and context-aware approach when selecting the most appropriate testing tools.

In the "testing containers" section of the survey, there was a consensus among respondents, indicating that most companies conduct testing both before and after containerization, with a focus on testing the containerized content. The responses also emphasized that current testing techniques are insufficient, and companies are endeavoring to develop procedures to enhance container testability tailored to their specific needs.

The survey results reveal that companies are actively strategizing to enhance their testing solutions, emphasizing automated testing. While the majority of participating companies already utilize automated testing, they view its expansion as a critical defensive measure to augment testing outcomes. Notably, companies exhibited diverse perspectives when asked about their strategies for expanding automated testing. Some companies directed their efforts towards automating testing for pipelines, aiming to streamline the testing process within their CI/CD pipelines. Others concentrated on automating code testing, seeking to automate code functionality and integrity verification. Additionally, some companies expressed their desire for a comprehensive, fully automated testing package, suggesting an overarching approach to automation covering various testing aspects.

These findings indicate the industry's growing inclination towards automation to achieve more efficient, reliable, and comprehensive testing practices. The diversity in companies' approaches highlights the nuanced nature of devising tai-

lored automated testing strategies to address their specific testing requirements and objectives.

The survey addressed the main challenges facing container testing. These challenges include longer compilation times, networking and configuration issues, test environment setup and management difficulties, dependency integration in microservices architecture, visualization and result reporting challenges, and debugging complexities.

The survey results extend the knowledge from industry to literature, adding details about the testing methods used. Moreover, the survey results confirm the common use of tools, diverse needs and corresponding methods and tools [1,7,16], concerns about security and vulnerabilities [4,5,3,17], the importance of automation and CI/CD implementations and dynamic program analysis [16], and the need for further knowledge and research [6].

5.1 Revisiting Research Questions

RQ1: What are the current approaches employed by companies for testing software containers? Currently, companies heavily rely on the tools employed to test containers. Docker stands out as the most utilized container platform, with a mention of 12 different container platforms in total. The tools for testing container-based applications exhibit even greater diversity, often being technology-stack dependent, with 18 tools and test frameworks mentioned and the possibility of many more.

Despite discrepancies in testing approaches, companies share a consensus on multi-phase testing processes. Commonalities among the companies' processes include manual tests, CI/CD pipelines with integrated tests and scans, monitoring, and reporting. Significant differences surface in container testing processes, particularly in two aspects. First is the testing environment, where variations arise depending on whether the test is conducted locally, in a replicated environment, or through other configurations. Second is the extent of testing, ranging from manual functional testing to the more comprehensive multi-container end-to-end setup. The latter involves replicating the production environment, simulation of the staging environment for deployment, verification testing, and, ultimately, production release. One respondent also stated the use of containers was for running development environments allowing the running of complete environments for integration tests.

Our findings also highlight disparities in companies' perspectives regarding conventional testing approaches. Numerous respondents outlined their typical testing approach, encompassing test cases, containers, CI/CD, and automated code change checks. Interestingly, two distinct groups emerged from the responses. The first group focused solely on testing the content within containers. Conversely, the second group stressed the necessity of incorporating additional container-specific testing measures. These measures encompassed container reliability, security assessments, enhanced monitoring capabilities, open-source license scans, input-output comparison validation, and functionality testing in a multi-container setup.

The contrasting viewpoints elucidate the diverse strategies adopted by companies when testing containerized applications. This underscores the significance of tailoring testing practices to suit individual needs and organizational contexts, leading to more robust and comprehensive testing outcomes. While there is shared recognition among companies about the implications of vulnerability testing and its ability to detect misconfiguration issues, not all companies prioritize it. Surprisingly, some companies did not mention vulnerability testing when asked about the types of testing performed.

In summary, the survey results revealed a lack of consensus among companies regarding container testing approaches, the essential phases to be incorporated into the testing process, understanding typical testing practices, and recognizing the significance of security and vulnerability testing. Furthermore, the findings demonstrate the diverse approaches adopted by companies, as discussed in [22].

RQ2: How can the approaches for testing software containers be improved? Although the results addressed various practices to enhance the testability of containers, companies do not agree on "best practices" for testing containers. Therefore, we collected general recommendations from the survey results to improve testing practices. The provided recommendations are from a managerial perspective to improve the culture around container testing and from a technical perspective to improve the testing practices.

The recommendations for managerial positions include prioritizing learning and staying updated on container testing practices, conducting thorough research to understand the benefits and challenges, reviewing existing testing processes in use to identify areas for improvement, and critically evaluating the suitability of testing tools based on specific project requirements. Managers also recommended fostering a culture of continuous learning and improvement, promoting collaboration, and allocating resources for training and skill development in container testing. Additionally, actively engaging with industry experts and seeking insights from current research and other organizations can provide valuable insights for optimizing container testing strategies.

