Comparing Cloud and On-Premises Kubernetes: Insights into Networking and Storage Tooling

Jakob Koller[1,*], Sebastian Böhm[1]

[1]University of Bamberg, An der Weberei 5, Bamberg, 96047, Germany

Abstract

Kubernetes (K8s) is nowadays a well-recognized platform for automating deployment, scaling, and management of containerized workloads. However, running K8s in an on-premises environment comes with unique challenges, because several crucial components are typically managed by the cloud providers. These challenges are especially pronounced in network-specific load balancing services and storage management. For running K8s in an on-premises environment, these functionalities must be provided additionally. To see how open-source tools compare to their cloud counterparts, we conducted experiments evaluating MetalLB and Cilium for networking, as well as Ceph & Rook and Longhorn for storage. The results showed that MetalLB and Cilium, for networking, could achieve similar results to cloud-based K8s. For storage, Ceph & Rook outperformed their cloud counterparts, whereas Longhorn delivered inferior results.

Keywords

Kubernetes, On-premises, Container orchestration, Networking, Storage

1. Introduction

Nowadays, Kubernetes (K8s) is one of the leading platforms for container orchestration. It is widely recognized as the state-of-the-art platform for automating deployment, scaling, and management of containerized workloads [1]. K8s was originally designed with cloud environments in mind, leveraging the scalability and flexibility that cloud services provide [2]. However, there are scenarios where cloud environments may not be ideal. For instance, workloads involving sensitive data that cannot be stored externally often necessitate alternative deployment strategies. In such cases, an on-premises environment becomes viable, enabling organizations to retain full control over their data and systems. Despite its advantages, running K8s on-premises introduces significant challenges. Many cloud providers offering K8s manage several critical components, including storage and networking, as part of their service. In an on-premises setup, these components must be replaced with something equivalent, making the process more complex. Providing equivalents for network and storage on-premises is considered the most significant challenge. Networking includes the load balancer component, which is essential for exposing the deployment to the public. K8s itself does not provide a load balancer; in an on-premises environment, the load balancer component must be replaced with an alternative solution [3]. The cloud environment provides scalable and distributed storage solutions as a managed service for storage. This concept must be adapted for the on-premises environment.

This paper aims to evaluate how tools for networking and storage in an on-premises K8s environment functionally compare to their cloud-based counterparts. To achieve this, we conducted an experiment to address the following research question: **How do networking and storage tools for running K8s in an on-premises environment functionally compare to their cloud-based equivalents?**

To answer our research question, we conduct a reproducible experiment to evaluate whether an on-premises K8s setup, enhanced with additional tooling, can provide a functional experience comparable to a cloud-based environment. By executing the experiment in both environments, our approach ensures

Vienna'25: 17th Central European Workshop on Services and their Composition (ZEUS), February 20–21, 2025, Vienna, Austria
*Corresponding author.

jakob.koller@stud.uni-bamberg.de (J. Koller); sebastian.boehm@uni-bamberg.de (S. Böhm)
https://www.uni-bamberg.de/en/pi/team/boehm-sebastian/ (S. Böhm)
0000-0003-3719-1923 (S. Böhm)

insights into the feasibility, performance, and usability of on-premises solutions compared to their cloud counterparts.

The remainder of the paper is structured as follows: Section 2 discusses the current research on on-premises K8s. Section 3 provides an overview of existing tools that enable K8s functionality in on-premises environments. In Section 4, we leverage these tools to conduct our experiment. Finally, we critically review the experiment in Section 5 and conclude our work in Section 6.

2. Related Work

There are already related approaches to this work that discuss and evaluate solutions for cloud-equivalent on-premises K8s. Packard et al. [4] discussed running and building a K8s cluster in an on-premises environment. They partly discussed the networking components of their cluster, focusing on the external connectivity to the Internet. In addition, they highlighted their solution for the storage aspect. However, they only described a use case-based proof of concept with InfluxDB[1] and K8s. Also, they considered only a small subset of network and storage tools and did not provide a comparison or reasoning for their tool selection. Ruiz et al. [5] used an on-premises K8s cluster to test their custom and QoS-aware autoscaling of deployments in different cluster setups. They provided a custom load balancer to address network challenges and did not use publicly available open-source solutions. Tackling storage-related aspects for on-premises K8s was not in scope. Manaouil and Lebre [6] discussed K8s in the domain of edge computing, especially the applicability of geographically distributed K8s clusters. They used a basic cluster setup not designed for production usage for testing. Mondal et al. [7] set up a K8s cluster from scratch without any auxiliary tooling. The deployments were only internally exposed. Hence, a solution for load balancing was not discussed. Böhm and Wirtz [8] compared different K8s distributions based on their performance characteristics. However, only the baseline functionality is included and needs to be extended for a production-like environment.

