2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS) | 978-1-6654-2605-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/IDAACS53288.2021.9660392

The 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications
22-25 September, 2021, Cracow, Poland

A Comparison of Kubernetes and Kubernetes-Compatible Platforms

Sergii Telenyk [2,1], Oleksii Sopov [1], Eduard Zharikov [1], Grzegorz Nowakowski [2]
[1] National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute",
37, Prosp. Peremohy, Kyiv, Ukraine, sopov.alex.a@gmail.com
[2] Cracow University of Technology, ul. Warszawska 24, Krakow, Poland, stelenyk@pk.edu.pl,
gnowakowski@pk.edu.pl

***Abstract*—Nowadays, Kubernetes is an advanced container orchestration tool due to its high reliability, scalability and fault tolerance. However, Kubernetes requires a significant number of resources for its work. Therefore, to ensure the operation of Kubernetes in conditions of limited resources, lightweight analogues such as MicroKubernetes and K3S were created. These platforms provide easier deployment and support. In this paper, the authors analyze performance metrics for orchestration actions such as adding/removing nodes and starting/stopping deployments in terms of resource utilization, cluster startup speed, and consumed time for lightweight platforms and original Kubernetes. The results show that the original Kubernetes outperforms MicroKubernetes and K3S in many tests, but K3S demonstrates better disk utilization. On the other hand, MicroKubernetes demonstrates worst results in the performed tests.**

Keywords—container virtualization; performance; MicroKubernetes; K3S; deployment

I. INTRODUCTION

Kubernetes is an open-source container orchestration system for container virtualization and for management of containerized computer programs [1]. It was originally developed by Google and is now supported by the Cloud Native Computing Foundation. Kubernetes is designed as a platform to automate the deployment, scaling, and operation of application containers on clusters from hosts. It works with a variety of container tools and launches containers in a cluster, often with Docker-created images. In containerization, applications use shared OS kernel and contain only required binaries and libraries, enabling performance isolation and flexible deployment of complex microservice systems.

In [2], the authors present the design principles of the fog computing platform summarizing performance results on the integration of Docker Swarm Mode and Kubernetes. Results show poor resource utilization in Kubernetes in a normal mode and an idle mode, compared to competing solutions such as Docker Swarm. The use of Kubernetes can be significantly limited in IoT applications, and applications where the amount of computing resources is very limited, but there is a need for high reliability, performance, and resiliency.

Such tools as minikube, MicroKubernetes, K3S provide Kubernetes-compatible platforms by modifying and reorganizing the main components. They aim to simplify the configuration, startup, and maintenance of low-resource deployment clusters.

In [3], the authors analyze the performance of Kubernetes achieved through a Petri net-based performance model comparing Kuberntes with its lighter analogues under workload mode. In [4]-[5], the experiments on the different Kubernetes workload during the addition or removal of containers were performed.

However, there is no detailed analysis of how different platforms work during the full life cycle of a cluster. The above papers focused mainly on workload variations, as well as adding nodes to an existing cluster. A resource consumption during starting, stopping, adding, and removing nodes from the cluster was not considered.

Therefore, it is necessary to compare the performance of lighter analogues of Kubernetes with Kuberentes itself in all operations and stages of the cluster life cycle. Thus, the resource usage and runtime parameters of lightweight versions of Kubernetes and Kubernetes during regular operations need further research. In this paper the authors consider MicroKubernetes and K3S, but Minikube is not considered due to the inability to create a cluster.

II. RELATED WORKS

As containers emerged as a light virtualization alternative to virtual machines [6]–[8], several surveys have been published comparing container technology deployment, performance, orchestration systems, and monitoring tools. The first survey [6] gave a brief overview of virtualization and containerization technologies, discussed the taxonomies of containerization

technologies of the literature, described the performance metrics used in most containerization techniques, and identified the most important application domains of containerization and their technological progress. The review [7] highlights several challenges related to the implementation and deployment of microservices with containers, namely performance comparison, applications, run-time adaptation, and security. The systematic review of container deployment technologies [8] introduces the essential deployment metamodel by facilitating the comparison, selection, and migration of container technologies.

The performance analysis of Kubernetes horizontal pod auto-scaling algorithm based on absolute metrics has been presented in [9]. Using absolute metrics versus relative allows to properly control the application response time. The results [9] can be applied to any container's auto-scaling algorithm for managing compute-intensive workloads.

