© Rzeszów University Press
ISSN 2719-6550
e-ISSN 2719-7417

"Journal of Education, Technology and Computer Science"
No. 6(36)/2025
https://journals.ur.edu.pl/jetacomps

Received: 3.08.2025
Accepted for printing: 24.11.2025
Published: 31.12.2025

DOI: 10.15584/jetacomps.2025.6.16
Scientific

PAWEŁ DYMORA[1], **MIROSŁAW MAZUREK**[2]

Performance Evaluation of Selected Containerization Methods in Web Services Applications

[1] ORCID: 0000-0002-4473-823X, Ph.D. Eng., University of Technology, Faculty of Electrical and Computer Engineering, Poland; email: pawel.dymora@prz.edu.p
[2] ORCID: 0000-0002-4366-1701, Ph.D. Eng., University of Technology, Faculty of Electrical and Computer Engineering, Poland; email: mirekmaz@prz.edu.pl

Abstract

This paper evaluates the performance of selected containerization methods. As part of the research, infrastructure was created using Google Cloud public cloud, and then load tests were conducted for each containerization method using Apache JMeter. The study has shown that choosing the right containerization method depends on the service to be implemented. It was demonstrated that the highest performance was achieved by combining Podman with Docker. An example implementation and performance of Kubernetes technology, together with Docker and autoscaling using Google Cloud, was demonstrated. The project demonstrates high educational value by combining theory with practice through cloud-based infrastructure testing using Google Cloud and Apache JMeter. It helps learners understand container behavior under load, resource usage, and orchestration techniques. Additionally, the study emphasizes critical decision-making in selecting appropriate technologies based on specific service requirements. It serves as a practical teaching tool for IT and computer science education, offering insights into real-world DevOps workflows, scalability strategies, and performance optimization.

Keywords: Kubernetes, Docker, Podman, Google Cloud, Apache JMeter

Introduction

Nowadays, web services play a key role in the operation and dynamic growth of many organizations, from small start-ups to multinational corporations. With each passing day, the role of IT infrastructure continues to grow in ensuring the scalability, reliability, and performance of the services in question. Before the advent of containerization, virtualization dominated the market as the main technology enabling efficient management of IT resources. Virtualization allowed the creation of multiple virtual machines (VMs) on a single physical server, which significantly

increased the efficiency of using available resources. Each VM ran in an isolated environment, which provided security and the ability to run different operating systems on the same hardware. This technology revolutionized the way companies managed their data centers, enabling server consolidation, reducing operating costs, and increasing flexibility in IT infrastructure management.

This work has significant educational value as it introduces students and professionals to the practical application of containerization technologies in modern IT infrastructures. By comparing Docker, Podman, LXC, and Kata Containers through real-world performance testing in a cloud-based environment, the project provides a hands-on framework for understanding key concepts such as resource allocation, container orchestration, and load balancing. The detailed deployment process using Google Cloud, the use of Apache JMeter for testing, and the comparative analysis of metrics allow learners to gain insights into how different container engines behave under various workloads. This experiential learning approach deepens theoretical understanding and equips learners with essential skills needed in cloud computing, DevOps, and systems engineering.

From a didactic perspective, the study also highlights critical thinking in selecting the right technology for a given use case, emphasizing that there is no one-size-fits-all solution in containerization. The evaluation encourages learners to assess trade-offs between performance, security, scalability, and usability. Furthermore, the integration of tools like Kubernetes and CI/CD pipelines introduces students to industry-standard practices, helping bridge the gap between academic knowledge and real-world implementation. As a teaching aid, the project can be used in advanced IT and computer science courses to demonstrate infrastructure planning, container deployment strategies, and performance benchmarking in a controlled, replicable environment.

Containerization

However, over time, containerization has started to gain popularity, offering even greater efficiency and flexibility. Containerization has made it possible to isolate applications so that only the elements necessary for a service to function properly are stored in containers. What is more, if you send a person from your team a ready-made container image, for example, one is 100 percent sure that the person will have an identical environment once it is up and running, which makes teamwork much easier. Containerization, due to its lightness, reliability, simplicity, and speed, quickly found a huge number of followers and instantly conquered the market. Hosting one's applications on servers or virtual machines was quickly replaced by containerization, which now dominates the production environments of most companies, combined with tools for orchestrating them, such as Kubernetes or Docker Swarm (Fava et al., 2024).

