Towards Orchestration of Cloud-Edge Architectures with Kubernetes

Sebastian Böhm(✉) and Guido Wirtz

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

Abstract. Edge computing brings computational resources, reliable network infrastructure, and real-time capabilities closer to devices. Providing resources and workloads at the edge is mainly realized with container technology. The appropriate placement in terms of when, where, and how to provide containerized workloads is still an ongoing problem domain. Kubernetes is nowadays the state-of-the-art platform for containerized service orchestration to tackle these issues. Although Kubernetes misses capabilities like using real-time network metrics for scheduling and topology awareness, it is still used for realizing cloud-edge architectures. In this paper, we analyze current cloud-edge architectures implemented with Kubernetes and how they solve general requirements of edge computing and orchestration. Furthermore, we identify shortcomings in these implementations based on the fundamental requirements of edge computing and orchestration. Even if issues like obtaining network-related metrics and implementing topology awareness are solved well, other requirements like real-time processing of metrics, fault-tolerance, and the placement of container registries are in early stages.

Keywords: Edge computing · Edge orchestration · Cloud computing · Container orchestration · Kubernetes

1 Introduction

In recent years, edge computing has evolved as a supplementary layer to cloud computing where computational resources, so-called edge nodes, are placed close to data-generating entities, often Internet of Things (IoT) devices. The number of IoT devices is still growing and will exceed 30 billion by 2025.[1] In consequence, it is necessary to re-think the current concentration on cloud computing. The primary driver of edge computing is the need to provide location-aware resource and service provisioning because typical use cases like autonomous driving, smart traffic control systems, and other real-time services require low response times (<20 ms) [5]. Edge computing aims to achieve low latency, higher network

All links were last followed on June, 26, 2021.

[1] https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/.

S. Paiva et al. (Eds.): SmartCity360° 2021, LNICST 442, pp. 207–230, 2022.
https://doi.org/10.1007/978-3-031-06371-8_14

reliability, and enhanced privacy. The data is processed close to data-generating devices and not necessarily uploaded to the cloud [35]. However, the assignment of workloads to heterogeneous edge nodes, often low-end devices, requires real-time monitoring of available and demanded resources and is a fairly complex task. An efficient and sophisticated management is inevitable [37]. The workloads are usually handled by the usage of container technology. Compared to full-blown Virtual Machines (VMs), this lightweight way of virtualization, meanwhile seen as a de facto standard in edge computing, facilitates providing workloads on cloud and edge [24,29]. Containers are small in size, share the same kernel of an operating system, and are executed on a container runtime (e.g., Docker) that is available for a large number of platforms and architectures [4]. In particular, the process of container orchestration comprises the dynamic assignment of containerized workloads to the cloud or edge nodes based on available and demanded resources. Further, orchestration must provide a certain quality of service, e.g., in terms of realized latency or response time [7].

Nowadays, Kubernetes (K8s) is the state-of-the-art container orchestration platform and used in many application areas to achieve highly available, scalable, and fault-tolerant clusters.[2] Nevertheless, considering, using, and evaluating K8s for the implementation of cloud-edge architectures is still an ongoing process in research. In this paper, we want to discuss the current state-of-the-art K8s-based cloud-edge implementations. At this, we analyze essential requirements of cloud-edge architectures especially w.r.t. their orchestration. In addition, we identify existing architectural and conceptual proposals for edge and fog orchestration that are using K8s as a platform to realize orchestration following the essential requirements of edge computing. Furthermore, we examine potential weaknesses in already existing implementations with K8s and propose a set of architectural and conceptual improvements that are compliant with the claims of cloud-edge orchestration. This enables us to evaluate the fit of K8s as a single, uniform, and easy-to-use orchestration platform, which offers the possibility to manage large cloud-edge systems. Our research questions are as follows:

RQ1: What essential requirements of cloud-edge orchestration do exist, are covered by K8s, and what are the arising shortcomings?

RQ2: What are the potential benefits and drawbacks of already established K8s-based cloud-edge environments?

RQ3: Are the potential drawbacks of the established cloud-edge environments using K8s solvable without breaking the fundamental concepts of edge computing and architectural design of K8s?

To answer these research questions, we perform a literature review and obtain the most critical characteristics, requirements, and challenges regarding orchestration of cloud-edge environments. This helps us to evaluate the capabilities of K8s and contributes to RQ1. Targeting RQ2, we review the most popular architectures using K8s to analyze the state-of-the-art. In the next step, we map the obtained requirements running a cloud-edge system to the already established cloud-edge implementations with K8s and identify potential strengths and weak-

[2] https://kubernetes.io/.

nesses. Lastly, we evaluate conceptually, based on the considered studies, if the obtained K8s-based drawbacks are solvable and if it is realizable and compliant with cloud-edge architectures to answer RQ3.

The remainder of the paper is structured as follows: Sect. 2 presents a short introduction on edge computing, orchestration, and K8s in general. Section 3 discusses the related work evaluating edge (orchestration) solutions. In Sect. 4, we analyze K8s-based orchestration architectures, limitations, and potential solutions for K8s in cloud-edge environments. Limitations of the considered solutions will be part of Sect. 5. We critically discuss our findings and outline the limitations of our study in Sect. 6. We conclude our work with a summary and what we plan to do in the future (Sect. 7).

2 Conceptual Foundations

This chapter describes the conceptual foundations of edge computing and orchestration of cloud-edge architectures. Furthermore, a basic understanding of K8s as orchestration platform will be provided to support answering RQ1.

2.1 Edge Computing

Edge computing is a new paradigm where computational resources are placed close to data-generating devices. This placement strategy aims to increase the bandwidth and reduce latency [6]. In comparison to cloud computing, where all data is stored and processed in a centralized manner, edge computing acts as an additional decentralized layer to support and take load from the cloud [37]. Especially resource-constrained IoT devices can benefit from the increased bandwidth and ultra-low response times (<20 ms) because they are often performing real-time analytics or video surveillance activities with a large amount of traffic. In addition, IoT environments usually work in a geographically distributed way that suffers from unstable network connections [30]. Long distances coupled with many hops to the data-receiving and data-processing endpoints (i.e., servers in the cloud) intensify breaking the core requirement of low latency for critical applications. The higher the physical distance to the processing endpoints, the higher the transmission latency [35]. Edge computing can be classified into different technologies. The most frequent types, also called edge technologies, are Cloudlet, Mobile Edge Computing (MEC), Micro Data Center (MDC) [23], and Fog [8] (Fig. 1). All of them offer provision models to interact with the cloud. MEC has been introduced by Nokia in 2014 and provides computing, network, and storage resources, predominantly near mobile base stations. Following the general objectives of the edge computing paradigm, MEC aims to enable billion of resource-constraint devices with network capabilities low-latency access to process compute-intensive tasks, like real-time analytics [13]. Usually, MEC resources are placed next to mobile Radio Access Network (RAN) stations, where a high bandwidth enables fast and dynamic deployment of applications that are processing requests with real-time needs. This reduces the amount of data that

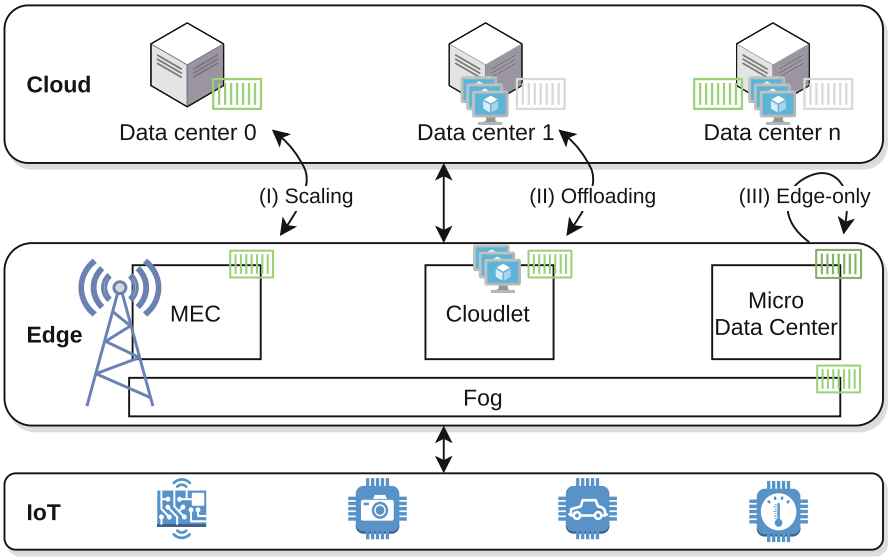

Fig. 1. Cloud-edge architecture with provision models and edge technologies.

is uploaded to the cloud considerably and mitigates network congestion. However, it is not clearly defined whether MEC layers should replace the cloud layer completely. Data may or may not be still forwarded to the cloud [2,8].

