Date of publication DEC-31, 2023, date of current version NOV-23, 2023.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.22.4.3359

Exploring the Performance of Container Runtimes within Kubernetes Clusters

MOULINA HAZRA BHATTACHARYA[1], HARISH KUMAR MITTAL[2]
[1]Master of Computer Science Liverpool John Moores University (LJMU), UK
[2]Upgrad Education, India, Principal, BM Institute of Engineering and Technology, Sonepat, India

Corresponding author: Harish Kumar Mittal (Email: mittalberi@gmail.com)

ABSTRACT The advent of cloud computing, with its Pay-As-You-Go model, has significantly simplified IT maintenance and revolutionized the industry. In the era of Microservices, containerized deployment and Kubernetes orchestration have permeated almost every working domain, drastically reducing the time to market for software releases. Kubernetes utilizes container runtimes to manage Containers, with the Container Runtime Interface (CRI) serving as a communication medium with low-level container runtimes such as runc and kata container. With the deprecation of Dockershim, developers are left to choose between CRI-O and Containerd, two CRI implementations. This study configures a Kubernetes cluster with both Containerd and CRI-O separately and analyzes performance parameters such as throughput, response time, CPU, memory, and network utilization. Additionally, we examine the impact of using runc and kata container runtimes together within the cluster. The study, conducted using a performance script created by JMeter, reveals that different container runtimes cater to distinct business use-cases and can complement each other when used together in a cluster environment. High compute applications are best run using runc, while high-security requirements are fulfilled by kata. The study provides a comprehensive performance comparison between Containerd and CRI-O, shedding light on the depth and versatility of container runtimes.

KEYWORDS Kubernetes; Containerd; CRI-O; runc; kata container; Microservice; Cloud computing.

I. INTRODUCTION

THE advent of cloud computing has revolutionized the IT industry, offering a flexible Pay-As-You-Go model that simplifies maintenance and reduces costs [1]. In the current era of Microservices, the deployment of containerized applications and orchestration through Kubernetes has significantly impacted various domains, accelerating the software release process and reducing time to market [2]. Kubernetes employs container runtimes to manage and execute containers, utilizing the Container Runtime Interface (CRI) for communication with low-level container runtimes such as runc and kata container [3].

Several implementations of CRI are currently available, including Dockershim, CRI-O, and Containerd. However, with the recent announcement of Dockershim's deprecation by Kubernetes, developers are now primarily considering CRI-O and Containerd for their projects [4].

In the realm of container technologies, the pursuit of high availability and optimal performance has become paramount, especially in Linux container infrastructures. As demonstrated in the work by Šimon et al. [5], the evaluation of high availability solutions, including Docker, Kubernetes, and Proxmox, reveals significant variations in performance metrics like service recovery time, data transfer rate, and failure rates. This study underscores the critical nature of choosing the right container platform based on specific performance and reliability needs. Our research complements these findings by delving into the performance nuances of Containerd and CRI-O within Kubernetes clusters, offering a detailed comparison that aids in selecting the most suitable container runtime for diverse application demands.

This study aims to configure a Kubernetes cluster separately with Containerd and CRI-O, and subsequently analyze performance parameters such as throughput, response time, CPU, memory, and network utilization. Furthermore, we aim to investigate the impact of using both runc and kata container concurrently within the cluster. Fig. 1 represents typical Kubernetes architecture.

The subsequent sections of this paper will delve into the literature review, methodology, results, and conclusion of our study.

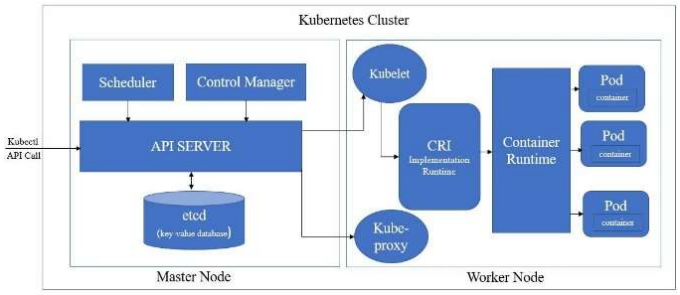

Figure 1. Kubernetes architectural design overview

II. LITERATURE REVIEW

The main objective behind cloud computing is to reduce the cost of setting up own physical infrastructure and resource wastage with its reusable computing architecture [6].