Containerization offers lower resource consumption, faster application start-up, and better scalability compared to traditional virtual machines, making it the

preferred solution in modern IT infrastructures. Virtualization has been central to the development of IT, enabling efficient management of resources through virtual machines. However, containerization has introduced a breakthrough, offering application isolation, lightness, reliability, and speed. With tools such as Kubernetes, containerization has replaced traditional servers and virtual machines, providing better scalability and performance in modern IT infrastructures.

Virtualization involves creating a virtual computer environment that is opposite to the physical environment. This allows organizations to divide the resources of a host machine into several different, separate, completely virtual, isolated machines. These machines can interact independently and embed different operating systems and applications. Type one virtualization, also known as bare metal virtualization, involves the installation of virtualization software directly on a physical server that acts as a virtualization host. No or minimal host operating system is required, and the virtualization layer runs directly on the hardware. Type two virtualization, also known as hosted virtualization, involves installing virtualization software on an existing operating system (host) that is already running on a physical server. The virtualization software runs as an application on this operating system (Zordevic, Timcenko, Lazic, Davidovic, 2022). Examples of such solutions are VMware Workstation (for Windows and Linux), Oracle VirtualBox, and Parallels Desktop (for macOS) (https://azure.microsoft.com/pl-pl/resources/cloud--computing-dictionary/what-is-virtualization; https://kubernetes.io/blog/2020/12/08/kubernetes-1-20-release-announcement/).

Containerization is a technology that has conquered the IT market in a very short time. A container is a defined environment that contains everything needed to run an application, including code, execution environment, libraries, tools, and system configuration, which ultimately results in the lightness and speed of running such a container. Containers are isolated from each other and from the host on which they are run, which allows the application to run consistently and independently of the surrounding environment, however, if necessary, it is also possible to create a special network, which is created inside the host machine and in which it is possible to create individual working environments and ensure communication between individual units. Another very important aspect is that containerization enables developers and system administrators to easily deploy, run, and manage applications in different environments, which contributes to faster software delivery, improved infrastructure performance and flexibility, and increased application security. Containerization also enables easier deployment of microservices-based architectures and the use of continuous integration and delivery (CI/CD) techniques (Fava et al., 2024; Zeng, Wang, Deng, Zhan, 2017; https://cloud.google.com/logging/docs/agent/ops-agent).

There are several different container engines in the IT market, each with its own unique features and advantages. Until recently, the dominant position was

held by Docker Runtime, which was widely popular among organizations. However, in recent years, Kubernetes, as one of the leading container management platforms, has expressed a preference for containerd as the standard runtime for containers. While this announcement has not made containerd the only choice, it has certainly increased its popularity in the Kubernetes ecosystem. In addition to Docker Runtime and containerd, there are other container engines on the market, such as Podman, CRI-O, Linux Containers (LXC), and Kata Containers. Each of these engines has its place and use, depending on the users' needs and preferences. It is also worth noting that Docker currently uses containerd as its default runtime for containers.

Docker was initially a private project, continuously developed by a company located in South Africa, specifically by Solomon Hykes and Sebastien Pahl. The first Docker container was launched in 2011 before being released to the public in 2013. Since Docker was made public, it has become increasingly popular. It is an open platform for creating, delivering, and running applications. With this containerization technology, one can isolate applications from the infrastructure so that we can quickly deliver new versions of a given software. Docker uses a client--server architecture, which means that the Docker client communicates directly with the Docker daemon, which takes care of building, running, and distributing containers. The client and daemon can run on a single system, or it is possible to connect the Docker client to a remote daemon. The Docker client and daemon communicate using the REST API, UNIX sockets or the web interface. Another Docker client is Docker Compose, which directly enables working with applications that include multiple containers (Fava et al., 2024; Zeng et al., 2017; https://www.docker.com/resources/what-container/; https://www.geeksforgeeks.org/architecture-of-docker/).

