Assessing Kubernetes Distributions: A Comparative Study

Pedro Ascensão[1], Luís Filipe Neto[1], Karima Velasquez[1], and David Perez Abreu[2,1]

[1]Centre for Informatics and Systems of the University of Coimbra
[2]Instituto Pedro Nunes

***Abstract*—Kubernetes is now the most widely used container orchestration tool in the Cloud. However, when deploying Kubernetes clusters in resource-constrained environments such as the Cloud-to-Edge continuum, new challenges arise. To address this issue, lightweight distributions of Kubernetes have been developed. It is crucial to fully understand the performance and security levels of the Kubernetes solution to deploy, as these factors could impact the services and applications running in the cluster. This research compares the performance and security of three Kubernetes distributions: K8s, K3s and K0s. Results indicated K3s lacks in performance due to scalability issues compared to K0s (top performer) and K8s. Besides this, the latter two exhibit fewer security vulnerabilities.**

Keywords: **Kubernetes, Cloud, Edge, benchmark, performance, security.**

I. Introduction

In a constantly evolving technological landscape, efficient container management has emerged as a fundamental pillar for software development and operations in Cloud environments [1]. With the growing adoption of microservices and the need for more agile and scalable IT operations, container orchestration platforms such as Zero Friction Kubernetes (K0s), Lightweight Kubernetes (K3s), and Kubernetes (K8s) have gained prominence.

In the transition of organisations to Edge computing, seeking faster and more efficient data processing, lightweight platforms are emerging as key solutions as these are specially designed to operate on resource-constrained devices typical in Cloud-to-Edge continuum scenarios. Additionally, K0s and K3s offer scalability and flexibility, allowing companies to expand or reduce services as needed without the burden of additional physical resources. While focused on the Edge, these platforms maintain the ability to seamlessly integrate with Cloud-based systems, providing a hybrid approach that combines the fast local processing of the Edge with the extensive capabilities of the Cloud.

However, these technologies not only facilitate automation and efficiency in application deployment but also bring significant challenges, especially in terms of security and performance. As organisations seek to maximise service availability and minimise vulnerabilities, it is imperative to understand how these platforms compare and adapt to different operational contexts. In the Cloud-to-Edge environment, where choosing the right platform can have profound implications for the security, stability and efficiency of systems, our study aims to offer an in-depth comparative analysis of K0s, K3s and K8s.

Hence, the present study aims to investigate the nuances of security and performance in Kubernetes distributions, specifically K0s, K3s, and K8s, in Edge computing and Internet of Things (IoT) contexts. These platforms have gained relevance due to the need for efficient solutions in resource-limited environments, where security and operational efficiency are crucial. Kubernetes, as the dominant container orchestrator, has evolved to meet these needs, resulting in varied distributions, each with its own quirks and optimisations. This study analyses these differences, focusing on how each distribution balances security with resource efficiency and performance.

To provide an in-depth understanding of these distributions, we examine the architecture, implementation, and specific configurations of K0s, K3s, and K8s. The analysis includes an assessment of the suitability of these distributions in computing scenarios and their capabilities to deal with emerging security threats. To aid in this, various benchmarking tools have been developed, enabling developers to pinpoint areas for improvement and optimisation in their applications. Through a methodological approach, which includes benchmarking and comparative analysis, we seek to identify the most suitable Kubernetes distribution for different usage scenarios, considering aspects such as agility and security robustness. Furthermore, our findings aim to contribute to the Kubernetes community by providing detailed information that can guide future optimisations and developments not only in lightweight Kubernetes distributions but also in Kubernetes itself.

The rest of the paper is organised as follows. Section II presents a revision on related work. Section III introduces the three Kubernetes distributions considered in this study: K0s, K3s, and K8s. Section IV describes the benchmark tools used for the evaluation. Section V presents the evaluation methodology and Section VI shows the results obtained, which are analysed in Section VII.

Finally, Section VIII presents the final remarks and some research directions for future work.