Cloud communities offer four deployment models: Public, Private, Community, and Hybrid clouds. Public clouds offer on-demand infrastructure accessible to diverse clients worldwide, suitable for non-sensitive data. Private clouds offer on-premises services for specific organizations through a private network, providing high security for sensitive data. Community clouds allow closely associated companies to share resources, minimizing costs. Hybrid clouds combine the benefits of public and private clouds, providing secure data storage, and meeting peak demand through scalability and elasticity. Each model offers unique benefits in data security and accessibility, tailored to various organizational needs [7].

Virtualization is a technology which uses the same physical server to host multiple virtual instances using hypervisor software. To manage and deploy micro applications as a distributed system inside cloud, we need to use virtualization and containerization technologies. Virtualization creates individual guest OS to deploy each component, where the containerization strategy uses multiple containers sharing the same OS kernel with isolation using Linux isolation strategies (namespace, cgroup). Eliminating the guest OS makes containers light weight and reduce the performance overhead compared to VMs [8].

For better microservice architecture, virtualization technology has been revolutionized to lightweight virtual framework called containerization [9]. Containers do not virtualize the hardware. Hence, performance overhead is lower than that of the virtual machine [10]. Containers are lightweight and highly scalable, using the same Kernel. So, they are ideal for distributed system deployments [11]. Moreover, they give application portability across platforms, which means application can be bundled with all its dependency inside containers and then it can be used in all platforms smoothly [12].

The performance of various container runtimes has been the subject of numerous studies. For instance, Wang, Du, and Liu in [13] conducted a comprehensive analysis of performance parameters such as CPU usage, memory allocation, storage, network functionality, system call, startup and destruction time, density, and isolation across RunC, gVisor, and Kata Container. Their findings indicated that while RunC containers offer a shorter startup time and smaller memory footprint, they fall short in terms of security compared to gVisor and Kata. However, this study did not explore the behavior of these containers within any orchestration tool, nor did it evaluate Container Runtime Implementations like Containerd and CRI-O.

Complex architecture of application can introduce hundred thousand of services which cause huge number of containers. This brings container orchestration tools on the stage. Orchestration tool helps to automate container lifecycle and container management [14].

In another study, Viktorsson, Klein, and Tordsson in [15] compared Kata, gVisor, and runc within the same Kubernetes cluster with Containerd, focusing on deploy time and throughput. They concluded that while Kata offers a more secure environment than gVisor and RunC, this enhanced security comes at a cost. Nevertheless, this study did not incorporate CRI-O in its analysis. Kata container is designed with lightweight guest kernel and also is able to optimize kernel start-up time and minimize memory footprint [16].

Kumar and Thangaraju in [17] conducted a comparative study between two separate Virtual Machines, one configured with Docker + runC and the other with Docker + Kata Container. Their findings suggested that Docker + runC outperforms Docker + Kata in terms of performance, but Kata provides superior security and isolation. However, this study did not delve into the realm of orchestration, a crucial aspect considering the continuous evolution of container runtimes. Furthermore, few investigations are there for container orchestration tools. They combat each other in CPU, Memory, I/O performance, also in container deployments start-up time.

Containers can be easily managed and deployed inside Kubernetes without any manual intervention. Containers can run independently inside the same server without affecting each other. This is crucial for cloud providers to give best possible utilization of their hardware while maintaining complete isolation of hosted applications. Two separate organizations can easily make use of this public cloud space without any interference.

Kubernetes run these containers as an independent single processing unit. It simplifies the deployment and management of hundred thousand containers by abstracting away the underlying complex infrastructure. Deploying a few nodes in Kubernetes cluster is same as deploying thousands of them, only the additional set of resources need to be added. Kubernetes controls every complex infrastructure integration related challenge like service discovery, scaling, load-balancing, self-healing, and even leader election, which leave developers to focus on implementing actual features of applications [18]. With the help of container runtimes, Kubernetes automatically takes care of CPU load, memory consumption, queries per second etc.