Our recommendations for technical positions in approaching container testing from a technical standpoint include:

1. Establishing standardized testing processes: establishing standardized testing processes tailored to their specific needs and project requirements can help ensure consistency and efficiency in testing practices.
2. Enhancing tool selection and compatibility: companies should carefully evaluate and select tools that align with their technology stack and testing requirements achieving compatibility, suitability of, and effectiveness of container testing.
3. Incorporating additional types of testing: companies emphasized the importance of incorporating automated tests, integration tests, security testing, and vulnerability scans to enhance the comprehensiveness and accuracy of container testing.

4. Paying attention to misconfiguration issues and anomaly detection: companies recommended focusing on misconfiguration issues and anomaly detection as an essential part of container testing to mitigate potential threats.
5. Combining testing techniques: companies suggested combining testing techniques like running containers in a local development environment, ensuring loose coupling, utilizing abstraction, implementing unit testing, and establishing clear network interfaces could improve the testing results.
6. Integrating testing within CI/CD pipelines: companies emphasised that using CI/CD pipelines for integrated tests and scans can streamline the testing process. Moreover, it ensures continuous monitoring and reporting, which improves the whole container environment's efficiency, reliability, and consistency.
7. Developing the knowledge of challenges and complexities: companies mention that developing the knowledge of container challenges and complexities among practitioners would keep them aware of the possible system vulnerabilities. Also, it would help them choose suitable testing techniques.

5.2 Limitations and Threats to Validity

The research has several limitations that should be considered. First, the survey relied on a relatively small sample size of 14 respondents from 10 different companies, which may limit the generalizability of the findings. Additionally, the data collected in the survey relies on self-reported responses from the respondents, introducing the possibility of response bias. Furthermore, we acknowledge that the lack of existing academic research on container testing practices may restrict the depth of analysis and comparison with prior studies. Obviously, these limitations should be considered when interpreting the results and considering the broader applicability of the findings.

6 Conclusion

This paper studies the current testing practices in the context of the growing interest in software containers and the lack of comprehensive knowledge of testing approaches. It aims to identify the challenges faced in container testing and potential areas for improvement by collecting information from companies that are actively engaged in container testing. The collected data centred on critical areas such as the tools utilized to support containers, prevailing approaches in container testing, challenges encountered during the testing process, and the role of anomaly and vulnerability detection in supporting container testing for improved system security. The findings derived from this research are of utmost significance for companies and practitioners involved in testing software containers. They reveal a lack of consensus on the best practices of container testing, shedding light on the current challenges faced and presenting proposed solutions to elevate testing practices. Moreover, this research offers valuable recommendations based on both technical and managerial experiences, providing valuable insights for improving container testing processes.

In summary, this research contributes with the following major findings:

1. It reveals real-world practices for implementing anomaly detection.
2. It offers actionable guidance to practitioners for implementing anomaly detection in software containers.

To conclude, this work contributes to understanding container testing practices and establishes a solid foundation for future research endeavors and industry advancements in this domain. Despite the limitations of the work, by addressing the complexities and evolving requirements of container testing, this research aims to propel the field forward, bolstering the overall security and reliability of containerized software applications.

Acknowledgements The research was conducted as part of the Containers as the Quantum Leap in Software Development (QLeap) project, involving the University of Jyväskylä and various industry partners.

References

1. Siddiqui, T., Siddiqui, S., Khan, N.: Comprehensive Analysis of Container Technology. 4th International Conference on Information Systems and Computer Networks (ISCON), 218–223 (2019). https://doi.org/10.1109/ISCON47742.2019.9036238
2. Douglis, F., Nieh, J.: Microservices and Containers. IEEE Internet Computing 23 (6): 5–6 (2019). https://doi.org/10.1109/MIC.2019.2955784
3. Shu, R., Gu, X., Enck, W.: 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 (2017). https://doi.org/10.1145/3029806.3029832
4. Sultan, S., Ahmad, I., Dimitriou, T.: Container Security: Issues, Challenges, and the Road Ahead. IEEE Access 7:52976–52996 (2019). https://doi.org/10.1109/ACCESS.2019.2911732
5. Wong, A., Chekole, E., Ochoa, M., Zhou, J.: Threat Modeling and Security Analysis of Containers: A Survey. ArXiv (2021). https://doi.org/10.48550/arXiv.2111.11475
6. Siddiqui, S., Siddiqui, T.: Quantitative Data Analysis of NonFunctional Testing in Container Applications. In 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO), 1–6 (2021). https://doi.org//10.1109/ICRITO51393.2021.9596457
7. Chen, C., Hung, M., Lai, K., Lin, Y.: Docker and Kubernetes. In Industry 4.1: Intelligent Manufacturing with Zero Defects, 169–213 (2022). https://doi.org/10.1002/9781119739920.ch5
8. Jamshidi, P., Pahl, C., Mendonça, N., Lewis, J., Tilkov, S. Microservices: The Journey So Far and Challenges Ahead. IEEE Software 35 (3): 24–35 (2018). https://doi.org/10.1109/MS.2018.2141039
9. Liu, P., Ji, S., Fu, L., Lu, K., Zhang, X., Lee, W., Lu, T., Chen, W., Beyah, R.: Understanding the Security Risks of Docker Hub. In European Symposium on Research in Computer Security (2020). https://doi.org/10.1007/978-3-030-58951-6_13
10. Gummaraju, J., Desikan, T., Turner, Y.: Over 30 percent of Official Images in Docker Hub Contain High Priority Security Vulnerabilities. (2015) https://www.banyansecurity.io/blog/over-30-of-official-images-in-docker-hub-contain-high-priority-security-vulnerabilities/ Last Accessed 20 Jun 2023.