A cloudlet is a decentralized set of devices that form a virtualized cluster for running low-latency applications. Cloudlets are self-managing, easy-to-deploy by local maintainers, and bring the cloud one step closer to the edge. They require a reliable, high bandwidth internet connection because workloads are transmitted as VM overlays that are executed on top of an already installed base image. In addition, it is one core requirement to shift those VM overlays rapidly if served devices change their physical location, like vehicles in smart traffic control systems. Originally, cloudlets are supposed to run VM overlays to serve requesting devices [34]. However, [3] considered containers for application migration in cloudlets and [31,32] linux containers as lightweight alternative.

Micro Data Centers (mDCs) are similar to cloudlets because they want to achieve low-response times by extending the cloud layer. In contrast to cloudlets that are set up in a decentralized manner maintained by independent owners, mDCs run in one single, isolated, and secured unit. This comprises protection in terms of physical as well as software-related access control because multi-tenancy is an essential requirement of mDCs. They usually have a reliable, fast, and persistent connection to the cloud to enable data exchange.[3]

[3] https://www.networkworld.com/article/2979570/microsoft-researcher-why-micro-datacenters-really-matter-to-mobiles-future.html.

Lastly, the fog paradigm, which is often interpreted as synonym to edge computing [30,37,41], can reduce the latency between cloud and IoT devices. In contrast to the presented edge technologies in the former paragraph, the fog computing layer is one step closer to the service requesting devices and is designed in a decentralized manner. A large number of heterogeneous fog devices is required to manage, cooperate, and communicate to achieve the objective of low-latency [9,40]. It is still an open question if there is a fundamental difference between edge and fog computing. However, both technologies are sharing a common set of services (e.g., computation, storage, and networking) in a similar way as an intermediate layer between the cloud and IoT layer. We treat edge and fog as similar and related technologies and do not introduce a further differentiation, as many other authors [9,30,37,41].

As indicated in Fig. 1, applications with low-latency requirements can be deployed in different ways. The three most frequent so-called provision models are (I) scaling, (II) offloading, and (III) edge-only deployments, even if the edge layer is not supposed to replace the cloud completely [2,8]. (I) Scaling to the edge, also called distributed offloading, keeps the application running in the cloud and edge layer simultaneously. This provision model applies especially if the cloud and edge layer need to work together because both are running out of resources. For example, the edge node may reach the available amount of storage and the cloud may violate the latency requirement [1]. (II) Offloading from cloud to edge and vice versa is the most frequently discussed and used approach. Applications and tasks are moved to the edge layer to achieve better response times, e.g., due to periodic high load. Equally, applications may be shifted to the cloud when the load decreases [18]. (III) Edge-only deployments are also possible, where the applications are placed only on the edge layer and moved across different edge nodes, to fulfill the needed latency. Offloading is not necessarily destined for this way of placement. However, many solutions perform a mixed approach even if they focus on edge placement in specific [36].

2.2 Orchestration of Cloud-Edge Architectures

Cloud-edge environments introduce new complexities that arise when new layers with different provision models are added to the cloud. Cloud-Edge orchestration comprises all activities to distribute a set of applications to the cloud, edge, and IoT layer given a set of objectives. There are multiple objectives involved to run this architecture efficiently (Sect. 2.1).

The essential parts are performance-related aspects like the optimal placement of applications dependent on the real-time latency and bandwidth of devices in the architecture to the origin of potential requests. Complex optimization and scheduling models are required to distribute applications according to resource demand and supply of CPU, memory, disk, and network utilization [37]. In addition, cloud-edge architectures must be fault-tolerant and resilient if some nodes in the network become unavailable [42]. The decentralized, distributed, and large-scale nature of edge layers intensifies the complexity because often edge resources are equipped with low-power devices [18,19]. Associated with

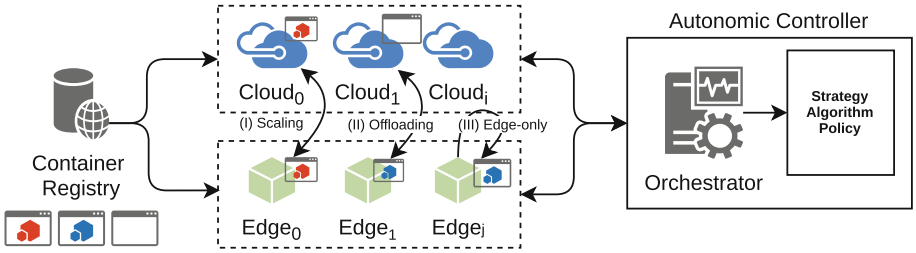

Fig. 2. Generic orchestration architecture, based on [10].

large-scale deployments, orchestration must provide extensibility to dynamically scale up and scale down the number of available nodes [28]. Security is also an important part of cloud-edge environments because edge technologies like cloudlets and mDCs are shared across different tenants (Sect. 2.1). In addition, they may be publicly accessible [41]. Hence, concepts like authentication, authorization, and user access control are inevitable running cloud-edge systems [42].

All types of provision models require a fast and efficient way of moving applications through the different layers. For this, container virtualization can be seen as the de facto standard to run applications in this regards. Containers include the application, all necessary libraries, and the runtime environment. They run on top of a so-called container engines, like Docker, and share the same kernel of an operating system. During runtime, the containers are isolated and can only use a previously defined amount of resources, such as CPU or memory. Running applications in this way involves several advantages: Containers are small in size and provide fast startup times [4]. Especially the portability of containers has led to wide acceptance in edge computing [24,29].

Figure 2 shows a generic container orchestration architecture based on [10]. Applications are provided by a container registry and executed on cloud and/or edge nodes. The autonomic controller consists of an orchestrator that implements an orchestration strategy, algorithm, or policy responsible for assigning those applications on different nodes. At this, different provision models can be realized, as explained in the former section.

2.3 Kubernetes

K8s can be used to run workloads on a set of nodes and to implement the described generic orchestration architecture from the last chapter. We present a minimal working cluster in Fig. 3. It consists of at least one master node, also called control plane, that carries all system-related services to run the cluster. Further, at least one worker node is needed to run the assigned workloads.[4]

The master node contains a set of services to manage the set of worker nodes. All system-related services are running as containers and can be distributed

[4] It is even possible to run single-node clusters by attaching workloads to the master node; however, this should not be done in production environments.