In [10], the authors compared the performance of Docker and Kernel-based virtual machine (KVM) in several ways, namely the comparison of CPU and memory usage of the host operating system, the measurement of CPU and memory usage in the idle state, the measurement of IO performance through large file copying, and the analysis of the Web server performance. Performance comparison shows that Docker uses hardware resources such as CPU, memory, and storage faster and more efficiently than KVM.

The authors of [11] provide results indicating that at the core of Kubernetes control plane performance problems as well as the problems of deployed applications is the etcd database and its performance. They conclude that using high-performance storage for hosting etcd can reduce the occurrence of poor application performance deployed on Kubernetes.

In [12], the authors investigate how Kubernetes resource metrics and Prometheus custom metrics affect Kubernetes horizontal pod autoscaler (KHPA) performance. Prometheus is customizable to monitor a wide collection of metrics, but it has additional performance overhead. On the other hand, Kubernetes ensure built-in resource metrics including CPU and memory usage of hosts and their pods. Obtained evaluation results allow to make several suggestions on how to optimize the performance of KHPA.

III. KUBERNETES PLATFORMS

Kubernetes is a portable, extensible open-source platform for managing workloads and container services, facilitating both declarative configuration and automation. It has a large, fast-growing ecosystem. Kubernetes services, support and tools are widely available. The main features of Kubernetes are service support and automatic load balancing, storage orchestration, automatic deployment and rollback of versions, auto-recovery, and reliability.

The Kubernetes cluster consists of a set of computing nodes that run container programs. Workstations place pods that are components of the program's workload. The Control Plane (main node, also known as master node) manages *N* working nodes in the cluster (Fig. 1). In large production environments, the Control Plane typically runs on multiple computers, and the cluster typically runs on multiple nodes, providing fault tolerance and high availability. For small workloads it is enough to run the Control Plane on a single compute instance.

The main processes of the Control Plane in Kubernetes are kube-apiserver, which provides the ability to communicate with the Kubernetes cluster, etcd - a database to save the state of the cluster, kube-scheduler, which distributes podes by nodes, kube-controller manager that manages the life cycle of nodes. Each node launches a kubelet, which executes containers in the pod and a kube-proxy to implement a network between nodes. Kubernetes requires at least 2 vCPU and 2 GB of memory to work in regular mode. The structure of Kubernetes is shown in Fig. 1.

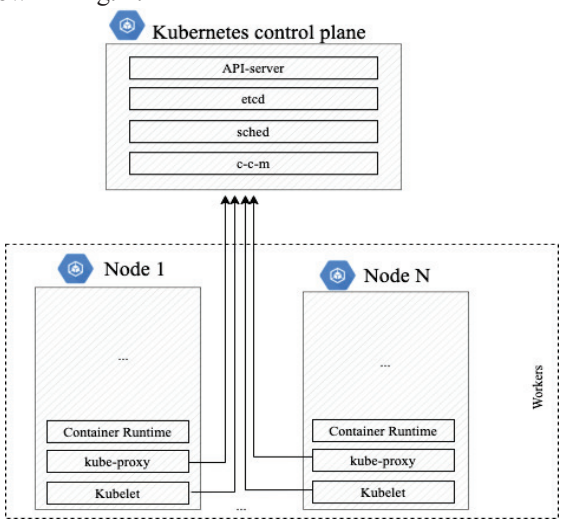

Figure 1. The structure of Kubernetes cluster

MicroKubernetes is a lightweight version of Kubernetes developed by Canonical [13]. It aims to simplify the use of Kubernetes by providing easy and fully compatible Kubernetes distributions, especially for low-end areas such as IoT. By default, MicroKubernetes uses all major Kubernetes components (e.g., API server, scheduler or controller manager) to make the cluster available.

While Kubernetes usually spins up a local virtual machine (VM) for the Kubernetes cluster, MicroKubernetes does not require a VM. It uses snap packages, an application packaging, and the isolation technology. MicroKubernetes uses Dqlite [14] instead of etcd [15] as highly-available storage. MicroKubernetes offers a quick and easy way to turn several computers or VMs into a multi-node Kubernetes cluster.