II. Related Work

Existing literature on lightweight Kubernetes distributions such as MicroK8s, K0s, K3s, and MicroShift provides a foundation for understanding the evolution and performance of these technologies. Previous studies, mainly focused on MicroK8s and K3s, highlighted the efficiency of these solutions in resource-limited environments but left gaps regarding the analysis of more recent distributions such as K0s and MicroShift [2].

Fogli et al. [3] appraise the efficacy of diverse Kubernetes distributions within tactical networks, typically beset by bandwidth limitations and elevated latency. The study revealed Kubernetes can operate in Tactical Networks (TNs)s, with distributions such as KubeEdge being more apt for these settings, and favouring multiple small clusters over a single large one.

Pereira and Sinnott [4] study managed Kubernetes services as provisioned by prominent Cloud platforms including Amazon EKS, Azure AKS, and Google GKE. The study found no universal best option; AWS or NeCTAR excelled for CPU-intensive apps, while GKE outperformed in network-intensive apps, even beating manual deployments in its cloud.

Kjorveziroski and Filiposka [5] address the performance of K8s, K3s, and MicroK8s in Edge computing environments focusing on serverless computing. They evaluated these distributions under various benchmarks to understand their performance in Edge environments and concluded that K3s and MicroK8s offer better performance for the majority of serverless Edge workloads, at the cost of reduced integration capabilities. Böhm and Wirtz [6] study the same distributions, K8s, K3s and MicroK8s and established K3s shows a better performance except for applying deployments and draining workers in terms of needed time.

Unlike previous works, this study provides a detailed comparison between K0s and K3s (two of the most popular and recent lightweight Kubernetes distributions), and K8s (an older distribution known for being heavier but more complete and reliable). We evaluate responsiveness under load and the suitability of each distribution for different usage contexts. This research not only expands existing knowledge on lightweight Kubernetes distributions but also provides practical insights for practitioners and researchers in choosing the most suitable distribution for their specific needs.

III. Kubernetes distributions

This section sheds light on the Kubernetes distributions under analysis (K0s, K3s, and K8s), describing their features and capabilities.

K8s[1] is an open-source system that allows the automatic deployment and management of groups of containerised applications, called *pods* [7]. The architecture is organised in a cluster with a master node and one or more worker nodes. The master node deploys container workloads on worker nodes.

K8s offers a wide set of features, including automated load balancing, and robust self-healing capabilities that maintain the desired state of the applications improving reliability and scalability [8].

K3s[2] is a lightweight, open-source container orchestration platform developed by Rancher Labs. Its main purpose is to facilitate the deployment and management of Kubernetes clusters. These clusters have the same type of architecture as K8s.

This tool is especially useful for resource-constrained environments or for Edge computing due to its high portability. It stands out for having fewer dependencies, which makes installation and maintenance simpler; by integrating core Kubernetes components such as *containerd* for running containers, Flannel for networking, and Traefik or CoreDNS for service discovery and load balancing; and security, offering automatic generation of Transport Layer Security (TLS) certificates and the option to run Kubernetes on a single node with enhanced security settings [9].

K0s[3] is a distribution designed to simplify the deployment and management of Kubernetes clusters. Originating from the idea of reducing complexity without compromising functionality, K0s seeks to offer a simplified, resource-efficient alternative to traditional Kubernetes distributions. As with previous distributions, the cluster architecture is also composed of a master and one or more worker nodes.

K0s encompasses a wide variety of features in terms of security, as it includes features such as the existence of an Role-Based Access Control (RBAC), security policies at the level of pods and network, control plane isolation, and support for micro Virtual Machines (VMs) and OpenID providers; and as in terms of the cluster, as it features Domain Name System (DNS) (by CoreDNS), cluster metrics (by Metrics Server), GPU Support, cluster backup and restore, Zero-Downtime Cluster Upgrade (via K0sctl) and Horizontal Pod Autoscaling (HPA).

IV. Benchmark applications

This research incorporates two such tools for assessing Kubernetes clusters. The first, *K-bench*, is geared towards evaluating performance metrics, helping in understanding and enhancing the operational efficiency of the clusters. The second tool, *Kube-bench*, primarily concentrates on security aspects, providing insights into potential vulnerabilities and compliance issues. Both tools are described below.

