EAI Endorsed Transactions
on Smart Cities

Research Article EAI.EU

Cloud-Edge Orchestration for Smart Cities: A Review of Kubernetes-based Orchestration Architectures

Sebastian Böhm* and Guido Wirtz

Distributed Systems Group, University of Bamberg, Bamberg, Germany

Abstract

Edge computing offers computational resources near data-generating devices to enable low-latency access. Especially for smart city contexts, edge computing becomes inevitable for providing real-time services, like air quality monitoring systems. Kubernetes, a popular container orchestration platform, is often used to efficiently manage containerized applications in smart cities. Although it misses essential requirements of edge computing, like network-related metrics for scheduling decisions, it is still considered. This paper analyzes custom cloud-edge architectures implemented with Kubernetes. Specifically, we analyze how essential requirements of edge orchestration in smart cities are solved. Also, shortcomings are identified in these architectures based on the fundamental requirements of edge orchestration. We conduct a literature review to obtain the general requirements of edge computing and edge orchestration for our analysis. We map these requirements to the capabilities of Kubernetes-based cloud-edge architectures to assess their level of achievement. Issues like using network-related metrics and the missing topology-awareness of networks are partially solved. However, requirements like real-time resource utilization, fault-tolerance, and the placement of container registries are in the early stages. We conclude that Kubernetes is an eligible candidate for cloud-edge orchestration. When the formerly mentioned issues are solved, Kubernetes can successfully contribute latency-critical, large-scale, and multi-tenant application deployments for smart cities.

Received on 14 March 2022; accepted on 23 May 2022; published on 25 May 2022

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

doi:10.4108/eetsc.v6i18.1197

1. Introduction

Over the years, Information and Communications Technologies (ICTs) have become inevitable and permeated more and more areas in everyday life. Besides the traditional usage in business-related contexts, ICTs are used in urban areas, which is also known as the concept of Smart City (SC). In SCs, various technologies sustainably enhance urban life [1]. Many fundamental areas in cities are equipped with sensors to gather data from community services, like transportation, power plants, information systems, and crime detection [2]. The goal to build a SC is accompanied by an increasing number of sensors and traffic volume. This development also affects applications that run in SC-related applications. Requirements like the provision of computational resources on-demand, low-latency in multiple regions, and flexible moving of services in a city must be tackled [3]. Comprehensive data collection is necessary to realize a SC. Often Internet of Things (IoT) devices are used [4–7]. An IoT device is a resource-constrained member of a larger network that consists of similar devices. Those devices, mainly sensors, collaborate on a common goal and need sophisticated management [8]. Since these sensors are often placed in different areas, a large amount of data needs to be collected, processed, and analyzed to decide accordingly to the long-term goal. Furthermore, real-time capabilities and fault-tolerance are inevitable for these services to be stick to the goals of SCs [9, 10]. These requirements set new challenges for designing software architectures.

*Corresponding author. Email: sebastian.boehm@uni-bamberg.de

So far, cloud computing, which offers a large pool of centralized resources, is the mean of choice for collecting and processing data from IoT devices [11]. In recent years, many SC architectures emerged that facilitate IoT devices and contributed to various challenges that are within the scope of SCs. Mostly, these architectures tackle the challenges of SCs by digitalizing more and more public services [4, 12–14].