None of the related works discussed a cloud-equivalent on-premises setup, considering both network and storage aspects. Specifically, no empirical and quantitative evaluation yet shows the outcome when K8s nodes fail. Consequently, this work wants to address these gaps by providing an overview of tooling with their functional comparability to cloud-based K8s.

3. Tooling

This section discusses the available tooling for networking and storage. To identify an eligible set of tools for evaluation, we first followed the findings mentioned in the previously discussed related work (Section 2). Additionally, we enriched the set of tools by researching the internet for further solutions. We eliminated solutions that do not contribute to our goal of cloud equivalence from a feature perspective. Furthermore, we do not consider tools with minor reputations, incomplete documentation, or inactive development.

3.1. Networking

Our research revealed two classes of solutions exist for providing load balancing for services, particularly load balancers and via the Container Network Interface (CNI). We selected two representative tools for the identified categories: MetalLB as the load balancer and Cilium as the CNI.

MetalLB. One of the most mature and widely used load balancers for on-premises K8s environments is MetalLB [9]. It operates in Layer 2 mode or Border Gateway Protocol (BGP)[2] mode. In Layer 2 mode, MetalLB uses the Adress Resolution Protocol (ARP) and Neighbor Discovery Protocol (NDP) to assign multiple IP addresses to a single machine. However, this mode has one significant limitation: the

[1]https://www.influxdata.com/

[2]BGP is a routing protocol used to exchange network reachability information between systems, enabling efficient traffic distribution and scalability in complex network environments [10].

incoming traffic from outside the cluster is limited by the bandwidth of a single node. All incoming traffic must pass through this node before being distributed across the cluster, creating a potential bottleneck. The second mode uses BGP to establish a direct BGP peering session between the router and the different nodes. This allows a BGP-compliant router to forward traffic directly to the appropriate node, bypassing the single node bottleneck[3].

Cilium. Unlike MetalLB, which is solely a load balancer, Cilium is primarily a CNI with additional features. One of the features is the capability to provide load balancing for services. Similar to MetalLB, it supports both Layer 2 and BGP mode. Since K8s requires a CNI by design, using the integrated load balancer from Cilium eliminates the need to install additional tools to provide load balancing[4].

3.2. Storage

Longhorn. Developed by Rancher, Longhorn is an open-source block storage system. It claims to be lightweight and reliable, which could be important in resource-constrained environments. It provides incremental snapshots of the block storage, automated backups to secondary storage, replication of block storage across multiple nodes or even data centers, non-disruptive upgrades, and an intuitive dashboard[5].

Ceph & Rook. Rook allows Ceph[6] storage to be used natively in K8s. Ceph is a highly scalable distributed storage solution for block storage, object storage, and shared file systems. Rook is a framework that automates the deployment and management of Ceph to provide self-managing, self-scaling, and self-healing storage. Rook achieves this by using K8s resources to deploy, configure, provision, upgrade, and monitor Ceph. Like Longhorn, Ceph can automatically replicate data to other available nodes to protect the cluster from data loss[7].

4. Functional Comparison

This chapter presents the functional comparison between the cloud-based and on-premises K8s setups. We describe the experimental environment and procedure and present, analyze, and discuss the results.

4.1. Experimental Environment

The test environment for the on-premises setup consists of five locally hosted Virtual Machines (VMs) distributed across multiple machines. Each VM is equipped with 2 vCPUs, 4 GB of RAM, a 40 GB SSD boot disk, and an additional 12 GB disk for the storage experiment. These five VMs are configured as a highly available K8s cluster using K3s[8] as the K8s distribution and Cilium[9] as the base CNI.

We use DigitalOcean's managed K8s Service for the cloud environment. The cluster is configured with the same specifications as the on-premises setup. For storage, the managed cluster automatically uses DigitalOcean's managed Block Storage service. The experiment is performed 5 times for each tool to evade any potential coincidences.

4.2. Network Experiment

The goal of the network experiment is to evaluate the load-balancing components of the cluster. In the following, we describe our experiment design and architecture. Afterward, we present our results, revealed by the experiment.