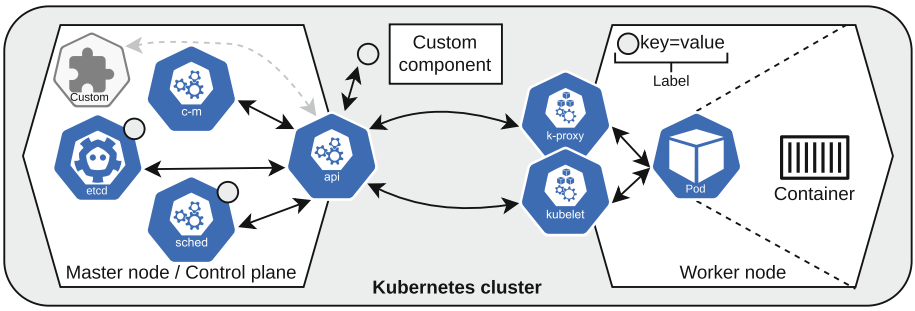

Fig. 3. General Kubernetes architecture.

among different nodes to achieve high availability. The controller-manager (*c-m*) observes all nodes running in the cluster and keeps track of the current state. For instance, as part of the controller manager, the replication manager can restart pods on other nodes in case of failing nodes. The Kube Scheduler (KS), in Fig. 3 named *sched*, assigns containerized workloads that are running in pods to worker nodes based on filtering and scoring to find the most appropriate node for deployment. For example, previously defined constraints and available resources on nodes are taken into consideration. K8s holds the cluster data in a strongly consistent distributed key-value store *etcd*, for example, currently running assignments. All data will automatically be distributed across all running etcd instances to achieve redundancy. The *api* component is used to interact with all system-related components, exposed as REST API. Not shown as optional component in Fig. 2 is the Horizontal Pod Autoscaler (HPA) that can dynamically increase and decrease the number of pods based on CPU and memory utilization.[5] All worker nodes run a so-called *kubelet* that interacts with the controller manager on the master node. This component manages the lifecycle of pods based on the commands from the controller manager. Furthermore, this component transmits the current node status to the control plane. Lastly, the kube-proxy (*k-proxy*) is used to make workloads available via the network in the form of services by opening ports and forwarding traffic.[6]

As annotated in Fig. 3, K8s allows to replace and modify several components. The scheduling component can be modified or replaced entirely. Custom labels can be assigned to nodes, for example, to enrich the information basis for the scheduling process. Possible for a customized scheduling and scaling are also external applications that are interacting with the api or containerized components that can even be deployed as control plane components, for example, if the scheduler is supposed to be replaced.[7]

[5] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/.
[6] https://kubernetes.io/docs/concepts/overview/components/.
[7] https://kubernetes.io/docs/concepts/extend-kubernetes/.

3 Related Work

The works that are related to our survey are three-fold. Firstly, we identified those works that defined and investigated fundamental characteristics of cloud-edge computing and performed a subsequent evaluation based on a set of criteria.

[45] conducted a literature survey on fog computing and similar or related computing paradigms, like the described edge technologies in Fig. 1. Besides general aspects of fog computing, they stated limitations, research directions, and potential solutions for fog orchestration. Limitations and research directions were mapped to potential solutions from various authors. Although cloud-edge orchestration was not the focus of this work in specific, many useful aspects were included for future research. In [39], a comprehensive literature survey was performed to obtain the state-of-the-art orchestration challenges, also for edge and fog computing. The authors also presented a mapping between these challenges and provided a list of corresponding potential solutions as described in different studies. [26] followed this qualitative approach and investigated several requirements, like infrastructural-, platform-, and application-related criteria. These criteria were used to evaluate established fog architectures and use cases. In terms of orchestration, the authors collected a comprehensive set of resource allocation and scheduling algorithms as a discipline of orchestration. Again, these obtained criteria were used to evaluate a selected set of solutions that tackle this problem domain. In [41], the motivation, challenges, and opportunities in edge computing were discussed. The authors provided a reasonable set of aspects that must be handled, especially in regards to orchestration-related activities.

Secondly, the works that evaluate edge (orchestration) architectures are related to our study. Whereas the studies in the former paragraph explicitly evaluate general challenges in edge and fog computing, the following studies target cloud-edge or edge orchestration in specific. For example, [19] defined several requirements for orchestration in terms of node-handling capabilities (e.g., joining/leaving the cluster, scheduling, and device mapping to containers). They evaluated container orchestration tools like Mesos, Kubernetes, and Docker Swarm. Although they do not compare different cloud-edge architectures implemented with these tools, their defined requirements contribute to the collection of features an orchestration system should comply with. Similar to the studies in the first part, [42] analyzed the state-of-the-art of fog orchestration. Their study provides a detailed overview of requirements and what orchestration must fulfill. Based on these obtained criteria, they evaluated well-established and standardized fog orchestration architectures. They conclude that most of these cloud-edge architectures can deal with the challenges in fog computing. However, there was no practical implementation covered.

Thirdly, those evaluations that focus only on one certain aspect of cloud-edge orchestration are further in relation to our work: General architectural and algorithmic challenges, for example, for resource provisioning and scheduling, were part of [25]. The qualitative literature review leads to a comprehensive set of limitations that are mapped to potential solutions. In addition, starting points were added to support further investigations. [1] also provided a comprehensive

survey in a subarea of cloud-edge orchestration. They analyze offloading strategies based on a set of criteria that was theoretically derived. A similar approach was made by [43]. They analyzed several offloading algorithms based on a previously set of criteria and presented an overview.

Although many of the approaches investigated several challenges, research directions, and potential solutions for issues in cloud-edge orchestration, there is no comprehensive overview of K8s-based cloud-edge and edge architectures in specific. Therefore, we provide a comprehensive evaluation of those modifications according to the requirements of cloud-edge architectures.

4 Kubernetes as Edge Orchestration Platform

To get an overview of the current state-of-the-art of cloud-edge orchestration with K8s, we analyzed 18 research papers. We obtained these papers from different databases like IEEE Explore, SpringerLink, and ArXiv. We used the term `Kubernetes` $\wedge$ `(edge` $\vee$ `fog)` $\wedge$ `(computing` $\vee$ `orchestration)`.

Finally, we only considered those papers which identified potential shortcomings of K8s and provided a set of solutions by retaining K8s. In total, seven out of 18 publications were excluded. Besides a short introduction on the considered solutions, this chapter also discusses limitations of K8s for edge orchestration and presents potential solutions stated in the literature to answer RQ2.

4.1 Kubernetes-Based Edge Orchestration Architectures

The set of K8s-based edge orchestration systems can be basically separated into three parts, namely (open-source) frameworks and solutions that are tackling essential orchestration activities in edge computing, solutions that applied custom modifications and extensions to K8s for cloud-edge orchestration, and finally those solutions that orchestrate only the edge-layer with a modified K8s. This chapter shortly outlines the general approach of these platforms and solutions.

Platform-Based Solutions. The first part comprises frameworks like KubeEdge (KE)[8], Baetyl[9], OpenYurt (OY)[10], or ioFog[11]. These frameworks often use K8s as an underlying platform without any modifications to run the required system-related services and the target applications as containerized workloads. These platforms usually provide a mechanism to enable an easy setup of cloud-edge architectures. They provide routines to deploy infrastructure components like virtual networks, additional middleware like message brokers, preconfigured service- or event bus endpoints, and management capabilities. Users can easily connect devices to the architecture and monitor their current

[8] https://kubeedge.io/en/.
[9] https://baetyl.io/en.
[10] https://github.com/openyurtio/openyurt.
[11] https://iofog.org/.

state. Platforms like KE advertise resource optimization to enable the usage of the platform with low-end devices, which are very common in the field of edge computing. The primary lack of those platforms is the missing dynamic workload allocation capability. Those platforms are not able to perform intelligent placement decisions based on the current resource utilization on edge devices, offloading of containerized workloads from cloud to edge and vice versa, and finally scaling out to the edge [17]. In consequence, some authors used the modifiability and extensibility of K8s (Sect. 2.3) to implement custom cloud-edge and edge architectures, which are described in the following paragraphs.