Podman, unlike other containerization methods, does not have daemons, i.e. it does not require any continuously running processes in the background to manage the container. Instead, it uses system calls, which are made directly via the command line. With this rather non-standard solution, containers are run in the context of the current user, which is beneficial for security and isolation. It was created to facilitate the entire process of running, sharing, and deploying applications through containers and OCI (Open Container Initiative) container images. It relies on an OCI-compliant container execution environment to connect to the operating system and create running containers. It provides both a CLI command line and applications along with a graphical user interface (Similar to Docker Desktop). A very important point is that containers in a sub-master can be created and run by both an administrator and a normal user (in Docker, by default, Docker commands can be executed either from root or by assigning a user to a Docker group) (Zordevic et al., 2022).

Its history is closely linked to the evolution of container technologies and the need for more secure and flexible container management tools. It all started with

the development of Docker, which was introduced in 2013 and quickly revolutionized the way applications are created, deployed, and managed through containerization. Docker became the standard for container management, but it ran with a central daemon (dockerd), which raised some issues around security, stability, and permissions management. As production environments and the open-source community began to recognize these limitations, there was a need for a tool that could manage containers without a central daemon, offering greater security and flexibility. Red Hat, a company actively developing open-source technologies, took on this task. Thus was born the Podman project, which was created as part of a larger ecosystem of tools called libpod for managing containers without a central daemon. The first versions of Kata Containers Kata Containers is a container technology that combines the features of virtual machines and containers, providing isolation and security at the virtual machine level while retaining the lightness and flexibility of containers. It uses hypervisor-level virtualization to run each container in an isolated, dedicated virtual machine. Creating such dedicated container spaces provides a much higher level of security and isolation from the rest of the system. It is Open Container Initiative (OCI) compliant, which enables integration with other existing tools such as Kubernetes, for example. It runs on both Linux and Windows-based applications as well as various cloud environments (Poojara et al., 2018; Gamess, Parajuli, 2024).

Kata Containers is a solution that is ideal for deployments requiring a high level of security and isolation. Kata containers consist of several components that, working together, ensure that containers run securely inside nested VMs. The first component is the agent, which runs exactly inside the VM and is responsible for communicating with the container running in that VM. It is also responsible for container management, process start-up, network, and file system management. The second component is the Runtime. Runtime manages the lifecycle of containers running in the VM. There is integration with popular container orchestrators such as Docker Swarm and Kubernetes. The third component is the Hypervisor, which provides the virtualization layer on which lightweight VMs are run. Another is Shim, which acts as an intermediary between the runtime and the agent in the VM. It is responsible for passing signals and data between the processes running on the host machine and the agent in the VM. The last component is the Proxy. It can be used to manage communication between multiple containers running in a single VM (Gamess, Parajuli, 2024; Randazzo, ITinnirello, 2019; https://katacontainers.io/docs/; https://github.com/ kata-containers/kata-containers/blob/main/docs/Limitations.md).

LXC (Linux Containers) is a tool for creating and managing containers on Linux systems that offers lightness and isolation similar to virtual machines but with less resource overhead. Containers in LXC, thanks to the use of namespaces and cgroups, are fully isolated from each other and the host system, ensuring secure and stable operation. The namespaces mechanism isolates processes,

networks, the file system, and other resources, while cgroups control the consumption of resources such as CPU, memory, and I/O. This allows users to run multiple containers on a single host while retaining full control over the allocated resources. LXC is often used as a core technology in cloud and virtualization environments, enabling organizations to manage computing resources flexibly and efficiently. Through its architecture, LXC enables developers, administrators, and organizations to run applications in isolated containers, helping to increase the security and efficiency of software deployments (Poojara et al., 2018; https://kubernetes.io/pl/docs/concepts/overview/; https://kubernetes.io/docs/concepts/architecture/).