[3]https://metallb.io/
[4]https://docs.cilium.io/en/stable/network/lb-ipam/#services
[5]https://longhorn.io/docs/1.7.2/
[6]https://ceph.io/
[7]https://rook.io/docs/rook/latest/Getting-Started/intro/
[8]https://k3s.io/
[9]https://cilium.io/

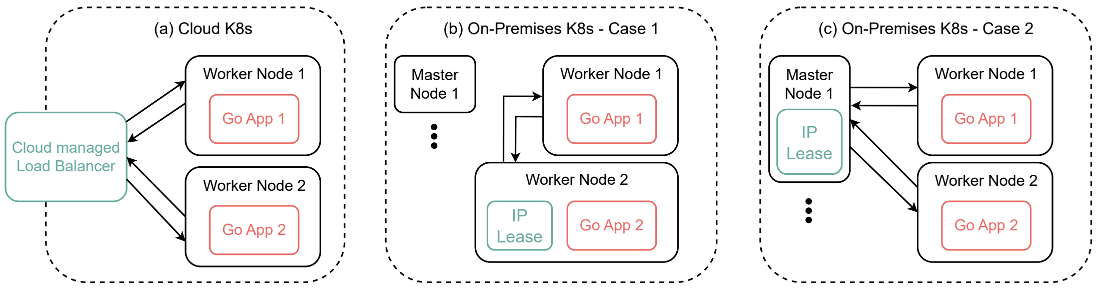

Figure 1: Experimental Design and Approach of the Network Test.

4.2.1. Network Experiment Design

The experiment architecture involves a simple client-server interaction[10]. As seen in Figure 1, a Golang-based HTTP server application is deployed in the cluster, with two replicas running on separate worker nodes. The worker exposes an endpoint that is routed out of the cluster using an ingress controller exposed via the IP address provided by the load balancer. A client application from a different network sends an HTTP GET requests to the server's endpoint every 100 milliseconds and logs the response. If there is a downtime detected, the application will measure the downtime and count the failed requests.

The cloud K8s cluster, as illustrated in Figure 1 describes the experiment architecture of the cloud experiment. In this setup, we simulate a node failure by shutting down a node containing the workload while logging the client's response. The managed load balancer should ensure that the workload is dynamically redistributed to the healthy node. For the on-premises environment, we need to slightly change the experiment architecture, because the tools are configured using Layer 2 mode. As already highlighted in Section 3, in Layer 2 mode, only one node in the cluster holds the lease for the service IP address at any given time. If traffic arrives on the node the holds the current lease, it then gets forwarded to the appropriate Pod running the workload. This fundamental difference in traffic routing introduces challenges for direct comparisons with the cloud setup. To address these challenges and ensure a fair evaluation, we split the experiment for the on-premises environment into two distinct cases:

1. **Worker Node Holding the Lease**: As illustrated in the first case in Figure 1, the worker node holds the service IP lease and hosts one of the workloads. When this node fails, the lease for the IP and the workload are interrupted, which is closer to the experiment performed in the cloud environment. However, it isn't fully representative, since the IP lease doesn't necessarily have to be announced by the worker node.
2. **Master Node Holding the Lease**: In the second case in Figure 1, the master node holds the service IP lease. Simulating a node failure on the master would require the load balancer to elect a new node to announce the IP lease. However, comparing this to the cloud environment would be unfair, as we don't interrupt one of the workloads.

For reference, we also tested the case where we simulated the failure on the worker node while the master held the lease. In combination with the other two cases, this will help us to precisely define the downtime that is created by the load balancer.

4.2.2. Network Experiment Results

For the cloud environment, the client reported an average of 2.60 ($\sigma = 0.89$) failed requests, resulting in a downtime of 2.77 seconds ($\sigma = 0.57$). A small amount of failed requests have to be expected, since there are requests being sent as the failed node was going offline.