Custom Cloud-Edge Architectures. Secondly, we present architectures using cloud and edge layers in collaboration to provide orchestrations.

[17] propose *KaiS*. This scheduling framework dispatches requests decentralized at the edge and cooperates with the cloud to improve the long-term rate of request processing and system overhead. Furthermore, they used centralized service orchestration in the cloud to assign containerized workloads to edge nodes. The architecture made usage of learning dispatching and orchestration techniques. The authors obtained that *KaiS* can distinctly reduce the throughput rate and scheduling costs compared to K8s or other greedy approaches. A scalable and custom solution was described in [21] as a multi-objective optimization problem to minimize interference and energy consumption of deployments. They deployed custom containers on all nodes in the cluster to meet the requirements of co-allocation of dependent workloads on a single node to reduce the carbon footprint of the entire cloud-edge orchestration. Compared to a First Come First Serve algorithm, *KEIDS* and the introduced optimizations on *KEIDS* led to different improvements in reducing carbon footprint, performance, and energy minimization. The solution, named *Swirly*, presented in [15], aims to fulfill several requirements of cloud-edge orchestration. *Swirly* is a scheduler running in the cloud and creates service topology to support scheduling. It supports large-scale and topology-aware deployments with a minimal number of container instances by taking real-time resource utilization (e.g., CPU, memory, and latency) into consideration. They deployed a custom container on all nodes to achieve the previously mentioned goals. As a benchmarking result, they concluded that the proposed solution is able to manage up to 300000 devices. [33] proposed a cloud-edge architecture with network-aware scheduling for an air monitoring service. They implemented several extensions, like custom schedulers, and applied modifications to K8s to support location-aware scheduling. As a result, they compared their network-aware scheduler with the default KS and other approaches based on integer linear programming. They showed that their proposed architecture with the modified scheduler leads to a noticeable reduction in network latency. A further notable example of K8s-based implementations is the work of [16]. The proposed architecture aims to get K8s ready for the orchestration of geographically distributed clusters. They placed additional components, e.g., for measuring network latency on each node, and proposed a custom scheduler for scheduling activities. This solution targets the cloud and edge layer for placing workloads and performs periodic latency probes to determine optimized latency-

aware deployments. An evaluation has shown that the architecture can adjust the deployment based on real-time conditions. [20] designed an architecture with the focus on fault-tolerance by defining application isolation, data transport, and multi-cluster management. They deployed fault-tolerant Kafka clusters in cloud and edge to achieve a reliable storage layer. In addition, they deployed master nodes on the cloud in high availability mode. Finally, they evaluated that a two-node failure does not mitigate the operational state of the cluster. Lastly, the solution of [44] is worth mentioning. They propose *Fogernetes*, a fog computing platform, which is based on a labeling system to achieve network-aware and resource-oriented deployments. The deployments are based on assigning a set of key-value pairs to nodes and adding those to the deployment file as well. As a case study, they deployed a video streaming architecture with a camera, client, middleware, and central processing unit in the cloud and showed that K8s followed a location-aware deployment based on the defined mappings.

Custom Edge Architectures. Thirdly, we give a short conceptual overview of solutions that considered only the edge layer for orchestration activities.

[22] propose a fog architecture based on K8s that aims to deploy multi-container applications on resource-limited fog devices. They introduced several plugins to the default KS to achieve more efficient and location-aware distribution of containers, for example, by distributing multi-container deployment on neighboring nodes. As a result, they conclude that the service quality was not mitigated and the procedure might evolve as common practice to utilize fog architectures. In [12], a decoupled and native modification was executed on K8s to achieve location-aware, latency-aware, and fault-tolerant deployments. Based on a decoupled architecture, deployments are calculated by the usage of an external component. This component passes the created deployment to an unmodified K8s cluster architecture. The native modification runs on K8s as an additional component and interacts directly via API calls to achieve a more robust integration into K8s. Finally, the authors performed several experiments on allocation costs and failover time to evaluate the proposed modifications. [27] proposes an edge solution for industrial IoT to reduce the scheduling time by applying single-step scheduling, especially in the field of 5G. The implemented custom scheduler achieves latency-aware deployments and optimizes the deployment time and temperature of all nodes in the cluster. As a result, they reduced the scheduling time and considered additional metrics, like latency, jitter, and packet loss in the scheduling process. In [11], the inherent issues in the scheduling process were addressed by implementing an agent-based approach to reduce the load on the master node while scheduling. The scheduler runs instead of the default scheduler and addresses only the fog layer. The authors followed an agent-based approach that shifts selected scheduling tasks of K8s, like node filtering and scoring, to agents, which are placed on nodes in the architecture. An evaluation has shown that the agentified scheduler significantly needed less time for deployments with a small number (<10) of replicas.

4.2 General Limitations of K8s for Edge Orchestration

We distinguished the general limitations of K8s as an orchestration platform in resource awareness and architectural shortcomings. These categories exhibit substantial obstacles running K8s in production for cloud-edge architectures.

Resource Awareness. Dealing with real-time resource demands and supplies in terms of the current CPU, memory, storage, and network utilization can be seen as a major challenge in orchestrating cloud-edge architectures. Since conditions are dynamically changing and shifting actions must be initiated in an appropriate amount of time, K8s shows several limitations in this regard. Frequently mentioned are the limited capabilities of the default K8s scheduler to deal with other resources like CPU and memory utilization. Especially for small edge and IoT devices other resource types are also essential, e.g., the current quality of the network connection [12,21,22,27,33]. Other authors using K8s as platform for orchestration mention that the missing capabilities taking the resource consumption for shifting applications or tasks from cloud to edge as further scheduling metric is a major drawback. Considering energy efficiency might be important for mitigating unnecessary offloading actions and reducing carbon footprint [21,33].

Cloud-edge deployments benefit from latency- and bandwidth-aware orchestrations. That means it is desirable to place multi-component applications close to each other in order to reduce the communication cost [22,33]. Following the main objective of edge computing, applications must be placed close to requesting devices. Therefore, latency is one of the most important measurements that should play an essential role in scheduling container workloads to edge devices. K8s does not provide a built-in mechanism to run deployments based on bandwidth and latency that might limit the application in cloud-edge environments [12,16,21,22,33].

Lastly, the KS shows several obstacles for the usage of K8s in edge computing. As pointed out in Sect. 2.3, the KS assigns workloads in the form of pods to nodes based on filtering and scoring by node priority calculation. K8s takes the workloads to be scheduled one-by-one at a time and does not preferentially consider assigned priorities to pods to a set of available nodes [17,22,27,33]. Furthermore, developers are allowed to define the resource requests of pods statically. If those definitions are far away from the actual resource usage, nodes might be underutilized. The scheduling algorithm might lead to unassigned pods if no resources are left on nodes, even if the containers in the pods to be scheduled could be distributed among different nodes [22,33]. The missing understanding of the network topology is one further downside. K8s interprets the set of worker nodes as homogeneous with similar capabilities, although cloud-edge architectures usually consists of heterogeneous nodes in a geographical distributed architecture [12,14,16,17,21,22,33,44]. Without a topology, K8s can not realize location-aware deployments that are required for the different provision models and edge technologies (Sect. 2.2).

Architectural Shortcomings. Besides the resource-related shortcomings of K8s, there are also architectural issues that are part of related works (Sect. 3). By definition, the decentralized organization is an essential characteristic of edge computing (Sect. 2.1). However, K8s exhibits a centrally organized control plane that manages the lifecycle of a large set of nodes and applications running on them. Even though the control plane can run in a high availability manner [20], this architecture violates the decentralized notion [11,17,22]. In addition and solid relation to the topology criteria in the former chapter, the distributed nature of edge-computing is degraded because K8s does not hold a network topology to model geographically distributed and independent cloud-edge clusters [17]. Lastly, one single K8s cluster can at most handle 5000 nodes with 150000 pods and 300000 containers in total.[12] A very large edge cluster requires additional coupling, e.g., by the usage of cluster federation [14,15].

