CEUR-WS.org/Vol-2839/paper11.pdf

Profiling Lightweight Container Platforms: MicroK8s and K3s in Comparison to Kubernetes

Sebastian Böhm and Guido Wirtz

Distributed Systems Group, University of Bamberg, Bamberg, Germany
{sebastian.boehm,guido.wirtz}@uni-bamberg.de

Abstract. Kubernetes (K8s) is nowadays the first choice for managing containerized deployments that rely on high–availability, scalability, and fault tolerance. To enable the usage of container orchestration in resource–constrained environments, lightweight distributions emerged. The platforms MicroK8s (mK8s) and K3s, which are analyzed in this paper, claim to provide an easy deployment of K8s in a simplified form and way. In terms of resource utilization and deployment time of a K8s cluster, the lightweight platforms promise savings compared to K8s. We analyzed lightweight K8s distributions in a quantitative way by performing an experiment that monitors the utilization and time consumption compared to a native K8s cluster lifecycle. This involves starting, stopping, and adding nodes as well as creating, running, and deleting deployments. We show that not all platforms exhibit a quantitative advantage over K8s. K3s caused a similar resource consumption but had some performance advantages for starting new nodes and adding nodes to the cluster. The platform MicroK8s has shown a higher resource utilization and time consumption for all steps in our modeled lifecycle simulation.

Keywords: Lightweight Kubernetes · Container orchestration · Container lifecycle · Performance model.

1 Introduction

Kubernetes (K8s), nowadays the state–of–the–art container orchestrator, enables an efficient and comfortable way to run large and complex sets of interacting containerized applications. The container platform offers a comprehensive set of features to build highly available, scalable, and fault–tolerant clusters.[1] Driven by the emergence of containerization, a lightweight way of virtualization, K8s pervaded many different application areas like Fog, Edge, and IoT computing [5–7]. Over the years, K8s was already in focus of research regarding the resource utilization during running workloads [1, 2, 4, 8, 9]. Eiermann et al. [1] have shown that K8s causes a higher utilization in idle and load conditions compared to alternative platforms like Docker Swarm. This may limit the applicability in fields like Fog, Edge, and IoT computing that are characterized by resource–constrained devices but require features like high–availability, scalability, and fault–tolerance

[1] https://kubernetes.io/

J. Manner, S. Haarmann, S. Kolb, N. Herzberg, O. Kopp (Eds.): 13th ZEUS Workshop, ZEUS 2021, Bamberg, held virtually due to Covid-19 pandemic, Germany, 25-26 February 2021, published at http://ceur-ws.org
Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

to work in critical areas like surveillance and smart cities. Hence, K8s distributions claiming to be lightweight emerged to leverage the former mentioned application fields with K8s–managed deployments. The platforms MicroK8s (mK8s)[2], K3s[3], KubeEdge (KE)[4], and minikube (MK)[5] provide K8s–compatible distributions by modifying and reorganizing essential components. They aim to simplify configuring, running, and maintaining clusters to enable deployments with low–end devices. So far, quantitative performance benchmarks refer mainly to idle and load conditions [1, 2]. Other studies also consider the resource consumption when creating, starting, and stopping container instances [8, 9]. Nevertheless, there is no detailed evaluation regarding the resource and time consumption of steps like starting, adding, draining, or stopping nodes. Therefore, this paper aims to propose an experimental approach to evaluate the overall lifecycle of K8s. For this, we conducted an experiment with selected platforms to answer the following research question: **What is the resource and time consumption of the lightweight distributions mK8s and K3s in comparison to native K8s during typical events in a cluster lifecycle?**

We perform a simulation in a reproducible manner to derive detailed insights regarding the resource consumption and time consumption of all platforms. We decided to select mK8s and K3s as platforms because the controlling of cluster operations, like starting and stopping nodes, is working similar to K8s. KE is not considered in our research because it requires an additional K8s in the cloud which impedes a fair comparison to other platforms.

The rest of the paper is structured as follows: First, we discuss existing approaches to evaluate K8s distributions shortly (Section 2). Then, we cover the concepts of the considered platforms (Section 3), which help to comprehend the design and the results of the performance comparison in Section 4. Finally, we provide a critical review of the proposed experiment (Section 5) and outline plans for further experimental studies (Section 6).