In the on-premises environment, we can see that both Cilium and MetalLB returned similar results for the first case of the experiment, where we simulate a node failure on the worker node with a workload

[10] https://github.com/jacolate/KubernetesTesting

Table 1
Results for the Network Experiment (μ/σ), values have been averaged over five runs of the experiment

Case	Fig. 1	Failure Simulated	*Current Lease*	Total Requests	Failed Requests	Downtime (sec.)
Reference	-	Worker	Master	800	74.40/2.30	16.48/0.52
Cloud	(a)	-	-	800	2.60/0.89	2.77/0.57
Cilium	(b)	Worker	Worker	800	74.40/6.69	19.56/2.44
	(c)	Master	Master	800	12.40/13.88	6.15/6.61
MetalLB	(b)	Worker	Worker	800	74.60/6.38	17.33/1.46
	(c)	Master	Master	800	0.00/0.00	0.00/0.00

and the IP lease. Both MetalLB and Cilium had an average of ≈ 75 failed requests. If we compare this to our reference case, where we have an average of 74.40 ($\sigma = 2.30$) failed requests, we can see that these values are similar. This shows that created downtime doesn't necessarily come from the load balancer, but from other cluster internals. The real impact of the load balancer on the failed requests can be observed in the second case of the experiment. Here MetalLB showed no failed requests at all, whereas Cilium showed an average of 12.40 failed requests, but with a high standard deviation of 13.88.

4.3. Storage Experiment

Similar to the network experiment, we designed an experiment to evaluate the performance and reliability of the storage tooling. The following section outlines the design and architecture of the storage experiment, followed by a presentation and analysis of the results obtained.

4.3.1. Storage Experiment Design

To evaluate the different tools for supporting distributed storage in K8s, we developed a custom K8s workload consisting of an application written in Golang and a MySQL database. At the start of the experiment, the application establishes a connection to the database and writes a unique character sequence to it (Figure 2). Afterward, the application enters a loop, validating the character sequence every 100 milliseconds.

The experiment is divided into two scenarios. In the first scenario, the workload is simply rescheduled to a different node. In the second scenario, a simulated node failure is performed by manually shutting down a node. Both scenarios allow us to measure how long it takes for the Golang application to re-establish a connection to the database. We also verify whether the previously written data remains intact after each scenario.

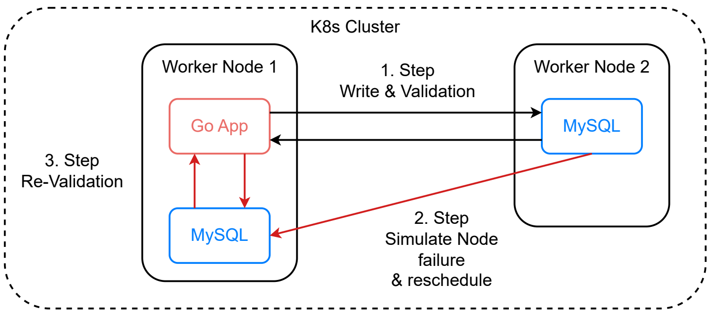

Figure 2: Experimental Design and Approach of the Storage Test.

4.3.2. Storage Experiment Results

In the cloud environment, the first scenario, involving rescheduling the MySQL deployment, resulted in the Golang application losing its connection to the database for an average of 7.42 seconds ($\sigma = 0.40$). However, once the connection was re-established, the application successfully validated the data in the database. The second scenario, simulating a node failure, encountered some issues. After the node crash, K8s automatically rescheduled the deployment to another node. However, the Pod failed to start and remained stuck in the *Creating* phase. The root cause was that the Block Storage volume was still attached to the failed node, preventing it from being remounted to the new node.

For on-premises, the results of the first scenario were similar to the cloud setup. With Rook & Ceph installed as a storage provider, the Golang application lost its connection for an average of 4.02 seconds ($\sigma = 0.40$). In contrast, using Longhorn, took significantly longer, with an average downtime of 20.08 seconds ($\sigma = 1.63$). The second scenario in the on-premises environment produced results similar to those observed in the cloud. After the node failure, the Pod could not be started because the volume remained attached to the failed node, preventing it from being remounted. This issue was consistent across both Rook & Ceph and Longhorn.

5. Discussion

The evaluation of the network tooling showed that the load balancing capabilities of an on-premises environment are similar to the cloud environment. However, the results also showed cases where we have an increased number of failed requests in the on-premises environment compared to the cloud environment. As highlighted, most of these failed requests aren't coming from the load balancer but from other cluster internals, which require further investigation.

For storage, the experiment showed, that Rook & Ceph can achieve similar results as their cloud counterpart. Longhorn however, performed worse, with significantly longer recovery times. The reason for this discrepancy requires further investigation. In the second scenario, all tested tools—including Rook & Ceph, Longhorn, and the cloud environment experienced the same issue: the inability to remount the storage volume to a new node due to its attachment to the failed node. Notably, Longhorn's documentation acknowledges this as expected behavior and requires administrative intervention[11].