Since the number of IoT devices is continuously increasing, the cloud may not be able to serve a large number of requests with a particular latency [15]. Furthermore, due to the highly distributed nature of placing IoT devices in different areas in an urban region, connectivity and bandwidth issues may arise. These issues make it hard to use the cloud without unfavorable implications [16, 17]. In the context of SCs, many digitalized services (like city monitoring and public transport) require real-time capabilities to operate efficiently [7, 14, 18]. Hence, the average response time must be reduced. To overcome this situation, edge computing comes into play as an additional layer to the cloud, offering computational capacity near data-generating devices. This realizes low latency and better network reliability for service-requesting devices [19, 20]. Consequently, edge computing offers the ability to selectively upload data to the cloud, contributing to enhanced privacy. Providing computation capacities for SCs, using the edge computing paradigm, can be a complex task. The assignment of workloads must be done according to the available infrastructure, which often consists of heterogeneous and resource-constrained devices [21]. In addition, a lot of different services need to be deployed in a SC, depending on the public services, which must be supported [10]. At this, assigning computational workloads for data processing is mostly done with containers, a lightweight alternative to Virtual Machines (VMs). Those containers are managed by a container engine and contain all necessary dependencies. This nominates them as an ideal candidate for edge computing [22–24].

Even if the complexity of deployments is simplified by container technology, new challenges, like latency requirements, arise. In consequence, an efficient management of container instances is required by taking resource demands and capacities into consideration. This is also known as edge orchestration [25]. Large-scale container orchestration is a rather complex task that requires a sophisticated management. For this, various container orchestration tools emerged, like the popular Kubernetes (K8s)[1]. It offers high availability, scalability, and fault-tolerant management of a large number of containers. In the context of SCs, a lot of solutions emerged over time that are using K8s as orchestration system [1, 26–29]. So far, no comprehensive survey of already existing K8s-based architectures has been done to verify whether K8s is still worth considering for edge orchestration. This targets deployments for SCs as well, where container technology is becoming critical to provide low-latency deployments in line with resource demand and supply.

Therefore, this paper aims to provide a detailed overview of existing K8s-based implementations for edge orchestration, partly for a SC context as well. For this, we analyze essential requirements of cloud-edge orchestration, also with a focus on SCs. In an additional step, we use the obtained criteria for an evaluation of existing K8s-based solutions. Finally, the results of the evaluation are used to assess if K8s is an eligible solution for cloud-edge orchestration, even in a SC context. This leads to the following research questions:

- **RQ1**: What are the most critical requirements for cloud-edge orchestration and are covered by K8s?
- **RQ2**: What are the benefits and drawbacks of already established cloud-edge architectures that are based on K8s?
- **RQ3**: Are the potentially identified drawbacks of the investigated solutions in K8s solvable with a realizable amount of effort?
- **RQ4**: Is K8s an eligible candidate for providing demand- and supply-aware deployments, also for a SC context?

To answer our research questions, we perform a literature review. At this, we extract fundamental characteristics of cloud-edge orchestration with a focus on SCs. This contributes to RQ1. To answer RQ2, we review already existing cloud-edge architectures that are using K8s for advanced orchestration, also in the area of SCs. We map the obtained characteristics to the existing solutions to identify the strengths and weaknesses of the implementations to answer RQ3. Lastly, we answer, based on our prior results, if K8s can still be considered as an eligible candidate for cloud-edge orchestration in the context of SCs (RQ4).

This paper starts with a conceptual overview of SC, edge computing, edge orchestration, and K8s (Section 2). Section 3 discusses related works evaluating cloud-edge orchestration solutions. In Section 4, K8s, K8s-based orchestration architectures, limitations, and potential solutions are further analyzed. The investigated shortcomings of the solutions are covered in Section 5. We discuss our work in Section 6 with a critical assessment of our findings and the threats to validity. Finally, we conclude our work in Section 7 with a short summary and the plan for our future work.

[1] Kubernetes

2. Concepts

This section gives a short overview of the core concepts required to survey cloud-edge orchestration architectures. First, we introduce SCs to outline the need for complex orchestrations. We describe the principles of edge computing in a second step. Finally, we conclude this section with a short introduction on the orchestration of cloud-edge architectures.

2.1. Smart City

In the following section, the concept of SC is explained in detail. Besides the definition of SC, fundamental characteristics and arising challenges are discussed.