[1] https://kubernetes.io [2] https://k3s.io [3] https://k0sproject.io

K-bench[4] is a performance benchmarking tool designed to assess the operational efficiency and resource utilisation of Kubernetes clusters. By simulating various workloads and testing scenarios, K-bench provides insights into the cluster's scalability, latency, and overall performance characteristics as it allows the configuration of different workloads simulating stress scenarios. K-bench has been previously used in Kubernetes performance analysis [2].

K-bench is engineered to be agnostic to Kubernetes distributions, making it suitable for benchmarking Kubernetes implementations such as K0s, K3s, and K8s. Whether deploying Kubernetes for mission-critical applications or lightweight IoT devices, K-bench aids administrators in optimising resource allocation, identifying bottlenecks, and fine-tuning cluster performance. This benchmark follows Koziolek and Eskandani's methodology to simulate harsher conditions than usual IoT scenarios, aiming for ease of replication and comparability in experiments [2].

Kube-bench[5] is an open-source benchmarking tool that evaluates the security configurations of Kubernetes clusters. The tool scans the cluster's configuration files, policies, and settings, comparing them against the predefined security benchmarks. Any deviations from the established benchmarks are highlighted, enabling administrators to pinpoint and remediate promptly security vulnerabilities [10]. This way, this application helps identify potential vulnerabilities and implement good security practices through thorough security analysis.

Kube-bench can be employed across various Kubernetes distributions, including the canonical K8s as well as lightweight K0s and K3s. Its versatility allows its usage by organisations of different sizes, ranging from enterprise-scale Kubernetes deployments to development and testing environments. The selection of Kube-bench was based on a survey from Red Hat, which revealed that 24% of the participants employed this particular benchmark. Its widespread adoption places it among the top three most utilised Kubernetes security benchmarks [11].

V. Evaluation Methodology

The experiments that were carried out aimed at evaluating the performance and efficiency of Kubernetes distributions, namely K0s, K3s, and K8s. In this scenario, 9 VMs were used, all with the same specifications so as not to distort the results. Each VM is equipped with 4 CPU cores, 4 GiB of RAM, and 50 GiB of storage, all running the Ubuntu 20.04.6 LTS operating system. The network topology was designed according to a standard clustered structure in order to reflect a realistic production environment for each Kubernetes distribution. Each cluster is composed of 3 nodes, distributed between one master and two workers, making a total of 9 VMs used, three VMs for the master node of each platform and other six VMs for the worker nodes.

The experimental procedures were divided into distinct phases. Initially, each cluster was installed and configured according to the recommended practices of each one of the distributions. Then, load tests using K-Bench were conducted to evaluate performance under varying load conditions, analysing responsiveness and scalability. The basic testing scripts included in the K-bench config folder were used, which differ in the number of deployments and pods created. In this paper, the testing for 1, 8, 16 and 24 pods/deployments (each deployment has 5 replicas) are presented. However, more results from other configurations are also available for consulting [6]. Each of these tests was run 30 times in order to mitigate the statistical error. Subsequently, Kube-Bench was applied using default configuration to verify the security posture of the clusters, identifying and correcting possible vulnerabilities.

VI. Results

This section presents the results obtained after executing the benchmarks presented in Section IV. The performance results obtained with K-bench are presented first (Subsection VI-A), then the security results obtained using Kube-bench are depicted in Subsection VI-B.

A. *Performance Results*

As mentioned above, the performance tests utilised K-Bench, a tool for generating metrics like pod creation time and server latency, including various latencies (initialisation, scheduling, starting, total startup) and Application Programming Interface (API) call latencies for Create-Read-Update-Delete (CRUD) operations across namespaces, services, deployments, and pods. This research specifically focused on CRUD operation latencies for services, along with creation throughput, scheduling latency, and starting latency, to better assess distribution performance.

We highlight the results obtained in terms of performance for each of the Kubernetes distributions, comparing the statistics for pod creation throughput and startup, scheduling, and pod initiation latency for each one of the tested workloads.