The proposed experiment and the results underline a few limitations. First, the selection of tools was limited to the most popular ones. Numerous alternative tools exist, each with potentially distinct capabilities, limitations, and performance characteristics. Secondly, the network tools were only tested in the Layer 2 configuration. Using the BGP configuration instead should, in theory, be preferable. Furthermore, the comparison between cloud and on-premise tooling was based on selected functional experiments. A more comprehensive benchmark may provide more detailed results.

6. Conclusion and Future Work

This paper presented an approach to compare on-premises and cloud-based K8s environments, specifically focusing on network and storage tooling. To answer our research question, we conclude that it is possible to achieve comparable functionality in on-premises environments in the area of storage and networking using appropriate tools. MetalLB and Ceph & Rook delivered comparable results to the cloud environment, demonstrating similar performance in terms of load balancing and storage functionality. While our research provided valuable insights, there is still room for improvement. For our future work, we want to expand the number of tools tested for a more comprehensive evaluation. Additionally, testing network tools in BGP mode over Layer 2 mode, could uncover a performance increase. Finally, expanding the scope to include other areas of K8s, such as monitoring, automated deployment, and authentication, would offer a more comprehensive understanding of the trade-offs between cloud-based and on-premises K8s setups.

[11]https://longhorn.io/docs/1.7.2/high-availability/node-failure/

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT-4 and DeepL Write in order to: Grammar and spelling check, Paraphrase and reword. After using the tool/service, the authors carefully reviewed and edited the suggested changes to ensure that the intended meaning of the content remained unchanged. The authors take full responsibility for the publication's content.

References

[1] M. A. Rodriguez, R. Buyya, Container-based cluster orchestration systems: A taxonomy and future directions, Software: Practice and Experience 49 (2019) 698–719. URL: https://onlinelibrary.wiley.com/doi/10.1002/spe.2660. doi:10.1002/spe.2660.

[2] P. Kayal, Kubernetes in Fog Computing: Feasibility Demonstration, Limitations and Improvement Scope : Invited Paper, in: 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), IEEE, New Orleans, LA, USA, 2020, pp. 1–6. doi:10.1109/WF-IoT48130.2020.9221340.

[3] K. Takahashi, K. Aida, T. Tanjo, J. Sun, A Portable Load Balancer for Kubernetes Cluster, in: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, ACM, Chiyoda Tokyo Japan, 2018, pp. 222–231. URL: https://dl.acm.org/doi/10.1145/3149457.3149473. doi:10.1145/3149457.3149473.

[4] M. Packard, J. Stubbs, J. Drake, C. Garcia, Real-World, Self-Hosted Kubernetes Experience, in: Practice and Experience in Advanced Research Computing, ACM, Boston MA USA, 2021, pp. 1–5. URL: https://dl.acm.org/doi/10.1145/3437359.3465603. doi:10.1145/3437359.3465603.

[5] L. M. Ruiz, P. P. Pueyo, J. Mateo-Fornes, J. V. Mayoral, F. S. Tehas, Autoscaling Pods on an On-Premise Kubernetes Infrastructure QoS-Aware, IEEE Access 10 (2022) 33083–33094. URL: https://ieeexplore.ieee.org/document/9732997/. doi:10.1109/ACCESS.2022.3158743.

[6] K. Manaouil, A. Lebre, Kubernetes and the Edge?, PhD Thesis, Inria Rennes-Bretagne Atlantique, 2020.

[7] S. K. Mondal, R. Pan, H. M. D. Kabir, T. Tian, H.-N. Dai, Kubernetes in IT administration and serverless computing: An empirical study and research challenges, The Journal of Supercomputing 78 (2022) 2937–2987. URL: https://link.springer.com/10.1007/s11227-021-03982-3. doi:10.1007/s11227-021-03982-3.

[8] S. Böhm, G. Wirtz, Profiling Lightweight Container Platforms: MicroK8s and K3s in Comparison to Kubernetes, 2021.

[9] B. Johansson, M. Ragberger, T. Nolte, A. V. Papadopoulos, Kubernetes Orchestration of High Availability Distributed Control Systems, in: 2022 IEEE International Conference on Industrial Technology (ICIT), IEEE, Shanghai, China, 2022, pp. 1–8. doi:10.1109/ICIT48603.2022.10002757.

[10] Y. Rekhter, S. Hares, T. Li, A Border Gateway Protocol 4 (BGP-4), RFC 4271, 2006. URL: https://www.rfc-editor.org/info/rfc4271. doi:10.17487/RFC4271.