4.3 Potential Solutions for K8s as Edge Orchestration Platform

This section provides a short overview of potential solutions to overcome the most critical challenges in edge computing, according to the former section.

Providing Resource Awareness. The missing capability of dealing with other resources like CPU and memory was the most frequently mentioned downside taking K8s as an orchestration system, especially in regards to latency, bandwidth, energy consumption, and costs to determine future scheduling activities. Some authors implemented a custom way to collect additional metrics, like bandwidth and latency, with containerized applications and agents that are running alongside the K8s components. These metrics are further analyzed by custom schedulers that are running as normal containers alongside the default KS to alter deployments [11,15–17,21,27]. K8s also offers to implement custom and native schedulers that can also run together with or without the default K8s scheduler.[13] Furthermore, there are already implementations that are using the default KS with scheduler extenders that allow to alter scheduling decisions of the filtering and scoring step of the default KS [22,33]. As pointed out in Sect. 2.3, the default KS starts with filtering and prioritizing nodes based on predicates. The priority step can be revised and added to the default KS by recompilation, for example, as performed by [22]. Lastly, creating new deployment manifests based on custom containers was already done by researchers to modify the orchestration [12,44]. Finally, the issue of potentially unscheduled pods must be addressed, for example, by splitting up deployments and distribute them to different nodes [22]. Moving already running pods to different nodes might also be an option [16]. This solution, however, does not address the issue of static resource assignments to pods and potential under- and over-utilization of the infrastructure. According to the investigated solutions, real-time data analysis

[12] https://kubernetes.io/docs/setup/best-practices/cluster-large/.

[13] https://kubernetes.io/docs/tasks/extend-kubernetes/configure-multiple-schedulers/.

is desired, respectively, must be regularly revised to avoid not properly chosen resource assignments.

Implementing Cloud-Edge Architectures. The orchestration of cloud-edge architectures should follow a decentralized organization and should not rely on a master node as it is the case with K8s. However, this requirement can be relaxed because it is common practice to introduce centralized schedulers for the edge, as shown by several examples [16,17,44]. Agent-based and other decentralized approaches together with centralized components have already been realized with K8s, even specific for edge-computing [11,17]. The missing awareness of network topology can be seen as a major drawback, perhaps the most serious. As explained in Sect. 2.2, edge orchestration is supposed to implement various provision models. For example, to perform (II) edge offloading from cloud to edge or vice versa requires knowledge about the position and assignment of nodes to layers (Fig. 1). Therefore, K8s must provide a way to define a network topology in order to orchestrate cloud-edge environments with different edge technologies. As indicated in Sect. 2.3, labels can be assigned to nodes in order to implement a network topology. Based on so-called affinities and anti-affinities, the set of usable nodes for pod deployment can be limited in a reasonable way. The investigated solutions commonly used affinities to express the corresponding layer (i.e., cloud, edge, or IoT) a node is assigned to and additional context-related information, like the target location [12,16,22,44] or the device type [33].

5 Limitations of the Proposed Solutions

Although the proposals based on K8s provided contributions to get K8s ready for the edge, there are still some issues that must be considered running cloud-edge environments in production. In this chapter, we present an overview of the investigated solutions and the identified drawbacks. Again, we distinguished the limitations in resource-related and architectural shortcomings.

5.1 Resource-Related Solutions

Table 1 shows an overview of which requirements in terms of resource-awareness are supported (●), partially supported (❍), or not supported (*no circle*). Also, we mention where no details could be found (✲).

First, most of the proposals modifying K8s for edge environments considered the *cloud* and the *edge* layer. Architectures where the cloud holds the K8s master node with control planes or custom scheduler components and the edge layer, consisting of a set of worker nodes, are a frequent architectural design. However, some solutions only consider the edge layer, which runs all necessary components. In regards to the scheduling process, most of the authors used *custom* containerized schedulers that are replacing the *native* default scheduler in K8s completely. [12,33] used an unmodified version of the *native* scheduler (❍) in collaboration with a *custom* scheduler [12] or an *extender* [33]. In addition,

Table 1. Comparison of resource-awareness in different Kubernetes implementations for cloud-edge architectures.

		Cloud	*Edge*	*IoT*	*Recompiled*	*Native*	*Extender*	*Custom*	*CPU*	*Memory*	*Disk*	*Energy*	*Latency*	*Bandwidth*	*K8s API*	*Custom*
Authors	**Year**	**Layer**			**Scheduler**				**Resources**				**Network**		**Metrics**	
[17] Han	2021	●	●					●	●	●	●		●			●
[22] Kayal	2020		●		●				❍	❍	❍		❍		❍	
[21] Kaur	2020	●	●					●	●	●		●		●		✲
[12] Eidenbenz	2020		●			❍		●	❍	❍	❍		●		●	
[15] Goethals	2020	●	●					●	❍	❍	❍		●		❍	●
[27] Ogbuachi	2020		●					●	●	●	●		●			●
[33] Santos	2019	●	●			❍	●	●	❍	❍	❍		❍	❍	❍	
[11] Casquero	2019		●					●	❍	❍	❍				❍	
[16] Haja	2019	●	●					●	❍	❍	❍		●		❍	●
[20] Javed	2018	●	●			❍			❍	❍	❍				❍	
[44] Wöbker	2018	●	●			❍			❍	❍	❍				❍	

●=fully supported; ❍=partially supported; *no circle* = not supported; ✲ = n/a

there are proposals that do not include customization of the scheduler and work with implicit scheduling [20] respectively affinities and anti-affinities to implement scheduling to the cloud and edge layer [44]. In general, all solutions are aware of *CPU*, *memory*, and *disk* resources. However, most of the solutions take the static resource assignments (❍) for scheduling decisions. Only a few approaches scheduled based on dynamic real-time metrics (●). One approach considers the *energy* consumption, which might be important for devices running on battery [21]. Latency-aware deployments are crucial in edge-deployments. Some authors achieved those deployments by periodically measuring the network latency (●) between nodes. Other solutions only consider predefined and static assignments (❍). The currently available bandwidth is rarely considered in the investigated solutions. There are predefined static (❍) definitions of bandwidth [33] as well as periodic (●) checks [21]. The *K8s API* is used to measure the current CPU and memory utilization to support the scheduling and scaling decisions. As clarified in Sect. 4.2, the set of considered metrics must be extended. Some solutions make usage of the unmodified K8s API (❍) without any modification or extensions. Also, we investigated solutions that deploy *custom* (●) containers in addition to the native metrics components [15,16]. However, there is one solution [12] that is replacing the metrics server compatible with the K8s API (●) to gather latency between nodes. Missing explanations and definitions were also the case (✲).

5.2 Architectural Solutions

After highlighting the resource-related shortcomings, we will review the architectural implementations according to the obtained requirements. Topology plays a major role because edge-cloud orchestrators must be aware of the network they orchestrate to support all provision models and edge technologies.

A few authors allow explicit addressing (●) of *cloud* and *edge* layers for deployment or scheduling workloads. Other solutions [20,21] take the cloud and edge layer as a single system running orchestration and workloads. Often, only the edge layer is in the focus of orchestration activities, as shown in Table 2. Workloads are not supposed to be moved through the different layers. Some solutions, like [17,33,44] facilitate the *scaling* model, where applications can run on cloud and edge in a replicated manner (●). Due to the implemented scheduler algorithms, [20] and [21] support scaling to edge only implicitly (❍). Explicit *offloading* (●) from cloud to edge is rarely supported because most works take the cloud as a control layer, which is not supposed to run workloads. Implicit *offloading* (❍) might occur when nodes are failing and K8s resiliency feature is shifting and restarting the workload on another node. However, this is only possible for architectures where workloads can be assigned to the cloud. Nearly all works also enable *edge-only* deployments, where workloads are executed and

Table 2. Comparison of architectural capabilities in different kubernetes implementations for cloud-edge architectures.