Starting with the creation throughput, according to Figure 1, K0s generally has the best throughput, indicating a superior ability to create pods quickly compared to K3s and K8s, especially noticeable under heavier loads. K8s, while maintaining competent performance, does not achieve the same level

[4] https://github.com/vmware-tanzu/k-bench/tree/master
[5] https://github.com/aquasecurity/kube-bench/tree/main
[6] https://github.com/ThunderStorm710/POWER

of throughput as K0s but still outperforms K3s, which shows the lowest throughput across all loads tested (except for the first one), which may indicate limitations regarding its scalability. Therefore, in terms of the ability to generate new pods, K0s leads, followed by K8s and K3s, which shows the most modest results.

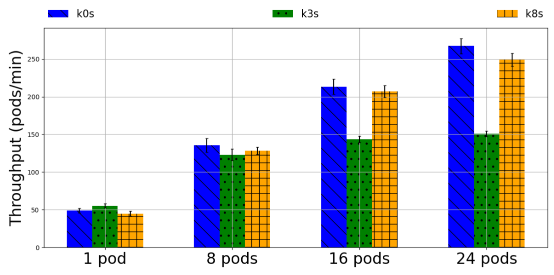


Figure 1: Pod Creation Throughput

For pod starting latency statistics, based on Figure 2, K0s and K8s show a more gradual increase in latency under heavier loads, suggesting better resource management that does not happen with K3s which shows a significant increase in maximum latency, indicating potential performance challenges under high load. K0s maintains relative consistency, highlighting its efficiency in launching pods even as the workload increases.

Lastly, for the pods scheduling latency, as shown in Figure 3, K8s tends to have lower latency compared to K0s, showing efficiency across all different scenarios. Besides this, K8s has the lowest scheduling latency in all scenarios, excelling in operational efficiency and agility. On the other hand, continuing the same trend, K3s records the highest latencies, which may be indicative of less efficient scheduling management under varying load conditions. In summary, K8s offers the best scheduling latency, closely followed by K0s which together show a considerable difference when compared to K3s.

The results of Service API calls for K0s, K3s, and K8s can be seen in the figure 4 in which the latency for the CRUD operations is displayed. In first analysis, for all cases, as expected the Get, List and Update operations require less time. On the other hand, the Delete operation, across all workloads depicts the highest latency for all distributions. K3s maintains generally higher latency, while K0s, although not the fastest, offers moderate latency and remains relatively stable even under heavier loads. These results emphasise the efficiency of K8s in managing services and reinforce the need for optimisation in K3s to improve performance under high demand.

B. *Security Results*

Having carried out and exposed all the performance tests, we now turn to another key point of our investigation, cluster security, taking especially into account the Control Plane, the Worker Node, and the Kubernetes policies in order to find out which distributions that passed/failed more Kube-bench security tests. Table I depicts the tests carried out for each of the areas under study and their results. In the following section, the content of this table will be discussed in more detail.

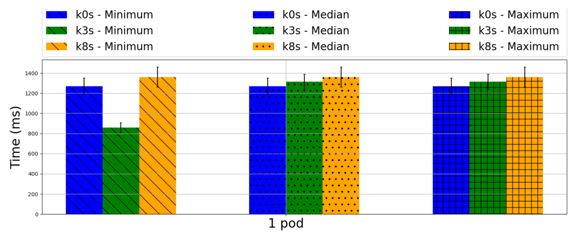
VII. Discussion

The performance results indicate that K3s is highly efficient, up to 16 pods per minute, beyond which point the throughput stabilises, indicating that the system has reached its maximum limit for pod creation per minute. This limitation may be due to factors such as available resources and the internal configurations of K3s. In contrast, both K8s and K0s maintain consistent performance even as the number of pods increases. K0s demonstrates superior performance in container orchestration, even under high load. On the other hand, K8s maintains stable performance but exhibits significant failures after 100 pods per minute, indicating a possible limitation in scalability.