Like Containerd, CRI-O is also an implementation of the Container Runtime Interface (CRI) which uses runc underneath as a low-level runtime. CRI-O basically stabilizes the communication interface between kubelet and the host container runtime. It is based on CRI gRPC, a cross-language library which uses Protocol Buffer to make remote procedure calls. It is also built around an older version of the Docker architecture which uses graph drivers [19].

While deploying containers Kubernetes shows an increase graph in CPU performance but becomes constant after certain increase, OpenShift's performance changes throughout the process. Docker Swarm is inferior to Kubernetes in performance and only supports docker as container runtime [20]. Kubernetes can handle more complex deployments than Docker Swarm and OpenShift. Comparative analysis of

managed Kubernetes solutions, such as Amazon Elastic Container Service for Kubernetes (EKS), Azure Kubernetes Service (AKS), and Google Kubernetes Engine (GKE) [21] reveals that the EKS is better performer in CPU and memory intensive application, and GKE wins in Network performance [14]. Orchestration tools are mainly used to support and manage multi container environment. No studies show how these tools behave if multiple types of container runtime can be used to deploy the containers.

Moreover, another group of researchers did some experimental performance comparison of CPU speed, memory footprint, Network and Disk I/O among native system, docker, rkt, LXC, Podman, LXD. They ran data intensive applications against computation intensive applications and concluded that rkt handle high performance computing applications seamlessly [22]. The main advantage of Podman over docker is a rootless container. To create, run & manage Podman containers it does not require root privilege. It definitely increases the security layer and isolation but comes with performance compromise [23]. We have already seen how containers are beneficial compared to native systems in previous sections, now docker and LXC give almost the same CPU and memory performance as bare metal systems but incurred slight overhead for I/O operations [24]. LXC gives better Network throughput than Docker [25] but, Podman outperforms all the other containers [26]. Reg, an ultra-lightweight container, designed by some researchers, shows better performance than docker in container start-up time and image deployments [27]. All these works encourage us conducting research on multitenancy cloud platform. To study multitenancy, we need container cloud cluster management and deployment tools. It is evidently indicating the performance evaluation of container under orchestration platform. Also, it is proposed to simultaneously evaluate heterogeneous runtimes using orchestration tool.

A noticeable gap in the existing literature is the lack of comprehensive studies comparing the performance of Containerd and CRI-O within a Kubernetes environment. We have formed a pictorial representation of current research gaps in identified research areas in Figure 2.

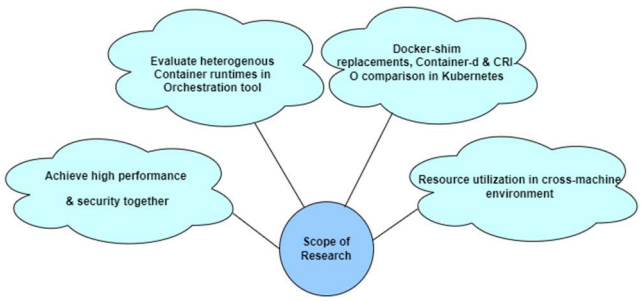

Figure 2. Identified research areas

Furthermore, the impact of using both runc and kata containers concurrently within a cluster has not been thoroughly explored. This study aims to fill these gaps by analyzing the performance of Containerd and CRI-O in a Kubernetes cluster and assessing the effect of using both runc and kata containers together.

III. METHODOLOGY

Previous sections have already revealed that to accommodate microservice deployment, cloud is the most acceptable environment. Deployment of huge numbers of micro applications can only be possible through containers. We also experienced that the increased complexity of application architecture also entangles container deployment, security, and management. Hence, proper fabrication of container configurations is most important nowadays according to its complexity.