		Topology			Provision model			Fault-tolerance			Container registry		
Authors	**Year**	*Cloud*	*Edge*	*IoT*	*Scaling*	*Offloading*	*Edge-only*	*Cluster*	*Control plane*	*Cluster storage*	*Cloud*	*Edge*	*Replicated*
[17] Han	2021	●	●		●	●	●	●			●		
[22] Kayal	2020		●				●				●		
[21] Kaur	2020	❍	❍		❍	❍	●	●				✲	
[12] Eidenbenz	2020		●				●					✲	
[15] Goethals	2020		●				●					✲	
[27] Ogbuachi	2020		●				●				●		
[33] Santos	2019	●	●		●	❍	●				●	●	●
[11] Casquero	2019		●				●				●		
[16] Haja	2019	●	●			●	●					✲	
[20] Javed	2018	❍	❍		❍	❍	❍		●	❍		✲	
[44] Wöbker	2018	●	●		●	❍	●					●	

●=fully supported; ❍=partially supported; *no circle* = not supported; ✲ = n/a

moved across the edge layer to achieve fast response times (●). Partial *edge-only* means that workloads are implicitly scaled-out (❍) by K8s, for example, if only edge nodes are available for deployment and users or the HPA increase the number of replicas. The investigated solutions poorly cover fault-tolerance. For our analysis, we only consider the fault-tolerance concerning the architecture of K8s itself, not the fault-tolerance of the deployed applications.

This follows our objective to identify architectural shortcomings. Even if most of the deployments offer high availability, a crashing master node that is running the *control plane* stops scheduling, scaling, monitoring, and the resiliency features in K8s and is not acceptable if these solutions operate in critical areas. [17,21] introduce geographically independent (●) K8s *cluster*s to provide fault-tolerance. However, the *control plane* is not replicated according to the recommendation deploying at least three master nodes (●) for production environments.[14] Only [20] implemented a fault-tolerant K8s architecture by considering three master nodes that are also running the distributed key-value store *etcd* as *cluster storage* (❍). Nevertheless, it might be worth considering to decouple etcd from the master nodes running the control plane components by deploying an external etcd cluster, which is interacting with the master nodes. Compared to the stacked cluster storage setup (❍), where etcd is distributed on the master nodes, an external storage cluster would achieve a higher degree of resiliency and reduce the load on the master nodes (●). This is especially recommended for setups that need to handle a large number of nodes.[15] Container registries are mainly provided by the cloud in a non-geographically *replicated* manner. The position of those registries is essential, enabling edge nodes to download container images very fast, for example, if workloads must be scaled, offloaded, or moved to achieve the edge-only model. Since latency reduction is a core characteristic of edge computing, this should also be reflected in placing container registries at appropriate locations. Only one publication [33] considered a fully replicated solution across *cloud* and *edge* (●). [44] provided a registry only at the edge (●) to enable fast download times. Often, placing strategies are neglected and neither specified nor discussed (✲).

6 Discussion

After the evaluation of the proposed K8s-based solutions, this chapter will provide a detailed discussion. In Sect. 6.1, we present our findings by answering our research questions from Sect. 1. Subsequently, Sect. 6.2 contains the limitations of our study. Finally, we conclude this chapter with a short assessment if the ambitions modifying and extending K8s should be retained (Sect. 6.3).

[14] https://kubernetes.io/docs/setup/production-environment/.
[15] https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/.

6.1 Findings

In the first chapters of this work, we analyzed fundamental characteristics of edge computing (Sect. 2.1), edge orchestration (Sect. 2.2), and K8s as container orchestration platform (Sect. 2.3). Furthermore, we presented and analyzed several studies that considered K8s as basis for cloud-edge orchestration (Sect. 4.1) to obtain the general limitations (Sect. 4.2). This procedure allows us to answer RQ1. The essential requirements of cloud-edge orchestration can be separated into resource- and architectural-related requirements. Resource-related requirements are the different *layers*, which are used by the orchestration architecture to operate. Furthermore, considering real-time *resource* utilization and providing *network* awareness in deployments are important requirements, ideally with dynamic consideration. In regards to architectural requirements, it is important to consider the *topology* of all nodes that are managed, the realization of different *provision models*, and most notably, the implementation of *fault-tolerance* even for the architecture-managing components. Some of these requirements are already covered by K8s. In terms of resources, for example, K8s provides basic scheduling using CPU and memory resources and horizontal scaling based on these metrics. Additionally, K8s offers built-in mechanisms building high availability clusters. As mentioned by [19], it is very convenient with K8s to add and remove nodes to a cloud-edge architecture during runtime. This is a typical practice in edge systems to dynamically add and remove nodes from and to the cluster [28]. We did not include this in our analysis in detail because all cloud-edge architecture implementations offer this implicitly by the usage of K8s. In addition, security was not further analyzed because the system-related communications are secured via HTTPS and provide a built-in system for authentication and authorization.[16] However, there are several shortcomings of K8s in orchestration activities. Firstly, in terms of resource-awareness, K8s is primarily built for the cloud and is not supposed for running on a heterogeneous structure [12]. A major problem is dealing with real-time resource demands and supplies that are not regarded during scheduling activities. Especially missing network-related measurements in the default scheduling behavior, as most significant issue, must be solved. Regarding architecture-related issues, the missing topology awareness was frequently mentioned, limiting the orchestration possibilities. This leads to the issue that K8s will be unable to perform other provision models like edge-only deployments. For example, standalone K8s clusters (i.e., single clusters for cloud, edge, and IoT) might offer basic horizontal scaling with acceptable performance. Inherently, K8s offers a centralized way of scheduling and managing the cluster. As outlined, this violates the decentralized notion of edge computing technologies. However, this issue can be relaxed because there are various non-K8s-based solutions for cloud-edge architectures, as shown by the works of [38,46]. Especially for task offloading, centralized management is used very frequently, as analyzed by [43].

In Sect. 4.1 we analyzed the state-of-the-art of K8s-based cloud-edge orchestration. We also outlined the general limitations of K8s for edge orchestra-

[16] https://kubernetes.io/docs/concepts/security/controlling-access/.

tions (Sect. 4.2) as well as potential solutions (Sect. 4.3). This contributes to RQ2 that aimed to investigate the benefits and drawbacks of the solutions that used K8s as orchestration platform for cloud-edge architectures. We argue that many solutions provided reasonable and meaningful proposals getting K8s ready for edge computing. The most notable benefits regarding resource-awareness include adding custom schedulers that make dynamic usage of other resources than CPU or memory and realizing network-aware deployments, especially for the scheduling process. Also, architecture-related issues were approached, like implementing topology-aware deployments. However, there is still a considerable lack in implementing different provision models, fault-tolerance of the cluster architecture, and placement of container registries as critical components for cloud-edge orchestrations.

The qualitative analysis of the K8s-based proposals enabled us to identify shortcomings (Sect. 5), according to our work from the previous chapters. As already indicated, the solutions that were part of our study showed several drawbacks. It is crucial solving them in order to run cloud-edge deployments in production. From our perspective, we believe that most of the challenges are solvable with a foreseeable amount of effort (RQ3). There are partial solutions for almost all problems, which can be combined to realize a unified solution.