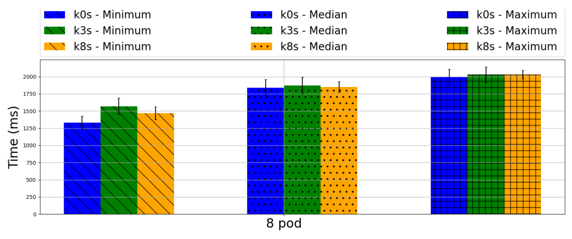
The latency analysis supports these findings, with graphs 2 and 3 showing considerably higher latency in K3s compared to the other distributions. This statement confirms that K3s has reached a high-stress point, which has directly affected the system's responsiveness. Taking into account the results shown in Figure 4 for 1, 8, 16 and 24 pods, as expected, latency increases with the increase in the number of pods, and this is higher in create and delete operations. Once again, except for the cases of 1 and 8 pods in which the distribution with the highest latency is K0s, in the remaining graphs, for the operations mentioned, K3s is the distribution with the highest latency. In the opposite direction comes K8s, which was consistently the distribution with the least time spent performing create and delete operations. K0s, despite not being the distribution with the best times, especially in the get, list and update operations, was always similar to K8s. Although this distribution took longer to process with smaller workloads, as can be seen in Figures 4a and 4b, with the increase in workload, the latency of K0s started following a similar trend to that of the K8s.

The security tests conducted using Kube-bench on the three distributions showed that K8s achieved the best overall results, with only 9 failed tests when using the default settings, without the need to modify any configuration documents. In contrast, both K0s and K3s performed poorly in terms of security. The only critical flaw identified in K8s is related to the Kubelet certification authority. This aspect involves generating or configuring a Certification Authority (CA). It is important to note that all other faults detected in K8s require adjustments to the configuration parameters, which is considered a relatively simple task.

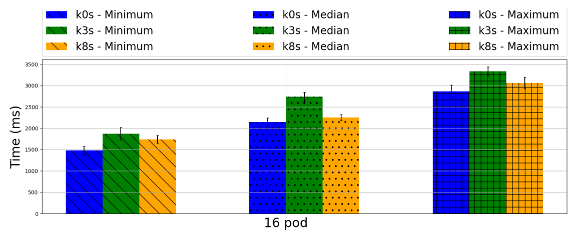
For K0s and K3s, in addition to the flaws already identified for K8s, significant problems are associ-