2 Related Work

Benchmarking container platforms is not a novel part of research. Eiermann et al. [1] compared the CPU and memory utilization of a cluster consisting of five low–end devices running idle, running Docker Swarm, and finally K8s. Based on an HTTP load testing scenario, K8s has shown a 9–40 times higher utilization on average in comparison to Docker Swarm. Fathoni et al. [2] followed this approach and evaluated the lightweight platforms KE and K3s. They captured the CPU and memory utilization of a two–node cluster in idle and load conditions. They did not obtain a significant difference. K3s was compared to an additional K8s–compatible platform, called FLEDGE, by Goethals et al. [4]. They measured the needed amount of memory and the disk utilization of FLEDGE, K8s, and

[2] https://microk8s.io/
[3] https://k3s.io/
[4] https://kubeedge.io/en/
[5] https://minikube.sigs.k8s.io/docs/start/

K3s with different processor architectures and container runtimes. FLEDGE used around 50 % less resources than K8s and 10 % less than K3s on a x64 architecture. Medel et al. [8, 9] investigated different operational states of pods and containers. They measured the time to create, execute, restart, and stop a varying number of containers managed by a certain amount of pods. In addition, they obtained metrics for CPU, memory, disk, and network utilization.

However, there is no comprehensive analysis how the different platforms perform during a complete cluster lifecycle. Former work focuses mainly on idle and load conditions as well as applying deployments to an already existing cluster. The resource consumption for operations like starting, stopping, adding and removing nodes from the cluster is not considered. Hence, we provide a performance comparison model that covers all steps to track creating, starting, running, stopping, and deleting a K8s cluster.

3 Lightweight Kubernetes

Kubernetes. The container platform K8s represents a cluster based on a set of worker nodes. The worker nodes run so–called pods that contain a set of workloads (e.g., applications or batch jobs) to be executed. The set of worker nodes is managed by the *control plane*, which consists of several components that can be distributed over different nodes. The most important components are the *kube–apiserver* that exposes an API to interact with the cluster, *etcd* as distributed persistence layer to keep track of the cluster data, the *kube–scheduler* which is assigning pods to available nodes based on a set of policies, and the *kube–controller–manager* that is in charge of managing the lifecycle of a node and exposing service endpoints. Each node runs a *kubelet* that ensures the execution of containers in a pod and a *kube–proxy* to realize networking between nodes.[6]

K8s can be installed via provided package repositories. Using K8s involves the startup of the control plane, adding worker nodes to the cluster, and applying a deployment (a description of the workload to be executed). The same steps need to be done vice versa to tear down the cluster completely. At this, K8s requires at least 2 vCPUs with 2 GB memory.[7]

MicroK8s. Maintained by Canonical, mK8s aims to simplify the usage of K8s on public and private clouds by providing a lightweight and fully compliant K8s distribution, especially for low–end application areas like IoT.[8] By default, mK8s enables all basic components of K8s (like api–server, scheduler, or controller–manager) to make the cluster available. Further add–ons (e.g., DNS, ingress, or the metrics–server) can be enabled with one single command.[9] The realiza-

[6] https://kubernetes.io/docs/concepts/overview/components/
[7] https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/
[8] https://microk8s.io/docs
[9] https://microk8s.io/docs/addons

tion of high–availability, where multiple nodes carry the control plane and the datastore, can be achieved with a few commands.[10]

mK8s is provided by *snap*, Canonicals's package manager that runs applications in a sandbox.[11] Instead of etcd, which is used by K8s, mK8s uses *Dqlite* as high–availability datastore.[6,10] It is recommended to run mK8s with at least 4 GB of memory and 20 GB of storage (SSD recommended).[8]

K3s. Rancher offers K3s as lightweight K8s distribution, also with focus on low–end application areas. It is also fully compliant to K8s, contains all basic components by default, and targets a fast, simple, and efficient way to provide a highly available and fault–tolerant cluster to a set of nodes. The deployment takes place via one single and small binary including dependencies.[3]

Similar to mK8s, Rancher replaced etcd by another datastore, here *sqlite3*. Also, in–tree storage drivers and cloud provider components are removed to keep the size small. K3s tries to lower the memory footprint by a reorganization of the control plane components in the cluster. The K3s master and workers, also called server and agents, encapsulate all the components in one single process.[12]

K3s is installed via a shell script that allows it to be run as a server or agent node. To achieve high–availability, new worker nodes can be easily added to the cluster by running a few commands.[13] The minimum hardware requirements are at least 1 vCPU and 512 MB of memory.[14]

4 Performance Comparison

This chapter covers the performance comparison of the three considered platforms. In the first place, we describe our experimental approach and environment shortly. Afterward, we analyze our obtained results from the experiment.