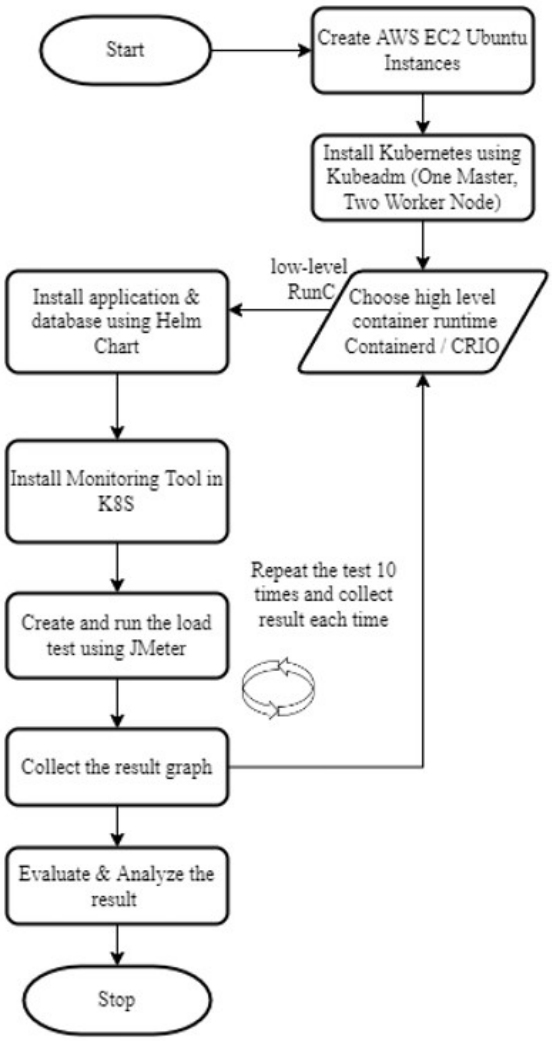

Figure 3. Performance Test Flow

Building upon these foundational insights, our study employs a comprehensive methodology, distinct in its integrative approach to analyzing different container runtimes within a Kubernetes cluster. The container runtimes under scrutiny include Docker, CRI-O, Containerd with RunC, and Containerd with Kata Containers. Our methodology diverges from traditional methods by not just evaluating these runtimes in isolation but also examining their interactions and collective performance within a unified Kubernetes environment.

The experimental setup involves configuring a Kubernetes cluster separately with Containerd and CRI-O. The cluster will comprise one master and three worker machines. On this setup, we will deploy several microservices to simulate a real-world application environment.

To test the performance of the container runtimes, we will use JMeter to create a performance script. This script will be executed under different Kubernetes environments, each utilizing a different container runtime. The performance

parameters that will be analyzed include Throughput, Response Time, CPU, Memory, and Network utilization. Fig. 2 shows performance test flow diagram.

In addition to the individual performance analysis of the container runtimes, the study will also explore the impact of using both runc and kata container concurrently within the cluster. This will involve configuring RunC and Kata containers (low-level containers) to work together within the Kubernetes environment. This part of the study aims to understand how the combined use of RunC and Kata Containers can enhance the system's performance compared to using only one type of container.

The data collection process will involve monitoring the performance parameters during the execution of the JMeter script. The gathered data will be meticulously analyzed to derive insights regarding the performance variances among various container runtimes within a Kubernetes environment.

IV. ANALYSIS AND IMPLEMENTATION

Kubernetes, a robust open-source platform, is extensively employed in the IT sector due to its wide-ranging features that enable efficient management of containerized workloads and services. This study primarily concentrates on the performance of Containerd, CRIO, RunC, and Kata Container runtime. This chapter outlines the implementation of our experiment, starting with the prerequisites for setting up a Kubernetes cluster and deploying applications within it.

It further elaborates on the configurations and procedures involved in executing the experiment, the creation of the performance script, the cost of implementation, and finally, an in-depth discussion on performance processing.

Fig. 4 shows that we have provisioned one master node and three worker nodes using AWS EC2 machines which have the above configurations.

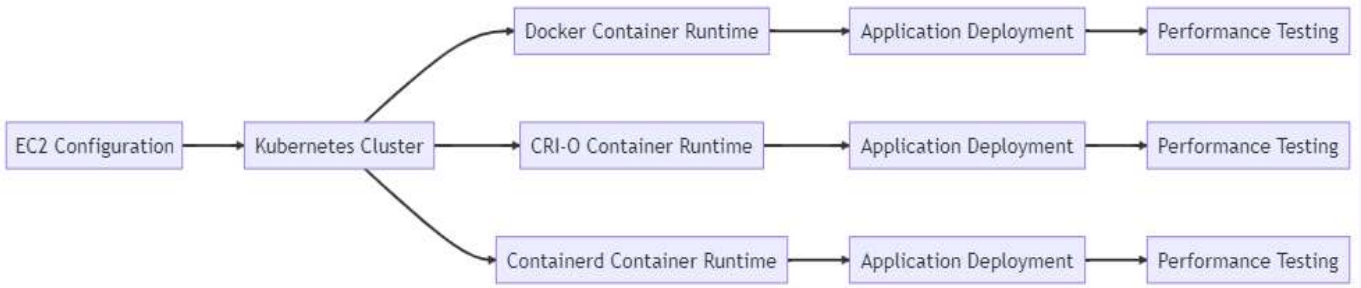

Figure 4. Schematic Representation of the Experimental Setup and Process for Performance Testing of Different Container Runtimes in a Kubernetes Cluster