We believe that all resource-related issues can be solved. Even if several cloud-edge architectures used only the edge layer for deployment, K8s clusters can be distributed over the different layers or deployed by the usage of cluster federation[17]. As already outlined, many components of K8s can be extended or replaced, like the scheduler component to be aware of real-time resources and network measurements. However, implementing native schedulers for the edge is considered a complex process [12]. This might be the reason why there are no native implementations. In addition, K8s allows the replacement of the metrics server to enrich the type of measurements.

We conclude that essential architecture-related requirements, like network topology, offering different types of provision models apart from edge-only deployments, high availability concepts, and the placement of container registries were not a major focus of the implementations. As already indicated, it might be desirable to include, for example, the IoT layer for containerized deployments as well. Issues like implementing fault-tolerance by replicating clusters, master nodes, or geographically distributed container registries, can be provided quickly and without much effort. However, this requires relaxing essential criteria like the decentralized architecture of edge computing in favor of a unified orchestration platform. As discussed above, this might be an acceptable trade-off.

6.2 Limitations

To the best of our knowledge, we aligned the set of evaluation criteria to the most critical general requirements of edge computing and the derived shortcomings of K8s. In consequence, the evaluation catalog might not cover the complete

[17] https://github.com/kubernetes-sigs/kubefed.

set of requirements and represents a simplification that should reveal the state-of-the-art in orchestration activities with K8s. Furthermore, it is still not clear if the selected solutions are comparable. They implement centralized, decentralized, and mixed architectures and follow different approaches and objectives, like implementing a cloud-edge topology, minimizing resource usage, optimizing container allocation, or reducing latency in deployments. However, we state that the type of architecture or objective should not constrain the mission-critical challenges in edge computing. In Sect. 2.1, we described different provision models in edge computing that different edge technologies can implement. Nevertheless, it is questionable if all provision models must be supported by one solution because edge-only deployments seem to be the most frequent deployment, according to our survey. More sophisticated solutions increase the complexity and might influence the performance of large-scale deployments.

6.3 Kubernetes as Unified Cloud-Edge Platform

Lastly, it must be evaluated if K8s should be further considered as a cross-layer platform. As already explained in Sect. 4.3, there are many modifications and extensions needed to fulfill the basic requirements of cloud-edge orchestrations. In addition, the platform is rather designed for cloud computing than for the orchestration of complex multi-layer architectures [22]. However, in our perspective, K8s should still be in focus for cloud-edge orchestrations.

There are mainly three reasons for this, which are as follows: K8s is widely spread and can be seen as the state-of-the-art orchestration platform for containerized workloads. Since container virtualization leverages edge computing, a well-established, maintained, tested, and reliable framework is a crucial success factor fostering the penetration of edge computing, enabling better and more reliable real-time services. There are a lot of cloud providers and other platform as a service companies that offer K8s without an advanced configuration to set up hybrid architectures, for example Google Cloud Platform[18] or Jelastic[19]. Further, because edge computing is a relatively new paradigm, developers can benefit from already available expertise. This reduces the initial hurdle deploying those architectures since developers are already familiar with the technology. Secondly, security and high availability is even a significant aspect in implementing cloud-edge architectures. All interactions between the controlling instances, nodes, and containers that collect metrics must be secured. Otherwise, deployment to production is irresponsible. K8s offers built-in mechanisms for authentication, authorization, and user management. Next, support for high availability deployments, not only restricted to the actual deployment but also to the system-related components that are required to operate the architecture, is worth mentioning at this point. In this regard, there are many other custom implementations for edge computing out there, mostly prototypes, without proper security management or option for high availability. As the third and last

[18] https://cloud.google.com/.
[19] https://jelastic.com/.

reason to continue working on K8s for cloud-edge architectures is the extensibility of a well-designed ecosystem. Replacing fundamental components, like the scheduler component, adding labels to realize a network topology, and revising the considered metrics make K8s an eligible candidate for a unified platform.

Indeed, the set of possibilities is a major challenge and needs some standardization. As mentioned by [39], there is a strong need for standardization in orchestration-related activities. Architectural and implementation-related proposals with standardized components can be helpful running K8s in different setups for orchestration of complex and large-scale architectures. Specifically, this refers to providing custom native schedulers for edge computing, which can be easily pulled from a public repository and run alongside or in place of the native scheduler. Potential algorithms for orchestrating the cloud or edge layer should be taken from the literature and provided for K8s to enable widespread usage. This could also foster the use of established edge orchestration strategies, algorithms, or policies. Additional components, like custom metrics server, must also be standardized to expand the limited set of built-in supported metrics like the metrics server.

7 Conclusion and Future Work

Edge computing complements the cloud by bringing additional resources closer to end-users to achieve reliable and low-latency quality of service for real-time applications. Providing and distributing applications is mainly performed with container technology, where workloads are executed on a container runtime, usually pulled from a central registry. Cloud-edge orchestration activities comprise the workload distribution on a set of resource-limited nodes. Over time, several approaches emerged that took K8s, a container orchestration platform built for the cloud, for cloud-edge orchestration. Since many authors claimed that K8s is not ready for this kind of orchestration, they proposed several improvements.

In this paper, we evaluated those architectural proposals that tackle several limitations of K8s as a cloud-edge orchestration platform. We have done this based on essential requirements of edge computing, edge orchestration, and capabilities of native K8s that were obtained from a literature survey.

As a result, we identified several benefits and drawbacks of the established architectures. Major issues like real-time resource utilization, network awareness, and network topology were solved quite well. Other aspects, however, were neglected. These aspects comprise especially implementing different provision models, like scaling and offloading between the cloud and edge layer. Furthermore, fault-tolerant cluster architectures for managing cloud-edge architectures are in the early stages. The missing consideration of appropriate container registry placement strategies was also identified as a major challenge.

In addition, we assessed if the shortcomings of K8s for cloud-edge orchestrations could be resolved with an appropriate amount of effort. We conclude that based on the already available partial solutions, K8s-based cloud-edge orchestration should still be in focus for further improvement and research.

For our future work, we still want to consider K8s as a unified orchestration platform, even for the edge. We do believe that a high degree in standardization of the extensions that are necessary to serve cloud-edge architectures can brace the position of K8s as a standard orchestration system. Our following contributions will specify a set of architectural blueprints that help set up production environments. Further, we want to guide how distinguished generic cloud-edge orchestration strategies, algorithms, and policies can be implemented by the usage of natively implemented K8s schedulers. The public availability of those schedulers provided by a public registry can foster the usage and active participation of K8s in edge computing. To examine the feasibility of the proposed architectural blueprints, we plan to provide reference implementations that are evaluated and tested at scale to achieve evidence that K8s can run large and sophisticated cloud-edge architectures in production.