4.1 Experimental Setup and Design

To evaluate the resource consumption, we set up a controlled environment to achieve reproducible, comprehensible, and consistent results. According to the recommended system requirements (Section 3), we used four Ubuntu 20.04 Virtual Machines (VMs) with 2 vCPUs, 4 GB memory and a fast SSD with a capacity of 50 GB each. All VMs run on-premises on one single physical host machine with Kernel–based Virtual Machine (KVM) as hypervisor and *containerd* as container runtime. The host machine is equipped with a AMD Ryzen 7 3700X CPU (8 cores), 64 GB memory and a fast SSD. We deployed *netdata* as monitoring tool on all machines to collect data about the system utilization with a sample rate of 5 s. The collected data is stored in a document–oriented database

[10] https://microk8s.io/docs/high-availability
[11] https://snapcraft.io/docs/getting-started
[12] https://rancher.com/docs/k3s/latest/en/architecture/
[13] https://rancher.com/docs/k3s/latest/en/
[14] https://rancher.com/docs/k3s/latest/en/installation/installation-requirements/

on another machine. Netdata's monitoring agent creates a small CPU utilization of around 1 %, a negligible memory usage and disk utilization.[15]

In order to evaluate the entire lifecycle, we extended the approach of Fathoni et al. [2] as follows: We redefined the set of events to be evaluated. That means, we collect the CPU, memory, and disk utilization during starting, adding, running, draining, and stopping of nodes, as well as applying, running, and deleting a small web server deployment[16] with three replicas of *nginx*[17].

The experiment is structured as follows: First, we installed all platforms with one master and three workers on the respective machines to create a high–availability cluster. Secondly, we stopped all platforms and platform–related services to put the system into idle condition. Finally, predefined *ansible playbooks* instruct the machines to perform the different steps of the lifecycle model. All playbooks and further details of the implementation are available on GitHub[18]. In total, we performed 25 runs per platform and averaged the results. The raw data and detailed metrics for all runs are available on GitHub[19] as well.

4.2 Experimental Results

Figure 1 shows the average resource utilization by master and worker nodes for all platforms over time. The small numbers at the top of each diagram imply the middle of the different steps in the lifecycle simulation.

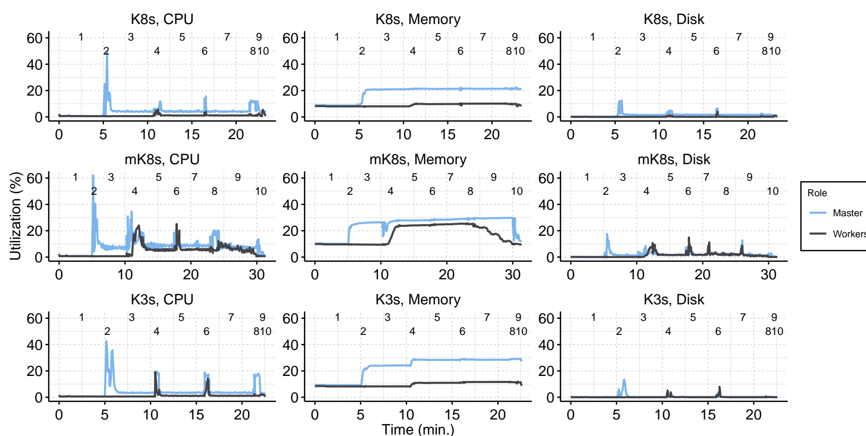

Fig. 1: Lifecycle analysis for K8s, MicroK8s, and K3s.

[15] https://github.com/netdata/netdata#features
[16] https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/controllers/nginx-deployment.yaml
[17] https://www.nginx.com/
[18] https://github.com/spboehm/kns-profiling
[19] https://spboehm.github.io/kns-profiling/

Table 1: Average resource consumption (μ/σ) for all platforms and events.