K3S is a lightweight version of Kubernetes developed by Rancher company [16]. K3S is designed to be a single binary of less than 40MB that completely implements the Kubernetes API. In order to achieve this, they removed a lot of extra drivers from the Kubernetes core and replaced drivers with add-ons. K3S is a fully Cloud Native Computing Foundation certified Kubernetes offering. This means that YAML files can be used to operate in the Kubernetes environment and YAML files can be also applied in the K3S cluster.

Due to its low resource requirements, it is possible to run a cluster on any server with RAM starting from 512MB. This means that pods are allowed to run on the master, as well as nodes.

K3S replaced etcd by sqlite3 [17] database. Also, in–tree storage drivers and cloud provider components are removed to keep the size small. K3S tries to lower the memory footprint reorganizing the Control Plane components in the cluster. Currently, there is only the option to run a single master. This means if the master node goes down there will not be further ability to manage the cluster.

IV. Evaluation and Quantitative Analysis

Firstly, the plan of experiments will be provided, including evaluation approaches and the experimental setup. Then, results of experiments will be compared and discussed.

The experiment was performed on four virtual machines of the e2-standard-2 type in Google Cloud with a resource capacity of 2 vCPU and 8 GB of RAM with Debian GNU/Linux 10 (buster) installed. Netdata is used as a monitoring tool on all machines to collect data about the system utilization at intervals of one second.

The collected data is stored in a database on the same virtual machine. Netdata is not resource intensive, so it will not affect experiment results in general. Entire lifecycle of Kubernetes consists of the following actions and states: start master, stop master, add worker, delete worker, apply deployment, remove deployment, idle cluster state. For each of these events, metrics for CPU, memory and disk utilization are collected. An average value for each metric is used for quantitative analysis. As a deployment, a public container image is used.

To conduct the experiments, the authors the authors performed the following: all virtual machines were prepared to run an experiment, then the cluster, consisting of one master and three worker nodes was set up, then different actions from the list above were performed and the defined metrics were collected. The experimental setup is shown in Fig 2.

There are four independent virtual machines connected by a common private network. Communication with virtual machines is carried out by means of ssh. VM2, VM3, and VM4 have the same basic setup as VM1, but components are omitted for readability.

A set of pre-known commands is sequentially sent to virtual machines. For each distribution appropriate commands are used.

Each experiment is repeated 10 times according to the same scenario to ensure its accuracy, and the average value was calculated for each action separately. All values are within the 95% confidence interval which indicates that experiment creates stable results in case of multiple runs.

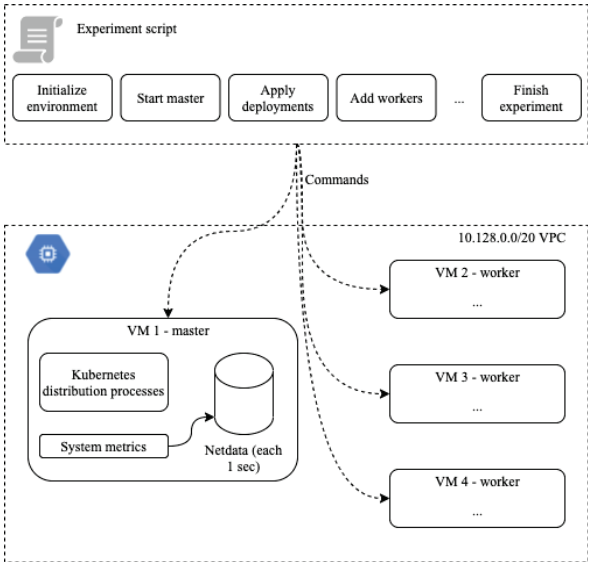

Figure 2. The experimental setup

Fig. 3, 4 and 5 show the resources utilization during experiments.