11. Kwon, S., Lee, J.: DIVDS: Docker Image Vulnerability Diagnostic System. IEEE Access 8:42666–42673 (2020). https://doi.org/10.1109/ACCESS.2020.2976874
12. Lin, X., Lei, L., Wang, Y., Jing, J., Sun, K., Zhou, Q.: A Measurement Study on Linux Container Security: Attacks and Countermeasures. In Proceedings of the 34th Annual Computer Security Applications Conference, 418–429 (2018). https://doi.org//10.1145/3274694.3274720
13. Wan, Z., Lo, D., Xia, X., Cai, L.: Practical and effective sandboxing for Linux containers. Empirical Software Engineering 24 (6): 4034–4070 (2019). https://doi.org//10.1007/s10664-019-09737-2
14. Lin, Y., Tunde-Onadele, O., Gu, X.: CDL: Classified Distributed Learning for Detecting Security Attacks in Containerized Applications. In Annual Computer Security Applications Conference, 179–188 (2020). https://doi.org/10.1145/3427228.3427236
15. Kang, D., Fuller, D., Honavar, V.: 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 (2005). https://doi.org/10.1109/IAW.2005.1495942
16. Javed, O., Toor, S.: Understanding the Quality of Container Security Vulnerability Detection Tools. (2021). https://doi.org/10.48550/arXiv.2101.03844
17. Efe, A., Aslan, U., Kara, A.: Securing Vulnerabilities in Docker Images. International Journal of Innovative Engineering Applications 4 (1): 31–39 (2020). https://doi.org//10.46460/ijiea.617181
18. Chen, L., Xia, Y., Ma, Z., Zhao, R., Wang, Y., Liu, Y., Sun, W., Xue, Z.: SEAF: A Scalable, Efficient, and Application-independent Framework for container security detection. Journal of Information Security and Applications 71 (2021). https://doi.org//10.1016/j.jisa.2022.103351
19. Syed, M., Fernandez, E.: The Secure Container Manager Pattern. PLoP '18 (2020). Portland, Oregon: The Hillside Group. https://dl.acm.org/doi/10.5555/3373669.3373676
20. Abhishek, M., Rajeswara Rao, D.: Framework to Secure Docker Containers. In 2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability, 152–156 (2021). https://doi.org//10.1109/WorldS451998.2021.9514041
21. Jolak, R., Rosenstatter, T., Mohamad, M., Strandberg, K., Sangchoolie, B., Nowdehi, N., Scandariato, R.: CONSERVE: A framework for the selection of techniques for monitoring containers security. Journal of Systems and Software 186:111158 (2021). https://doi.org//10.1016/j.jss.2021.111158
22. Siddiqui, S., Siddiqui, T.: Non-Functional Testing Framework for Container-Based Applications. Indian Journal of Science and Technology 14 (47): 343-344 (2021). https://doi.org//10.17485/IJST/v14i47.1909
23. Siddiqui, T., Ahmad, R.: A review on software testing approaches for cloud applications. Recent Trends in Engineering and Material Sciences, Perspectives in Science 8:689–691 (2016). https://doi.org//10.1016/j.pisc.2016.06.060
24. Molléri, J., Petersen, K., Mendes, E.: Survey Guidelines in Software Engineering: An Annotated Review. Proceedings of the 10th ACM/IEEE ESEM, Article 58 (2016). https://doi.org//10.1145/2961111.2962619
25. Ralph, P., Baltes, S., Bianculi, D., Dittrich, Y., Felderer, M., Feldt, R., Filieri, A., Furia, C., Graziotin, D., He, P., Hoda, R., Juristo, N., Kitchenham, B., Robbes, R., Méndez, D., Molléri, J., Spinnelis, D., Staron, M., Stol, K., Tamburri, D., Torchiano, M., Treude, C., Turkhan, B., Vegas, S.: Empirical Standards for Software Engineering Research. ACM SIGSOFT Empirical Standards (2020). https://doi.org//10.48550/arXiv.2010.03525