No.	Event	Role	K8s			mK8s			K3s		
			CPU	Memory	Disk	CPU	Memory	Disk	CPU	Memory	Disk
1	System idle	Master	0.64/0.3	8.66/0.81	0.05/0.09	0.76/0.35	10.16/1.12	0.08/0.11	0.6/0.27	9.15/0.8	0.04/0.09
		Workers	0.62/0.29	8.04/0.59	0.04/0.09	0.74/0.3	9.69/0.81	0.07/0.12	0.62/0.25	8.28/0.51	0.04/0.08
2	Start master	Master	19.15/17.87	13.32/3.98	4.15/5.55	24.78/20.43	20.6/4.64	4.88/7.61	28.83/12.65	16.46/4.71	2.68/2.84
		Workers	0.5/0.15	7.99/0.58	0.01/0.03	0.61/0.17	9.6/0.79	0.04/0.07	0.52/0.14	8.22/0.5	0.01/0.05
3	Master idle	Master	4.67/2.82	20.83/0.77	1.87/1.92	8.61/6.08	25.88/1.54	2.19/3.15	5.73/8.28	23.82/1.11	1.12/3.44
		Workers	0.57/0.16	8/0.56	0.02/0.06	0.66/0.18	9.49/0.76	0.03/0.07	0.57/0.16	8.24/0.48	0.02/0.09
4	Add workers	Master	6.33/2.59	21.27/0.65	3.18/1.77	15.95/10.62	25.91/3.13	3.23/2.82	14.99/6.88	27.07/2.08	0.39/0.37
		Workers	2.77/3.51	9.06/0.96	0.57/1.51	12.67/19.76	16.02/6.36	3.63/6.07	8.37/10.84	9.66/1.03	1.93/3.66
5	Cluster idle	Master	4.27/1.12	21.3/0.64	1.71/0.46	8.83/2.58	27.85/1.1	1.87/0.83	3.77/2.62	28.38/0.99	0.23/0.22
		Workers	1.1/0.4	9.77/0.48	0.05/0.18	5.78/3.49	23.99/1.14	1.92/2.13	1.26/0.86	10.95/0.43	0.16/0.67
6	Apply deployment	Master	7.19/4.53	21.3/0.61	3.12/1.92	17.72/5.06	28.29/1.09	3.74/3.39	15.37/5.85	28.74/0.96	0.77/0.97
		Workers	1.67/1.02	9.73/0.51	0.49/1.02	11.03/7.72	24.42/1.04	4.41/3.78	5.63/6.38	11.18/0.44	1.59/2.79
7	Deployment idle	Master	4.23/1.33	21.4/0.62	1.67/0.63	9.01/2.5	28.6/1.07	2.46/2.82	3.69/2.51	28.48/0.99	0.23/0.2
		Workers	1.16/0.38	10.06/0.43	0.07/0.28	5.92/2.77	25.03/0.99	2.49/3.26	1.38/2.2	11.61/0.4	0.18/1.4
8	Delete deployment	Master	10.71/2.96	21.5/0.6	2/0.57	17.63/3.63	29.18/0.98	2.34/1.32	14.67/6.03	28.82/1.07	0.31/0.15
		Workers	1.45/0.81	9.97/0.42	0.33/0.39	5.49/1.79	25.16/0.92	2.15/1.41	1.54/0.8	11.58/0.36	0.29/0.37
9	Drain workers	Master	4.58/1.69	21.71/0.5	1.36/0.16	7.77/2.52	29.55/1.03	1.89/2.78	3.56/2.26	28.89/0.73	0.28/0.2
		Workers	2/1.97	9.94/0.42	0.22/0.51	5.95/10.27	17.3/7.1	1.52/3.24	2.08/1.5	11.37/0.47	0.08/0.1
10	Stop master	Master	3.91/0.65	21.53/0.7	1.24/0.17	3.38/3.91	15.61/6.12	0.66/1.48	3.15/0.87	27.16/1.46	0.24/0.11
		Workers	0.64/0.14	8.31/0.48	0.1/0.08	0.71/0.21	9.89/0.66	0.07/0.12	0.65/0.18	8.71/0.29	0.18/0.29

For instance, event no. 2 involves the event *Start master*. This explains the very short peek in the CPU and I/O utilization as well as the increasing amount of memory. To measure the utilization of all idle events (i.e., event no. 1, 3, 5, and 7 - see Table 1 for details), we defined a time interval of five minutes to obtain the average utilization. Aside from mK8s, K8s and K3s are showing similar results, especially in terms of the time needed to pass through all steps in the experiment. Only mK8s needed more time (μ = 1860 s / σ = 17.1 s) in comparison to K8s (1379 s / 12.9 s) and K3s (1361 s / 9.47 s). The platforms K8s and K3s cause similar load profiles for all metrics as displayed in Table 1. However, K3s burdens CPU and memory slightly more than K8s but shows a smaller average and volatility in disk utilization. K3s has shown slightly better results for CPU and disk utilization in comparison to K8s only for a few events (e.g., event no. 5, 7, 9, and 10). The memory consumption of mK8s and K3s is quite similar. However, mK8s shows the highest utilization for CPU and disk. On average, the master nodes mostly need more resources compared to the worker nodes because the cluster–managing services are located there.