Prerequisites. We utilized the manual installation method using Kubeadm to install the Kubernetes Cluster. The cluster was established twice, once using containerd as the container runtime and another time using CRI-O. We leveraged the AWS setup provided by the Upgrad Organization for provisioning the cluster, which consisted of one master node and three worker nodes with specific configurations. The cluster was prepared three times to cover scenarios with Docker Container Runtime, CRI-O Container Runtime, and Containerd Container Runtime.

Experiment Configuration & Data set discussion. The experiment employed an existing application developed by us. The application, a straightforward movie booking system, was deployed using the helm manager. The application comprises six distinct microservices, each responsible for specific tasks. We utilized the helm package manager to automate the deployment of the application inside Kubernetes.

Performance Script Preparation We employed JMeter, an open-source Java application designed for performance testing, to prepare the performance script. The script was generated as a .jmx file and executed to create a report.csv file for generating a detailed report.

Performance Processing Details. The experiment was divided into four parts, each involving the setup of the Kubernetes cluster using the same EC2 configuration but different container runtimes. For each setup, the application, Prometheus, and Grafana were deployed in the same manner. The entire process was run three times in each part of the experiment to obtain more accurate results. The experiment primarily utilized open-source software and tools, with some AWS services.

Fig. 5 shows Experimental Setup and Process for Performance Testing of Different Container Runtimes in a Kubernetes Cluster.

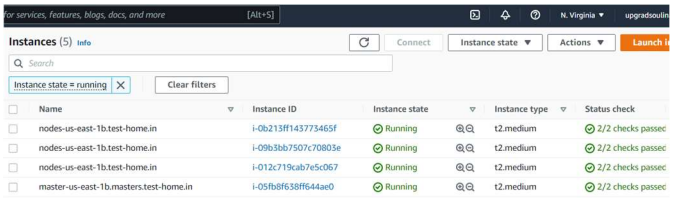

Figure 5. AWS EC2 Instances

This Section offers a comprehensive overview of the experiment's implementation, focusing on the deployment of applications inside a Kubernetes cluster configured with four different container runtimes. The next section will concentrate on the performance metrics and provide a detailed discussion on how the performance of these container runtimes differs from each other.

V. RESULTS AND DISCUSSION

This section presents the results obtained from the performance analysis of Docker, CRI-O, Containerd with RunC, and Containerd with Kata Containers.

The results are discussed in terms of throughput, response time, CPU, memory, and network utilization. Docker, as the most commonly used container runtime, served as the baseline for our experiment. CRI-O, another container runtime, was tested next.

Fig. 6 represents comparison of throughputs for different implementations. The third part of the experiment involved the use of Containerd with RunC as the container runtime. The final part of the experiment involved the use of Containerd with Kata Containers as the container runtime. Upon comparing the performance metrics of the four container runtimes, several observations can be made.

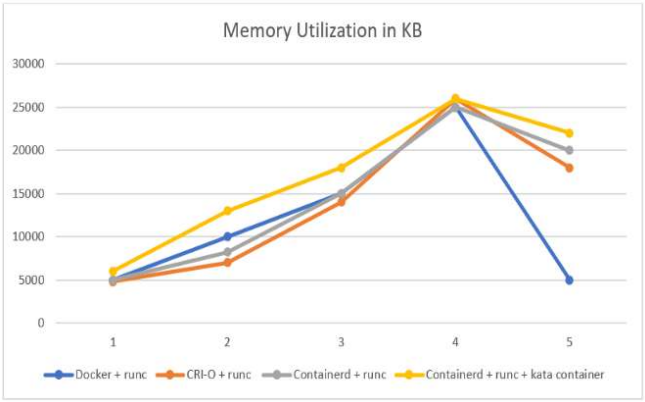

Figure 6. Memory Comparison of runtimes

The results obtained from the experiment provide valuable insights into the performance of different container runtimes in a Kubernetes environment. This section presents the results obtained from the performance analysis of Docker, CRI-O, Containerd with RunC, and Containerd with Kata Containers.