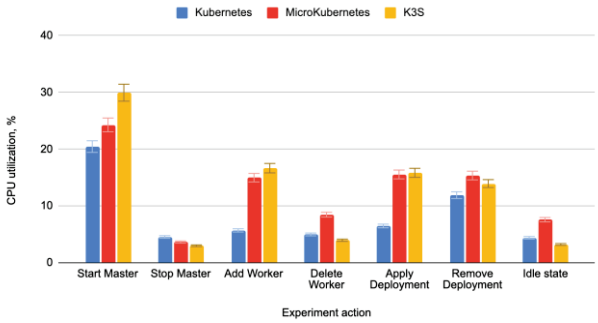

Figure 3. CPU utilization results

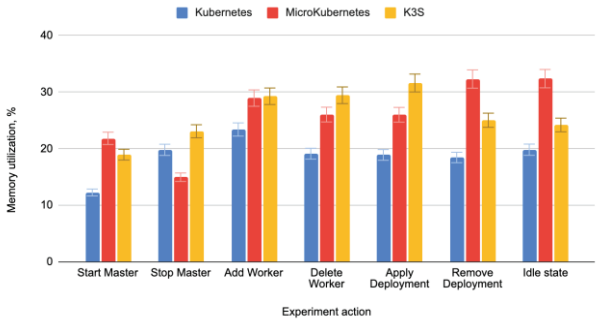

Figure 4. Memory utilization results

In general, the differences in resource utilization between different platforms are insignificant but taking into account that lightweight platforms developed to be run in resource limited environments, even little change can be meaningful.

K3S shows better disk utilization in all cases, it may be caused by replacing the original etcd database with a lightweight sqlite3 database. Kubernetes and MicroKubernetes show similar results in disk utilization (the difference is less than 1%).

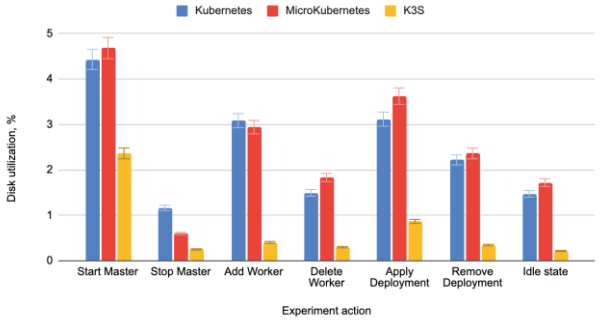

Figure 5. Disk utilization results

Original Kubernetes shows best memory utilization in most cases. MicroKubernetes shows better memory utilization in cases of master node stop, deleting workers and applying deployments compared to K3S. However, K3S is better in idle state, adding workers and removing deployments.

Starting a master node is the most CPU intensive operation over all platforms. K3S requires the most resources over MicroKubernetes and Kuberentes. Also, K3S requires more resources while adding workers and adding deployments. This behavior may be caused by packing everything into a single binary. However, K3S is the best in idle state. MicroKubernetes shows better performance while adding deployments and workers. In general, Kubernetes shows better results and MicroKubernetes along with K3S shows worse results that are almost the same. Fig. 6 show time consumption for the specified actions.

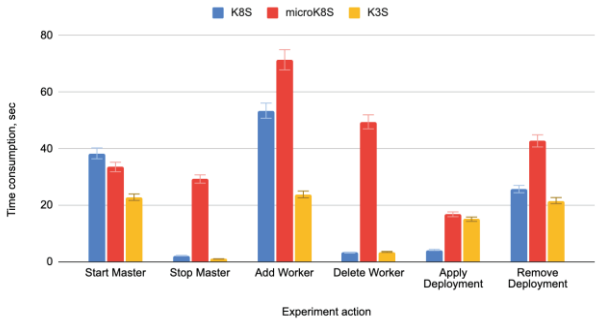

Figure 6. Time consumption for specified actions

Kubernetes shows the best time consumption for all actions, except starting master node. MicroKubernetes shows results similar to Kubernetes in starting master, adding worker and adding deployment. K3S shows better performance in starting master, adding worker, and removing deployment. MicroKubernetes shows bad results in adding and deleting workers and stopping master node.

Taking into account that cluster operations are performed infrequently, the authors make a conclusion that MicroKubernetes and K3S have similar metric values in operations concerning a workload management, namely applying deployments and removing deployments.

Despite the relatively high resource consumption, as it was described before, Kubernetes shows best results in all actions, so it can be used in applications that require fast deployment and fast cluster changes.

The difference of results is due to the difference of architectures of the presented lightweight platforms. Generally, platforms like MicroKuberentes and K3S are not very different in terms of resource and time consumption for different actions (operations). However, these platforms are slightly slower.