Docker Swarm was originally introduced by Docker, Inc. in response to the need to manage and orchestrate Docker containers in production environments. The first versions were developed as part of the Docker Engine project, allowing users to run and manage applications in containers on multiple nodes. As the Docker ecosystem evolved, Docker Swarm became more popular, offering a tool to scale and manage applications in an automated and efficient manner. Since Docker Engine 1.12, Swarm Mode has been integrated directly into Docker Engine, simplifying the process of running and managing Swarm clusters. Docker Swarm is popular among organizations that use Docker containers to deploy their applications in cloud and on-premises environments. With its ease of use, integration with Docker tools, and ability to scalability, Docker Swarm remains an essential tool in the container ecosystem, although competition in the container orchestration market has increased in recent years, with Kubernetes becoming the dominant platform in the field (https://azure.microsoft.com/pl-pl/resources/cloud-computing-dictionary/what-is-virtualization; https://www.docker.com/ resources/what-container/; https://kubernetes.io/pl/docs/concepts/overview/).

Kubernetes is a flexible open-source platform that enables control and management of applications and services running in containers. It allows configuration and automation of tasks in a declarative manner. Kubernetes is constantly evolving, offering a wide range of services, support, and tools. It was patented, developed, and made public by Google in 2014. Kubernetes supports application scaling, failure handling, and various deployment strategies. For example, with Kubernetes, it is easy to manage the roll-out of new software versions according to canary deployments, rolling updates, or blue/green deployments. With the creation of a cluster using the public cloud, one can very quickly create a workload ready for traffic from all over the world. Kubernetes is currently the most widely used tool for creating such environments and has a very good reputation. Kubernetes is a complex platform consisting of many components that help orchestrate containers and allow flexible container management, i.e., for example, maintaining a specific number of pods, or scaling them horizontally and vertically (https://kubernetes.io/pl/docs/concepts/overview/; https://kubernetes.io/docs/ concepts/architecture/).

Development of a test environment for examining individual containerization methods

In order to deploy the application, the Google Cloud public cloud was used, more specifically, the Compute Engine service. For this purpose, a Virtual Private Cloud was created, i.e., a network in the space of a given project in the cloud, and at the network level, a firewall was configured to allow the HTTP requests necessary to test the performance of the application, which will be generated using Apache JMeter. To carry out a meaningful experiment for the project, virtual machines with identical configurations (4 vCPUs and 16 GB of RAM) were created, and a WordPress deployment with a MySQL database was carried out on them on the various containerization methods. Each test using different containerization methods will use the same infrastructure shown in Figure 1. Only the containerization engine will be changed, i.e., Docker, Podman, LXC, and Kata Containers, respectively, for each test.

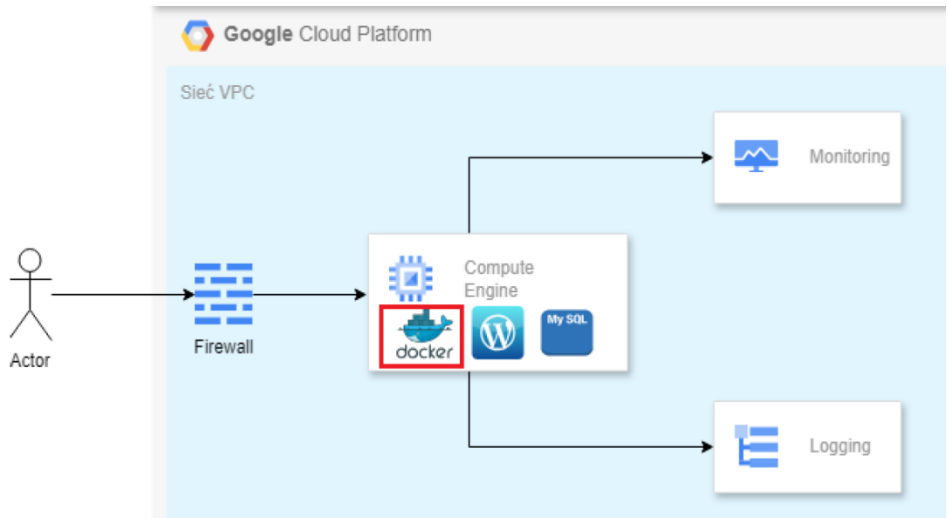

Figure 1. Implementation diagram of the test infrastructure

The Compute Engine service's built-in monitoring was used to verify the behavior of a given containerization method at times of increased traffic. To access more advanced metrics, an agent (Ops Agent) was additionally installed on the VM, which was responsible for collecting and sending metrics to Google Cloud. Google Cloud provides very good monitoring. In real-time, it shows a number of graphs responsible for the individual resources of individual virtual machines, such as, for example, CPU utilization level, RAM utilization, disk metrics, or network utilization metrics. Some examples of [%] metrics for Docker are shown in Figure 2 and Figure 3.