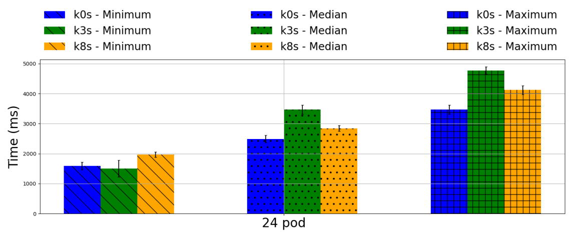
(a) 1 pod



(b) 8 pods

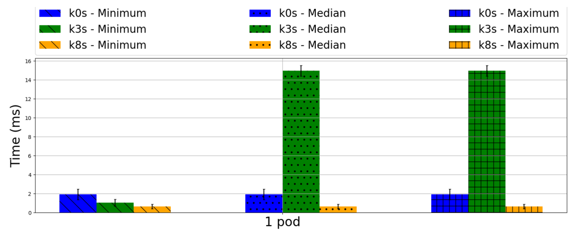


(c) 16 pods

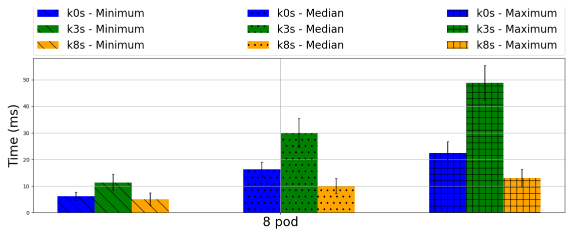


(d) 24 pods

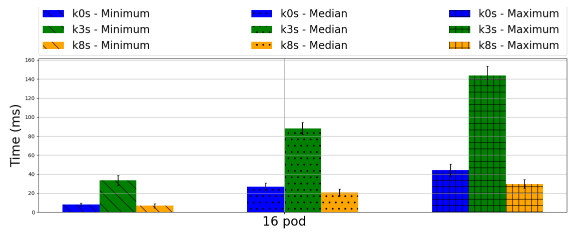
Figure 2: Pod Starting Latency



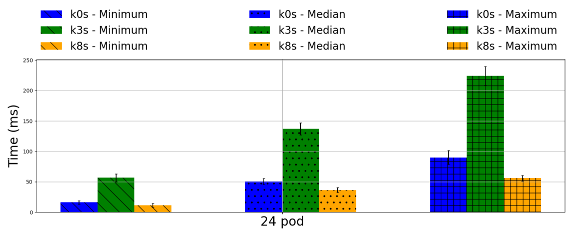
(a) 1 pod



(b) 8 pods



(c) 16 pods



(d) 24 pods

Figure 3: Pod Scheduling Latency

Table I: Kube-bench benchmarking results

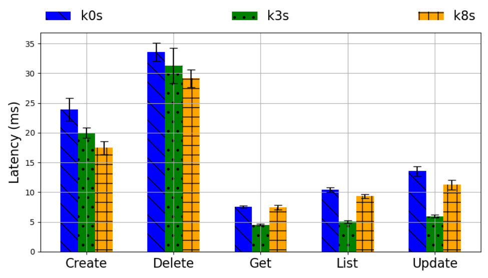
Security Tests	**Distributions**								
	K0s			**K3s**			**K8s**		
	PASS	**FAIL**	**WARN**	**PASS**	**FAIL**	**WARN**	**PASS**	**FAIL**	**WARN**
Control Plane	26	23	11	0	46	15	39	9	13
Worker Node	8	8	7	2	9	12	18	0	5
Kubernetes Policies	0	0	35	0	0	35	0	0	35

ated with authorisation and authentication. Due to the lightweight nature of these Kubernetes distributions, many of the identified test failures are related to the absence or deactivation of configurations. It is the user's responsibility to create or activate these configurations as needed.

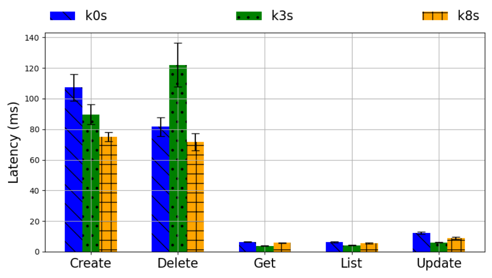
It is important to note that all distributions achieved the same result in the final test. None of them applied specific policies, leaving the creation and implementation of security policies up to the user. This analysis demonstrates that K8s is more efficient than other distributions in resolving the identified flaws. This advantage is due to the fact that K8s applies almost all security requirements by default, reducing the need for complex configuration modifications. Furthermore, directly comparing K0s with K3s, K0s, although much lower in terms of security compared to K8s, showed much better results than K3s especially in the Control Plane.

VIII. Conclusions and Future Work

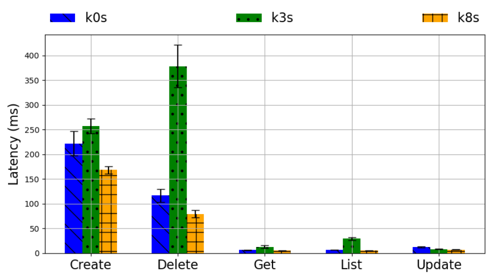
The results obtained in this research are in line with what was expected from each distribution, except for K3s, which demonstrated scalability difficulties, consistently obtaining values far above