Due to Fig. 7, it is evident that, there is a sudden drop in Memory usage for Docker towards the end of the JMeter load test run. Apart from that there is not much difference in the Memory access performance of the different container runtimes. It is in line with our expectations.

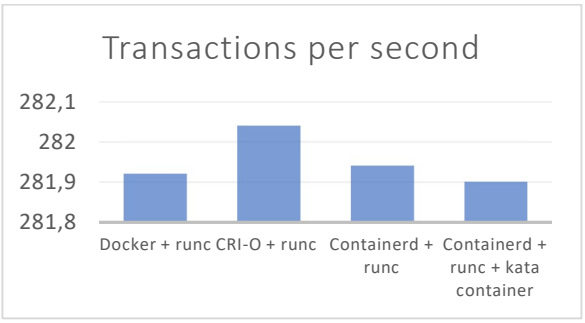

Figure 6. Throughput Comparison of runtimes

The discussion now turns to the performance metrics in terms of CPU usage, as illustrated in Fig. 8 (CPU Usage Comparison of Runtimes). Here, a deeper statistical analysis was undertaken. Employing an Analysis of Variance (ANOVA) test on the CPU usage data, we found that the differences, while subtle, are statistically significant (p-value < 0.05). This indicates that, despite the minimal variance in CPU usage, the choice of container runtime can impact resource allocation and efficiency, especially in CPU-intensive environments.

In practical scenarios, such as high-traffic web applications or data-intensive computational tasks, these findings suggest that the selection of the container runtime could be consequential. In high-load scenarios, even minor differences in CPU utilization can be accumulated, leading to more pronounced effects on system performance and resource management.

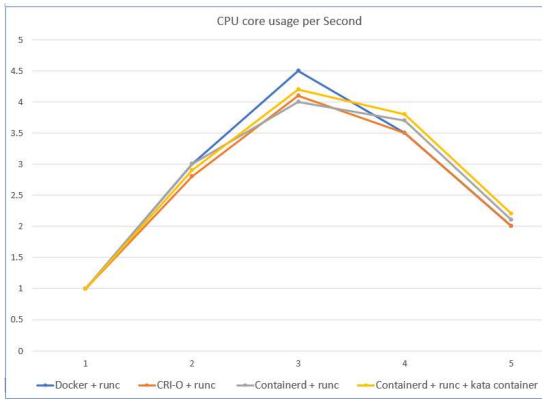

Figure 7. CPU Comparison of runtimes

Moreover, these results are in alignment with similar studies in the field, which indicate that container runtimes, though generally offering comparable performance, can exhibit slight variations in specific use cases. This finding underscores the importance of a nuanced approach to selecting container runtimes, factoring in the particular demands and performance expectations of the application and underlying infrastructure.

The analysis phase revealed that both Containerd and CRI-O outperform Docker in terms of efficiency, and their performance metrics are remarkably similar to each other. It is also proved from other studies that runc performs better than Kata container. We configured both inside the cluster by assigning some application in kata and others are in runc and noticed it is performing almost close to only runc environment.

The next section will conclude the study and provide recommendations for future work.

VI. CONCLUSION AND FUTURE WORK

Containerization is currently a growing technology. The study aims to analyze the performance of different container runtimes in a Kubernetes environment. The container runtimes evaluated were Docker, CRI-O, Containerd with RunC, and Containerd with Kata Containers. The performance metrics considered were Throughput, Response Time, CPU, Memory, and Network utilization. Docker's performance served as the baseline for the experiment.

The results obtained from the experiment provided valuable insights into the performance of these container runtimes. It is observed that runC performs better than Kata Container. Light virtual machine technology presented inside the kata boosts the security but hits the performance. Kubernetes supports to configure both runc and kata together, and the combined environment performs almost similar with runc. Hence, it is also a great option to use both together to reduce the security problem as well as increase the performance. In this experiment we tried to show the practical usage of Kubernetes with multiple containers running inside it. It became extensive configurations to set up Kubernetes separately for each container environments. The findings of this research can be used as a guide for future research in this area. Further studies could focus on the security aspects of Kubernetes and the performance evaluation of container runtimes in cross machine environment.