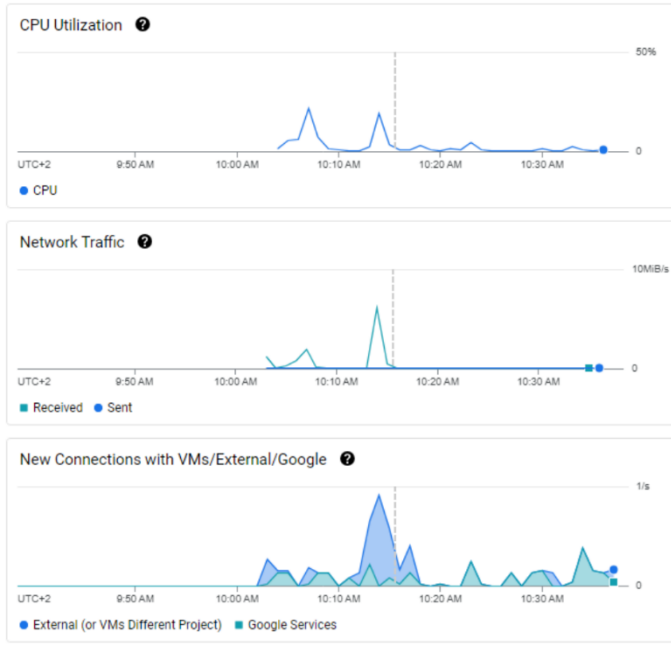

Figure 2. CPU utilization [%] and network traffic metrics for Docker

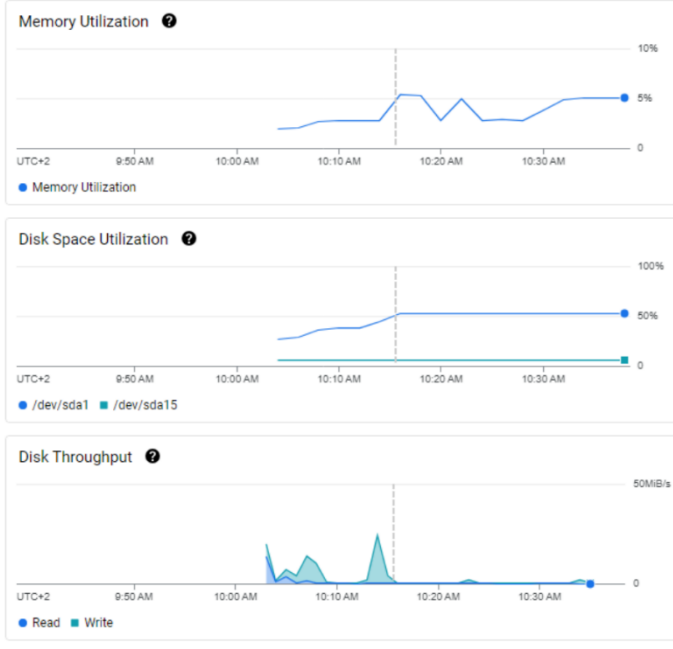

Figure 3. Memory and disk space utilization [%], disk throughput metrics for Docker

Comparative analysis of selected containerization methods

Load tests were carried out to evaluate the methods described. A machine with a 4vCPU and 16 GB RAM configuration was chosen. Any 'larger' machine would have coped better with the traffic simulated in this project, but the configuration most commonly used in real-world conditions was chosen. The simulations started by assuming extremely low parameters. A duration of one second and a traffic volume of 1,000 requests for 10 repetitions is unrealistic for infrastructures that base their production environment on single VMs in such a configuration, and CPU consumption values can reach almost 100%. In tests, the duration increases. The test results for each containerisation type in turn are summarised in Table 1–4. The results can be used as a reference for further simulations.