K3s shows the best disk utilization mostly due to use sqlite3 instead of etcd as a database. In terms of time consumption, K3S is comparable to original Kubernetes and shows better performance than MicroKubernetes. This may be caused by a single binary that contains a compiled MicroKubernetes distribution.

The overall resource utilization of MicroKubernetes is poor, especially in operations of cluster management, namely starting or stopping master, and starting or stopping workers. The reason for this is that MicroKubernetes is optimized for a single-node cluster.

V. Conclusion

The authors compared lightweight Kubernetes-compatible platforms MicroKubernetes and K3S with the original Kubernetes distribution in terms of resource utilization and time consumption during the whole cluster

lifecycle. Generally speaking, original Kubernetes shows better results comparing to its lightweight distributions and can be used as in computing nodes, as in Edge devices. K3S shows slightly worse results, however, it has better disk utilization. MicroKubernetes performs actions with poor resource utilization and time consumption compared to presented platforms. However, results are almost similar in terms of real-world usage examples and the revealed difference may be neglected. The main disadvantage of Kubernetes is that it requires a lot of resources by default, that may not be available in devices with limited resources. Therefore, using lightweight platforms may be useful in wide area of applications without any significant performance loss.

REFERENCES

[1] Kubernetes Components, [online] Available: https://kubernetes.io/docs/concepts/overview/components/

[2] A. Eiermann, M. Renner, M. Großmann, U. R. Krieger, "On a fog computing platform built on ARM architectures by docker container technology," *Innovations for Community Services*, Springer International Publishing, pp. 71–86. 2017.

[3] H. Fathoni, C. Yang, C. Chang, C. Huang, "Performance comparison of lightweight kubernetes in edge devices," *Algorithms and Networks in Pervasive Systems*, Springer International Publishing, pp. 304–309, 2019.

[4] V. Medel, R. Tolosana-Calasanz, J. A. Bañares, U. Arronategui, O.F. Rana, "Characterising resource management performance in kubernetes," vol. 68, pp. 286–297, 2018.

[5] V. Medel, O. Rana, J. A. Bañares, U. Arronategui, "Modelling performance and resource management in kubernetes," *Proceedings of the 9th International Conference on Utility and Cloud Computing*. pp. 257–262, 2016.

[6] O. Bentaleb, A. S. Belloum, A. Sebaa, & A. El-Maouhab, "Containerization technologies: taxonomies, applications and challenges," *The Journal of Supercomputing*, 1-38, 2021.

[7] E. Casalicchio, & S. Iannucci, "The state ‐ of ‐ the ‐ art in container technologies: Application, orchestration and security," *Concurrency and Computation: Practice and Experience*, vol. 32, issue 17, e5668, 2020.

[8] M. Wurster, U. Breitenbücher, M. Falkenthal, C. Krieger, F. Leymann, K. Saatkamp, & J. Soldani, "The essential deployment metamodel: a systematic review of deployment automation technologies," *SICS Software-Intensive Cyber-Physical Systems*, vol. 35, issue 1, pp. 63-75, 2020.

[9] E. Casalicchio, "A study on performance measures for auto-scaling CPU-intensive containerized applications," *Cluster Computing*, vol. 22, issue 3, pp. 995-1006, 2019.

[10] M. Chae, H. Lee, & K. Lee, "A performance comparison of Linux containers and virtual machines using Docker and KVM," *Cluster Computing*, vol. 22, issue 1, pp. 1765-1775, 2019.

[11] L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, & M. Kihl, "Impact of etcd deployment on Kubernetes, Istio, and application performance," *Software: Practice and Experience*, vol. 50, issue 10, pp. 1986-2007, 2020.

[12] T. T. Nguyen, Y. J. Yeom, T. Kim, D. H. Park, & S. Kim, "Horizontal pod autoscaling in Kubernetes for elastic container orchestration," *Sensors*, vol. 20, issue 16, 4621, 2020.

[13] Introduction to MicroK8s, [online] Available: https://microk8s.io/docs

[14] Dqlite. High-availability SQLite, [online] Available https://dqlite.io/

[15] Etcd. A distributed, reliable key-value store for the most critical data of a distributed system, [online] Available https://etcd.io/

[16] The certified Kubernetes distribution built for IoT & Edge computing, [online] Available: https://k3s.io/

[17] SQLite. Embedded SQL database engine, [online] Available https://www.sqlite.org/