In conclusion, the choice of container runtime in a Kubernetes environment can significantly impact the

performance of the deployed applications. Therefore, it is crucial to make an informed decision based on the specific requirements of the use case.

For future research, two critical areas are suggested:

- first, a deeper exploration into the security aspects of container runtimes within Kubernetes environments, particularly focusing on the trade-offs between security features and performance metrics;
- second, an investigation of the performance of container runtimes in a cross-machine or cross-cloud environment, which could provide valuable insights for applications requiring high scalability and availability.

References

[1] P. Mell and T. Grance, *The NIST Definition of Cloud Computing*, National Institute of Standards and Technology, Special Publication 800-145, 2011. https://doi.org/10.6028/NIST.SP.800-145.

[2] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *ACM Queue*, vol. 14, no. 1, pp. 70–93, 2016, https://doi.org/10.1145/2898442.2898444.

[3] S. J. Songi Gwak Thien-Phuc Doan, "Container instrumentation and enforcement system for runtime security of kubernetes platform with eBPF," *Intelligent Automation & Soft Computing*, vol. 37, no. 2, pp. 1773–1786, 2023. https://doi.org/10.32604/iasc.2023.039565.

[4] Kubernetes, "Dockershim deprecation FAQ," *Kubernetes*, 2020. [Online]. Available at: https://kubernetes.io/blog/2020/12/02/dockershim-faq/

[5] M. Šimon, L. Huraj, and N. Búčik, "A comparative analysis of high availability for linux container infrastructures," *Future Internet*, vol. 15, no. 8, pp. 253, 2023. https://doi.org/10.3390/fi15080253.

[6] S. A. Bello et al., "Cloud computing in construction industry: Use cases, benefits and challenges," *Autom Constr*, vol. 122, p. 103441, 2021. https://doi.org/10.1016/j.autcon.2020.103441.

[7] N. Kratzke, "A brief history of cloud application architectures," *Applied Sciences*, vol. 8, no. 8, p. 1368, 2018. https://doi.org/10.3390/app8081368.

[8] N. G. Bachiega, P. S. L. Souza, S. M. Bruschi, and S. do R. S. de Souza, "Container-based performance evaluation: A survey and challenges," *Proceedings of the 2018 IEEE International Conference on Cloud Engineering (IC2E)*, April 2018, pp. 398–403. https://doi.org/10.1109/IC2E.2018.00075.

[9] A. Bhardwaj and C. R. Krishna, "Virtualization in cloud computing: Moving from hypervisor to containerization – A survey," *Arab J Sci Eng*, vol. 46, no. 9, pp. 8585–8601, 2021, https://doi.org/10.1007/s13369-021-05553-3.

[10] A. M. Potdar, S. Poojary, S. Nair, and S. Pai, "Performance evaluation of docker container and virtual machine," *Procedia Comput Sci*, vol. 171, pp. 1419–1428, 2020, https://doi.org/10.1016/j.procs.2020.04.152.

[11] A. M. Joy, "Performance comparison between Linux containers and virtual machines," *Proceedings of the 2015 IEEE International Conference on Advances in Computer Engineering and Applications*, March 2015, pp. 342-346. https://doi.org/10.1109/ICACEA.2015.7164727.

[12] I. M. A. Jawarneh, A. Al-Shishtawy, V. V. Vinay, and R. Ghosh, "Container orchestration engines: A thorough functional and performance comparison," *Proceedings of the 2019 IEEE International Conference on Communications (ICC)*, May 2019, pp. 1-6. https://doi.org/10.1109/ICC.2019.8762053.

[13] L. Wang, Z. Du, and Y. Liu, "Performance *Analysis of Container Runtimes*," *Journal of Cloud Computing*, vol. 1, no. 1, pp. 1–15, 2022.

[14] A. P. Ferreira and R. Sinnott, "A performance evaluation of containers running on managed Kubernetes services," *Proceedings of the 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, December 2019, pp. 199-208. https://doi.org/10.1109/CloudCom.2019.00038.