Table 1. Docker performance results

Number of requests	Test duration [s]	CPU usage [%]	Memory usage [%]
100 x 5	1	14.01	6.20
100 x 10	1	28.01	6.63
1000 x 5	1	63.14	9.58
1000 x 10	1	93.62	10.17
1000 x 10	5	95.29	11.27
1000 x 15	5	99.62	13.11

Table 2. Kata Containers performance results

Number of requests	Test duration [s]	CPU usage [%]	Memory usage [%]
100 x 5	1	21.52	23.12
100 x 10	1	25.62	25.62
1000 x 5	1	87.32	31.25
1000 x 10	1	98.14	30.52
1000 x 10	5	99.48	32.96
1000 x 15	5	99.99	34.67

Table 3. Podman performance results

Number of requests	Test duration [s]	CPU usage [%]	Memory usage [%]
100 x 5	1	12.05	3.95
100 x 10	1	27.95	4.07
1000 x 5	1	61.57	6.23
1000 x 10	1	91.61	7.38
1000 x 10	5	97.46	9.18
1000 x 15	5	99.38	11.59

Table 4. LXC performance results

Number of requests	Test duration [s]	CPU usage [%]	Memory usage [%]
100 x 5	1	3.74	15
100 x 10	1	4.59	15.23
1000 x 5	1	78.77	20.2
1000 x 10	1	99.27	22.43
1000 x 10	5	99.99	26.49
1000 x 15	5	99.99	30.21

In the containerization methods used, significant differences can be seen between both container start-up and stopping speeds, server resource consumption, and container security. Comparative results for performance tests of CPU consumption and RAM consumption are shown in Figure 4 and Figure 5.

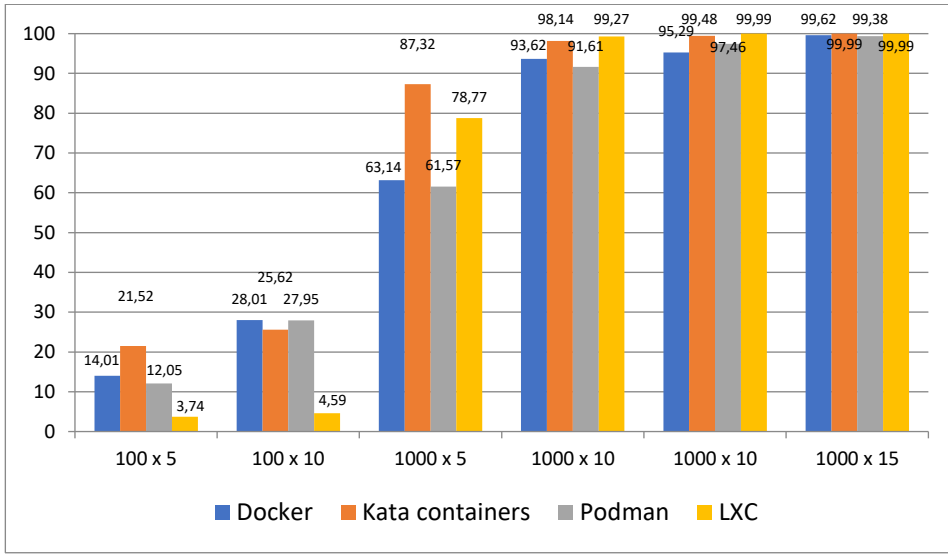

Figure 4. Performance test results for CPU consumption [%]