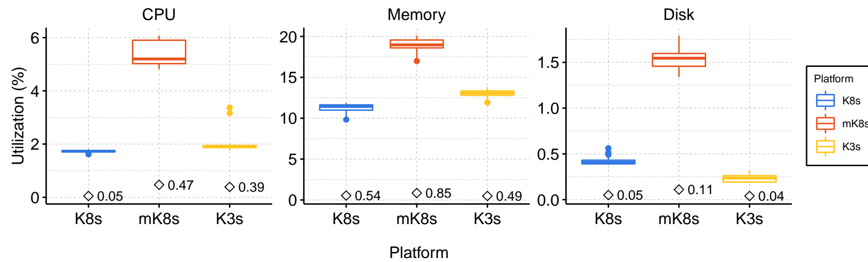

Fig. 2: Resource utilization for K8s, MicroK8s, and K3s ($\diamond = \sigma$).

Figure 2 shows the average utilization of all events for all platforms as box plots. Furthermore, the standard deviation is displayed ($\diamond = \sigma$). The previously described results are also reflected in this overall and averaged comparison. The platform mK8s shows the worst results for all metrics. Regarding the CPU and memory utilization, the differences between K8s and K3s are very small. K3s shows a slightly smaller disk utilization. Consequently, the claim of lightweight K8s holds only partially. All platforms exhibit a small σ which indicates that the experiment created stable results under repeated simulations. Also, the box plots do neither show an extensive amount of outliers nor a high dispersion.

The taken amount of time for selected steps in the cluster lifecycle is displayed in Figure 3. The values refer to all four nodes. K3s shows better results for nearly all events compared to K8s, except the needed time to apply deployments and drain workers. In this comparison, mK8s is quite close to K8s but needs a very long time for adding and draining workers. mK8s starts a master node a bit faster compared to K8s but slower than K3s. The creation of new deployments happens nearly in the same time for mK8s and K3s. K8s applies the deployment around four times faster than the other platforms. The deletion of a deployment by mK8s roughly takes twice the time K8s or K3s needs. Tearing down the cluster happens rapidly as well, mK8s needs around 32 seconds.

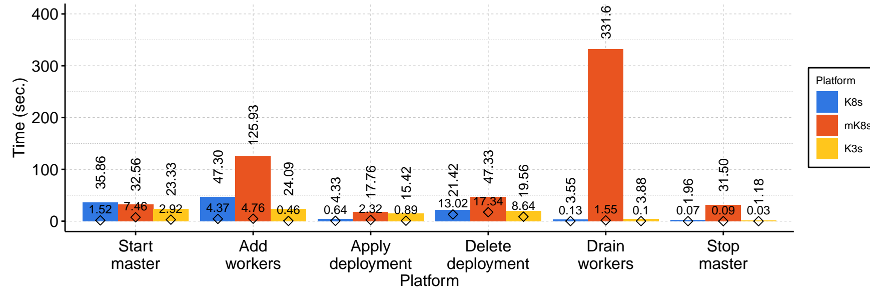

Fig. 3: Average time consumption for K8s, MicroK8s, and K3s ($\diamond = \sigma$).

5 Discussion

The obtained results from the previous chapter can be explained with the different characteristics of the lightweight platforms (Section 3). K3s has shown a very small disk utilization potentially due to the usage of sqlite3 instead of etcd as database. In terms of time, K3s shows merits for all events except applying deployments or draining workers compared to K8s. Bundling all components of K8s into one single process may lead to this performance enhancement. mK8s had overall a higher resource utilization. Especially adding and draining workers needed a long time. The reason for this is that mK8s is optimized for a

single-node cluster. When a particular node leaves the cluster, it restarts again as single-node K8s automatically.[20] There may be an option to avoid this restart and reduce the time for draining nodes by stopping mK8s forcefully. However, we followed the official documentation, which also states that adding and graceful draining of nodes may require minutes.[10] It is worth noting that mK8s has the highest system requirements, followed by K8s and finally K3s (Section 3).