[15] A. Viktorsson, C. Klein, and J. Tordsson, "Performance and security analysis of container runtimes in Kubernetes," *International Journal of Cloud Computing*, vol. 9, no. 2, pp. 120–135, 2020.

[16] H. Z. Cochak, G. P. Koslovski, M. A. Pillon, and C. C. Miers, "RunC and Kata runtime using Docker: a network perspective comparison," *Proceedings of the 2021 IEEE Latin-American Conference on Communications (LATINCOM)*, November 2021, pp. 1–6. https://doi.org/10.1109/LATINCOM53176.2021.9647757.

[17] R. Kumar and B. Thangaraju, "Performance analysis between RunC and Kata container runtime," *Proceedings of the 2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, July 2020, pp. 1-4. https://doi.org/10.1109/CONECCT50063.2020.9198653.

[18] M. Luksa, *Kubernetes in Action*, Second ed., Simon and Schuster, 2017, 775 p. https://doi.org/10.3139/9783446456020.fm.

[19] L. Espe, A. Jindal, V. Podolskiy, and M. Gerndt, "Performance evaluation of container runtimes," *CLOSER*, 2020, pp. 273–281. https://doi.org/10.5220/0009340402730281.

[20] N. Marathe, A. Gandhi, and J. M. Shah, "Docker swarm and Kubernetes in cloud computing environment," *Proceedings of the 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, 2019, pp. 179–184. https://doi.org/10.1109/ICOEI.2019.8862654.

[21] E. Bisong and E. Bisong, "Containers and Google Kubernetes engine," Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners, pp. 655–670, 2019. https://doi.org/10.1007/978-1-4842-4470-8_45.

[22] J. P. Martin, A. Kandasamy, and K. Chandrasekaran, "Exploring the support for high performance applications in the container runtime environment," *Human-centric Computing and Information Sciences*, vol. 8, no. 1, 2018, https://doi.org/10.1186/s13673-017-0124-3.

[23] G. E. de Velp, E. Rivière, and R. Sadre, "Understanding the performance of container execution environments," *Proceedings of the 2020 6th International Workshop on Container Technologies and Container Clouds*, 2020, pp. 37–42. https://doi.org/10.1145/3429885.3429967.

[24] Z. Kozhirbayev and R. O. Sinnott, "A performance comparison of container-based technologies for the cloud," *Future Generation Computer Systems*, vol. 68, pp. 175–182, 2017. https://doi.org/10.1016/j.future.2016.08.025.

[25] M. Moravcik, P. Segec, M. Kontsek, J. Uramova, and J. Papan, "Comparison of lxc and docker technologies," *Proceedings of the 2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 2020, pp. 481–486. https://doi.org/10.1109/ICETA51985.2020.9379212.

[26] R. Debab and others, "Containers runtimes war: a comparative study," *Proceedings of the Future Technologies Conference*, Springer, 2021, pp. 135–161. https://doi.org/10.1007/978-3-030-63089-8_9.

[27] W. Wang, L. Zhang, D. Guo, S. Wu, H. Cui, and F. Bi, "Reg: An ultra-lightweight container that maximizes memory sharing and minimizes the runtime environment," *Proceedings of the 2019 IEEE International Conference on Web Services (ICWS)*, 2019, pp. 76–82. https://doi.org/10.1109/ICWS.2019.00024.

***MOULINA HAZRA BHATTACHARYA** pursues her Master's degree in Computer Science at Liverpool John Moores University (LJMU). With a strong emphasis on research and practical application, her areas of scientific interest span across Cloud Computing, Full Stack Development, and Software Development. Bhattacharya's passion for these domains has positioned her to contribute meaningfully to the evolving landscape of computer science and technology.*

***DR. HARISH KUMAR MITTAL** is the Principal and Head of Computer Science at BM Institute of Engineering and Technology, Sonepat, with over 21 years of experience in teaching, entrepreneurship, and R&D. He holds a Ph.D. from Guru Jambheshwar University, has authored numerous research papers and engineering textbooks, and holds four patents. Dr. Mittal has served in various roles in academic conferences and journals, and his research interests span Software Engineering, Soft Computing, Cloud Computing, and Machine Learning.*