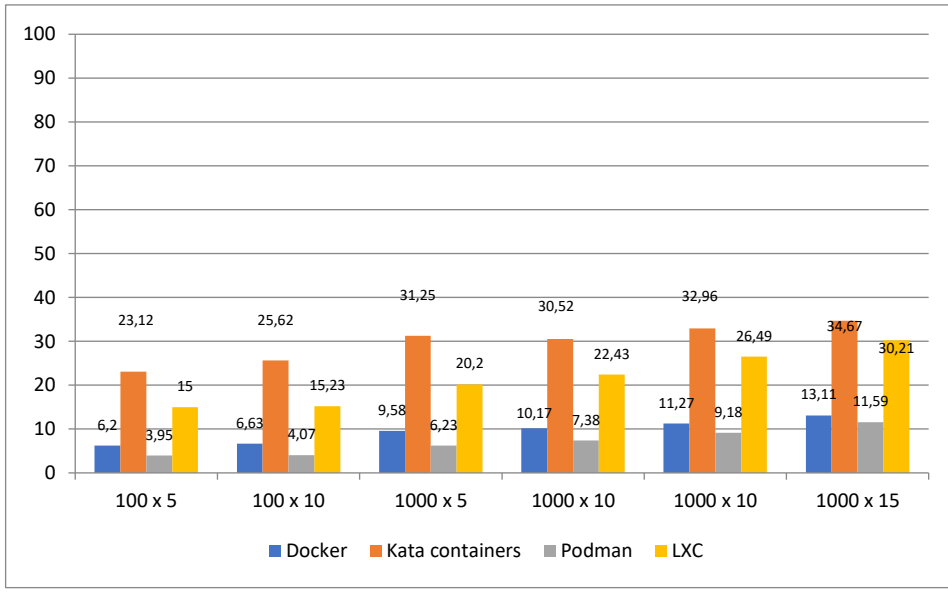

Figure 5. Performance test results for RAM consumption [%]

Analysis of the performance tests showed that the containerization method that used the fewest resources to handle the given amount of server load was Podman, followed by Docker, LXC, and Kata. When it comes to the speed of starting and stopping containers, Docker comes first, followed by Podman, LXC, and Kata. It is worth noting that Kata used by far the most resources of the host machine and, with the increasing number of containers, could, despite its highest level of security, prove to be a poor solution when it comes to, for example, hosting a web service. LXC has a very high RAM load, but with fewer requests per publicized service, it was by far the best performer in terms of CPU usage. With more requests, the CPU usage increased significantly, which may mean that LXC is not as well-optimized for high traffic (like Docker or Podman). It may find use, for example, in internal organizational services. The big disadvantage of LXC is that it is not possible to use ready-made images from the public Docker Hub repository. When creating a service using LXC, you have to base it on a Linux operating system image and prepare the entire configuration yourself inside an empty container. LXC is also the least user-friendly for new and novice users due to its complex networking solution; the user themselves has to ensure that the service is properly publicized through appropriate forwarding rules, whereas with other containerization methods, the whole thing is done automatically. Both Docker and Podman, on the other hand, are well suited for the deployment of publicized services, due to their fast creation and setup times for new containers. They are suitable for horizontal scaling and vertical scaling and are great at preparing an organization's infrastructure for sudden increases in traffic and server load. For this reason, they are probably the most commonly used containerization method, together with Kubernetes, for the deployment of services that are prepared for global, regional, or general public traffic.

Both Docker and Podman have several integrated tools that can greatly facilitate the work of all users using these containerization methods. Performance tests aside, it should also be noted that they performed best with incoming traffic. The key point, however, is that despite their long list of advantages, both of these containerization methods are nowhere near Kata Containers in terms of security. All of these containerization methods have their advantages as well as their disadvantages, and their implementation in an actual project should be determined by the respective requirements for the service to be implemented.

Conclusion

The research presented here has shown that choosing the right containerization method depends on the service to be implemented. All the methods described have both advantages and disadvantages. If you want to deploy a public service that is prepared for heavy traffic, then Docker or Podman would be the best choice. However, focusing on providing a high level of security at the expense of

performance and overall resource consumption, Kata Containers is the best solution. For internal organizational services based on the Linux operating system, LXC containers are worth considering.

All of these methods work very well, sticking to their priorities. However, it is worth emphasizing that all containerization methods by themselves may not be sufficient for all organizations. If a company starts to grow dynamically and would like to exploit the full potential of containers and prepare its infrastructure for a potentially sudden large surge in service load, it would need to focus its attention on a container orchestrator. Containers combined with Kubernetes (or Docker Swarm) and the public cloud can significantly improve the performance of any infrastructure, both in terms of security, performance, and overall cost per infrastructure, while providing dynamic vertical scaling of servers (Worker Nodes) and application instances (Pods). Introducing such a concept will allow a given infrastructure to be prepared for massive traffic (even global) while maintaining a very good level of service availability and reliability, and customer satisfaction.