References

1. Aazam, M., Zeadally, S., Harras, K.A.: Offloading in fog computing for IoT: review, enabling technologies, and research opportunities. Futur. Gener. Comput. Syst. **87**, 278–289 (2018)
2. Ahmed, E., Rehmani, M.H.: Mobile edge computing: opportunities, solutions, and challenges. Futur. Gener. Comput. Syst. **70**, 59–63 (2017)
3. Al-Tarawneh, M.A.B.: Mobility-aware container migration in cloudlet-enabled IoT systems using integrated muticriteria decision making. Int. J. Adv. Comput. Sci. Appl. **11**(9), 694–701 (2020)
4. Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M., Steinder, M.: Performance evaluation of microservices architectures using containers (2015)
5. Babou, C.S.M., Fall, D., Kashihara, S., Niang, I., Kadobayashi, Y.: Home Edge Computing (HEC): design of a new edge computing technology for achieving ultra-low latency. In: Liu, S., Tekinerdogan, B., Aoyama, M., Zhang, L.-J. (eds.) EDGE 2018. LNCS, vol. 10973, pp. 3–17. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94340-4_1
6. Bagchi, S., Siddiqui, M.B., Wood, P., Zhang, H.: Dependability in edge computing. Commun. ACM **63**(1), 58–66 (2019)
7. Barika, M., Garg, S., Zomaya, A.Y., Wang, L., Moorsel, A.V., Ranjan, R.: Orchestrating big data analysis workflows in the cloud. ACM Comput. Surv. **52**(5), 1–41 (2019)
8. Bilal, K., Khalid, O., Erbad, A., Khan, S.U.: Potentials, trends, and prospects in edge technologies: fog, cloudlet, mobile edge, and micro data centers. Comput. Netw. **130**, 94–120 (2018)
9. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing - MCC 2012. ACM Press (2012)
10. Casalicchio, E.: Autonomic orchestration of containers: problem definition and research challenges. In: Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools. ACM (2017)
11. Casquero, O., Armentia, A., Sarachaga, I., Perez, F., Orive, D., Marcos, M.: Distributed scheduling in Kubernetes based on MAS for fog-in-the-loop applications. In: 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE (2019)

12. Eidenbenz, R., Pignolet, Y.A., Ryser, A.: Latency-aware industrial fog application orchestration with Kubernetes. In: 2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC). IEEE (2020)
13. ETSI: Mobile-edge computing - introductory technical white paper (2014). https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V118-09-14.pdf
14. Goethals, T., DeTurck, F., Volckaert, B.: Extending Kubernetes clusters to low-resource edge devices using virtual Kubelets. IEEE Trans. Cloud Comput. (2020)
15. Goethals, T., Volckaert, B., de Turck, F.: Adaptive fog service placement for real-time topology changes in Kubernetes clusters. In: Proceedings of the 10th International Conference on Cloud Computing and Services Science. SCITEPRESS - Science and Technology Publications (2020)
16. Haja, D., Szalay, M., Sonkoly, B., Pongracz, G., Toka, L.: Sharpening Kubernetes for the edge. In: Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos on - SIGCOMM Posters and Demos 2019. ACM Press (2019)
17. Han, Y., Shen, S., Wang, X., Wang, S., Leung, V.C.M.: Tailored learning-based scheduling for Kubernetes-oriented edge-cloud system (2021)
18. Hong, C.H., Varghese, B.: Resource management in fog/edge computing. ACM Comput. Serv. **52**(5), 1–37 (2019)
19. Hoque, S., Brito, M.S.D., Willner, A., Keil, O., Magedanz, T.: Towards container orchestration in fog computing infrastructures. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC). IEEE (2017)
20. 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)
21. Kaur, K., Garg, S., Kaddoum, G., Ahmed, S.H., Atiquzzaman, M.: KEIDS: Kubernetes-based energy and interference driven scheduler for industrial IoT in edge-cloud ecosystem. IEEE Internet Things J. **7**(5), 4228–4237 (2020)
22. Kayal, P.: Kubernetes in fog computing: feasibility demonstration, limitations and improvement scope: invited paper. In: 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), pp. 1–6. IEEE (2020)
23. Klas, G.I.: Fog computing and mobile edge cloud gain momentum. Open Fog Consortium-ETSI MEC-Cloudlets (2015)
24. Morabito, R.: Virtualization on internet of things edge devices with container technologies: a performance evaluation. IEEE Access **5**, 8835–8850 (2017)
25. Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R.H., Morrow, M.J., Polakos, P.A.: A comprehensive survey on fog computing: state-of-the-art and research challenges. IEEE Commun. Surv. Tutorials **20**(1), 416–464 (2018)
26. Naha, R.K., et al.: Fog computing: survey of trends, architectures, requirements, and research directions. IEEE Access **6**, 47980–48009 (2018)
27. Ogbuachi, M.C., Reale, A., Suskovics, P., Kovács, B.: Context-aware Kubernetes scheduler for edge-native applications on 5G. J. Commun. Softw. Syst. **16**(1), 85–94 (2020)
28. Pahl, C., Ioini, N.E., Helmer, S., Lee, B.: An architecture pattern for trusted orchestration in IoT edge clouds. In: 2018 Third International Conference on Fog and Mobile Edge Computing (FMEC). IEEE (2018)
29. Pahl, C., Lee, B.: Containers and clusters for edge cloud architectures - a technology review. In: 2015 3rd International Conference on Future Internet of Things and Cloud. IEEE (2015)
30. Premsankar, G., Francesco, M.D., Taleb, T.: Edge computing for the internet of things: a case study. IEEE Internet Things J. **5**(2), 1275–1284 (2018)

31. Qiu, Y., Lung, C.H., Ajila, S., Srivastava, P.: LXC container migration in cloudlets under multipath TCP. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC). IEEE (2017)
32. Qiu, Y., Lung, C.H., Ajila, S., Srivastava, P.: Experimental evaluation of LXC container migration for cloudlets using multipath TCP. Comput. Netw. **164**, 106900 (2019)
33. Santos, J., Wauters, T., Volckaert, B., Turck, F.D.: Resource provisioning in fog computing: from theory to practice †. Sensors **19**(10), 2238 (2019)
34. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput. **8**(4), 14–23 (2009)
35. Satyanarayanan, M.: Edge computing. Computer **50**(10), 36–38 (2017)
36. da Silva, D.M.A., Asaamoning, G., Orrillo, H., Sofia, R.C., Mendes, P.M.: An analysis of fog computing data placement algorithms. In: Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services. ACM (2019)
37. Svorobej, S., Bendechache, M., Griesinger, F., Domaschka, J.: Orchestration from the cloud to the edge. In: Lynn, T., Mooney, J.G., Lee, B., Endo, P.T. (eds.) The Cloud-to-Thing Continuum. PSDBET, pp. 61–77. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-41110-7_4
38. Taherizadeh, S., Stankovski, V., Grobelnik, M.: A capillary computing architecture for dynamic internet of things: orchestration of microservices from edge devices to fog and cloud providers. Sensors **18**(9), 2938 (2018)
39. Vaquero, L.M., Cuadrado, F., Elkhatib, Y., Bernal-Bernabe, J., Srirama, S.N., Zhani, M.F.: Research challenges in nextgen service orchestration. Futur. Gener. Comput. Syst. **90**, 20–38 (2019)
40. Vaquero, L.M., Rodero-Merino, L.: Finding your way in the fog: towards a comprehensive definition of fog computing. ACM SIGCOMM Comput. Commun. Rev. **44**(5), 27–32 (2014)
41. Varghese, B., Wang, N., Barbhuiya, S., Kilpatrick, P., Nikolopoulos, D.S.: Challenges and opportunities in edge computing (2016)
42. Velasquez, K., et al.: Fog orchestration for the internet of everything: state-of-the-art and research challenges. J. Internet Serv. Appl. **9**(1) (2018)
43. Wang, J., Pan, J., Esposito, F., Calyam, P., Yang, Z., Mohapatra, P.: Edge cloud offloading algorithms: issues, methods, and perspectives. ACM Comput. Serv. **52**(1), 1–23 (2019)
44. Wöbker, C., Seitz, A., Mueller, H., Bruegge, B.: Fogernetes: deployment and management of fog computing applications. In: NOMS 2018–2018 IEEE/IFIP Network Operations and Management Symposium. IEEE (2018)
45. Yousefpour, A., et al.: All one needs to know about fog computing and related edge computing paradigms: a complete survey. J. Syst. Archit. **98**, 289–330 (2019)
46. Yu, Z., Wang, J., Qi, Q., Liao, J., Xu, J.: Boundless application and resource based on container technology. In: Liu, S., Tekinerdogan, B., Aoyama, M., Zhang, L.-J. (eds.) EDGE 2018. LNCS, vol. 10973, pp. 34–48. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94340-4_3