The proposed experiment and the obtained results underlie a few limitations. Firstly, we tested all platforms running on a virtualized setup with KVM and containerd. Other alternatives may have an impact on the experimental results. However, containerd is the recommended runtime for mK8s and K3s.[21,22] We measured only the utilization on the overall system–level without network utilization and did not consider the utilization of K8s–related processes in detail. Furthermore, we modeled the entire lifecycle with a limited set of machines to obtain results on how long each platform needs to perform a set of actions. We did not burden the web server deployment and kept everything in idle condition. However, a performance comparison based on applications was not in focus of our research. There may be a need for synthetic benchmarks for lightweight on–premises container platforms, such as performed by Ferreira and Sinnott [3].

6 Conclusion and Future Work

This paper showed an approach to compare different K8s distributions in a quantitative way. To answer our research question, we conclude that replacing K8s with a lightweight distribution can be beneficial in particular circumstances. For the most part, there are only small differences regarding the resource utilization between K8s and K3s. mK8s showed a higher resource and time consumption for nearly all events. K3s has shown a better performance except applying deployments and draining workers in terms of needed time. Although our experiment shows only preliminary results and the findings are limited to some extent, we argue that not all platforms fulfill the claim being more lightweight compared to K8s. In particular, areas like Fog, Edge, and IoT computing with a highly varying number of nodes over time can benefit from lightweight K8s platforms.

We plan to enrich our proposed simulation model with detailed analysis at the process–level for future work. To increase the overall validity, we want to run the experiment at a larger scale to get better insights how the different distributions manage a larger number of nodes and workloads. Furthermore, we want to deepen the statistical analysis to obtain significant differences between the platforms regarding resource utilization and time consumption of our lifecycle model. Finally, it is worth considering the qualitative dimension. As pointed out in Section 3, each platform provides various ways to deploy and interact with the cluster. We want to consistently evaluate these differences in a qualitative survey by taking other lightweight platforms like KE and MK into consideration.

[20] https://microk8s.io/docs/clustering
[21] https://microk8s.io/docs/configuring-services
[22] https://rancher.com/docs/k3s/latest/en/advanced/

References

1. Eiermann, A., Renner, M., Großmann, M., Krieger, U.R.: On a fog computing platform built on ARM architectures by docker container technology. In: Innovations for Community Services, pp. 71–86. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-60447-3_6
2. Fathoni, H., Yang, C.T., Chang, C.H., Huang, C.Y.: Performance comparison of lightweight kubernetes in edge devices. In: Pervasive Systems, Algorithms and Networks, pp. 304–309. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-30143-9_25
3. Ferreira, A.P., Sinnott, R.: A performance evaluation of containers running on managed kubernetes services. In: 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 199–208. IEEE (2019). https://doi.org/10.1109/cloudcom.2019.00038
4. Goethals, T., Turck, F.D., Volckaert, B.: FLEDGE: Kubernetes compatible container orchestration on low-resource edge devices. In: Internet of Vehicles. Technologies and Services Toward Smart Cities, pp. 174–189. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-38651-1_16
5. Javed, A., Heljanko, K., Buda, A., Framling, K.: CEFIoT: A fault-tolerant IoT architecture for edge and cloud. In: 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), pp. 813–818. IEEE (2018). https://doi.org/10.1109/wf-iot.2018.8355149
6. Kayal, P.: Kubernetes in fog computing: Feasibility demonstration, limitations and improvement scope : Invited paper. In: 2020 IEEE 6th World Forum on Internet of Things, pp. 1–6. IEEE (2020). https://doi.org/10.1109/wf-iot48130.2020.9221340
7. Kristiani, E., Yang, C.T., Huang, C.Y., Wang, Y.T., Ko, P.C.: The implementation of a cloud-edge computing architecture using OpenStack and kubernetes for air quality monitoring application pp. 1–23 (2020). https://doi.org/10.1007/s11036-020-01620-5
8. Medel, V., Rana, O., Bañares, J.Á., Arronategui, U.: Modelling performance & resource management in kubernetes. In: Proceedings of the 9th International Conference on Utility and Cloud Computing. p. 257–262. UCC '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2996890.3007869
9. Medel, V., Tolosana-Calasanz, R., Bañares, J.Á., Arronategui, U., Rana, O.F.: Characterising resource management performance in kubernetes. vol. 68, pp. 286–297. Elsevier BV (2018). https://doi.org/10.1016/j.compeleceng.2018.03.041

All links were last followed on February 20, 2021.