The tests carried out allow recommendations to be made for the use of the methods described. The Docker and Podman platforms are worth using for web applications, containers containing applications, and all dependencies that can be easily deployed through the CI/CD process. They can work well for creating isolated test environments.

LXC is the ideal choice when the user needs to create a fully isolated operating system and HPC (High-Performance Computing) environment. For high--performance computing, LXC can provide low virtualization overhead, which is crucial in HPC.

Kata Containers will find its use in the deployment of sensitive applications by creating containers in virtual machines (nested virtualization). It provides great security, so it can find its use in applications where isolation and security are the highest priority.

The conducted research provides a valuable educational foundation for students and professionals aiming to deepen their understanding of containerization technologies. By working with real-world tools such as Docker, Podman, LXC, and Kata Containers in a cloud-based testing environment, learners gain practical experience in deploying, configuring, and benchmarking web services. The use of Apache JMeter for load testing and the analysis of system metrics through Google Cloud monitoring equips participants with essential skills in performance evaluation, infrastructure planning, and decision-making in IT systems. This hands-on approach encourages active learning and bridges the gap between theoretical knowledge and professional practice.

From a didactic perspective, the project fosters critical thinking and problem--solving by presenting real implementation scenarios with measurable outcomes. It highlights the importance of selecting the right containerization method based

on specific service requirements such as performance, scalability, or security. Furthermore, it introduces learners to modern DevOps practices, CI/CD pipelines, and container orchestration tools like Kubernetes, which are essential in today's IT landscape. As such, this work can be effectively integrated into university courses, workshops, or training programs focused on cloud computing, system administration, and application deployment strategies.

Acknowledgments

We are grateful to Mikołaj Piatek, a graduate student at Rzeszów University of Technology, for supporting us in collecting useful information.

References

Fava, F.B. et al. 92024). *Assessing the Performance of Docker in Docker Containers for Micro-service-based Architectures*. Euromicro Conference on Parallel, Distributed and Network--Based Processing.

Gamess, E., Parajuli, M. (2024). *Containers Unleashed: A Raspberry Pi Showdown Between Docker, Podman, and LXC/LXD*. IEEE SoutheastCon Conference on Engineering the Future.

https://azure.microsoft.com/pl-pl/resources/cloud-computing-dictionary/what-is-virtualization (11.06.2025).

https://cloud.google.com/logging/docs/agent/ops-agent (11.06.2025).

https://github.com/kata-containers/kata-containers/blob/main/docs/Limitations.md (11.06.2025).

https://github.com/kata-containers/kata-containers/tree/main/docs/design/architecture (11.06.2025).

https://katacontainers.io/docs/ (11.06.2025).

https://kubernetes.io/blog/2020/12/08/kubernetes-1-20-release-announcement/ (11.06.2025).

https://kubernetes.io/docs/concepts/architecture/ (11.06.2025).

https://kubernetes.io/pl/docs/concepts/overview/ (11.06.2025).

https://www.docker.com/resources/what-container/ (11.06.2025).

https://www.geeksforgeeks.org/architecture-of-docker/ (11.06.2025).

Poojara, S.R., Ghule, V.B., Birje, M.N. et al. (2018). *Performance Analysis of Linux Container and Hypervisor for Application Deployment on Clouds*. 1st International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS).

Randazzo, A., Tinnirello, I. (2019). *Kata Containers: An Emerging Architecture for Enabling MEC Services in Fast and Secure Way*. 6th International Conference on Internet of Things – Systems, Management and Security (IOTSMS).

Zeng, H., Wang, B.S., Deng, W.P., Zhang, W.Q. (2017). *Measurement and Evaluation for Docker Container Networking*. International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC).

Zordevic, B., Timcenko, V., Lazic, M., Davidovic, N. (2022). *Performance comparison of Docker and Podman container-based virtualization*. 21st International Symposium on INFOTEH--JAHORINA (INFOTEH).