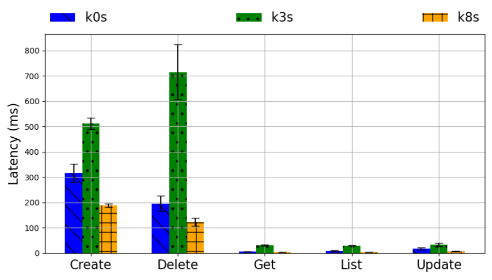
(a) 1 pod



(b) 8 pods



(c) 16 pods



(d) 24 pods

Figure 4: API Service Calls - CRUD Operations

the corresponding other distributions and simultaneously with the largest number of failed security tests. K8s once again proved to be a very powerful tool, with fewer security vulnerabilities and good scalability, as it demonstrated good results up to the point of creating 24 pods. Finally, the results obtained for K0s were the most balanced in terms of security and performance, as they showed even better scalability than K8s and a lower number of vulnerabilities than K3s. In scenarios with more resources and system security is of the utmost importance, K8s is the ideal choice due to its excellent results and superior functionalities. In situations where resources are limited, K0s is the best choice due to its high performance and balanced security. It is important to note that configure K0s is simpler than traditional Kubernetes, providing a more accessible experience. K3s only proved to be efficient for very low workload scenarios. In future work, more distributions will be tested, namely Microk8s and Microshift, in a more exhaustive way and with a greater workload.

Acknowledgement

This work is funded by the FCT - Foundation for Science and Technology, I.P./MCTES through national funds (PIDDAC), within the scope of CISUC R&D Unit - UIDB/00326/2020 or project code UIDP/00326/2020; and by the project POWER (grant number POCI-01-0247-FEDER-070365), co-financed by the European Regional Development Fund (FEDER), through Portugal 2020 (PT2020), and by the Competitiveness and Internationalisation Operational Programme (COMPETE 2020).

References

[1] I. Ponimansky, “The pillars of secure containers,” https://scalesec.com/blog/the-pillars-of-secure-containers/, 2023, accessed: 2023-12-31.

[2] H. Koziolek and N. Eskandani, “Lightweight kubernetes distributions: A performance comparison of microk8s, k3s, k0s, and microshift,” in *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*. New York, NY, USA: Association for Computing Machinery, April 2023, p. 17–29.

[3] M. Fogli, T. Kudla, B. Musters, G. Pingen, C. Van den Broek, H. Bastiaansen, N. Suri, and S. Webb, “Performance evaluation of kubernetes distributions (k8s, k3s, kubeedge) in an adaptive and federated cloud infrastructure for disadvantaged tactical networks,” in *2021 International Conference on Military Communication and Information Systems (ICMCIS)*. The Hague, Netherlands: IEEE, May 2021, pp. 1–7.

[4] A. Pereira Ferreira and R. Sinnott, “A performance evaluation of containers running on managed kubernetes services,” in *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. Sydney, NSW, Australia: IEEE, December 2019, pp. 199–208.

[5] V. Kjorveziroski and S. Filiposka, “Kubernetes distributions for the edge: serverless performance evaluation,” *The Journal of Supercomputing*, vol. 78, no. 11, pp. 13 728–13 755, 2022.

[6] S. Böhm and G. Wirtz, “Profiling lightweight container platforms: Microk8s and k3s in comparison to kubernetes,” in *ZEUS*. Bamberg, Germany: CEUR-WS, February 2021, pp. 65–73.

[7] M. Chima Ogbuachi, C. Gore, A. Reale, P. Suskovics, and B. Kovács, “Context-aware k8s scheduler for real time distributed 5g edge computing applications,” in *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. Split, Croatia: IEEE, September 2019, pp. 1–6.

[8] Kubernetes, “Installing kubeadm,” https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/, 2023, accessed: 2023-12-31.

[9] ——, “K3s,” https://docs.k3s.io/installation/requirements, 2023, accessed: 2023-12-31.

[10] B. Ünver and R. Britto, “Automatic detection of security deficiencies and refactoring advises for microservices,” in *2023 IEEE/ACM International Conference on Software and System Processes (ICSSP)*. Melbourne, Australia: IEEE, May 2023, pp. 25–34.

[11] B. Wilson, “Kube-bench: Kubernetes cis benchmarking tool,” https://devopscube.com/kube-bench-guide, 2023, accessed: 2023-12-31.