Definition. In recent years, SCs have become very popular. By the usage of ICTs, living in cities should be made more comfortable. Although there is no standard definition and understanding of the term SC, there are a couple of definitions with different perspectives on the topic, as researched by [30]. All of them have in common to improve the quality of life of citizens by usage of ICT in basic needs, as outlined in Section 1. Many authors, like [30] and [31], come to this inference.

According to the emerging popularity of SCs, a lot of different architectures emerged that should simplify establishing smart services. There are also a few surveys that investigate the emerged architectural proposals and work out the granular differences [32–34]. Quite common is the three-tier architecture, where multiple layers are used to define and classify SC architectures (Figure 1). The architecture, taken from [5], as representative architecture for a SC mainly consists of three tiers, namely the urban environment, the communication layer with further separation into different steps, and finally, the service layer. The urban environment has many different sensors that are continuously collecting data. For example, this targets technical installations for smart traffic control systems, air quality monitoring systems, and video surveillance [35]. Also, the core infrastructure is covered in this tier to connect all the devices to data-receiving endpoints, e.g., by using ethernet, wireless, or mobile broadband connections. The first step, which is performed by the communication layer, is data collection from all the devices placed in a SC environment (①). After collection, the data processing is triggered (②). This step includes transforming heterogeneous data into homogeneous data for further processing. The data is also enriched with additional information, like metadata, to create semantic relationships between data entities. This step is essential for the data integration, which takes place as the third step in the communication layer (③). In data integration, the processed data is further analyzed by an inference engine to draw conclusions from the data. This might be a complex process that requires sophisticated methods and the expertise of domain experts.

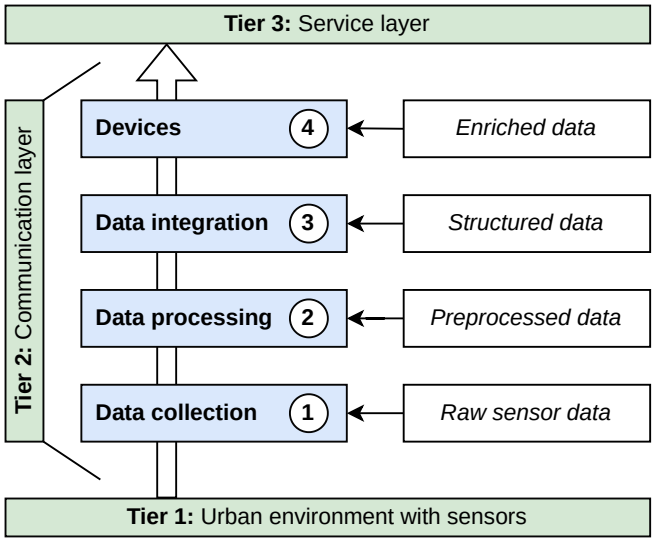

Figure 1. General Smart City Architecture (three-tier) [5]

However, this step is inevitable to inform stakeholders, especially citizens, about important events. That is covered in step ④. Finally, SC architectures allow for several customized services that have access to the formerly presented data processing pipeline. For example, in the service layer, the government or third-party software developers can create web applications, dashboards, or other services that contribute to the aims of SCs. Typical use cases are also presented (Section 2.4).

Characteristics. Over the years, many attempts were performed to conceptualize the term and determine aspects of SCs in a broader sense. One frequently cited characterization of SC was done by [18]. They describe a SC in six aspects, mainly based on the findings of [36] with a critical assessment. This characterization comprises a holistic view by including the social, economic, and technical perspectives and allows a well-balanced overview of how a SC should be designed [18]. These characteristics can be summarized and reorganized in different application domains for a SC that enables particular benefits for the urban development and the quality of life. In [34], a popular classification is presented that helps to organize the contributions and benefits a SC can accomplish. The authors outline that SCs can lead to a (i) more efficient government, for example, by monitoring and managing public safety. Furthermore, (ii) real-time information in daily life leads to happier citizens [4]. (iii) More efficient logistics and supply chain platforms can make businesses more prosperous. (iv) Finally, also due to climate change, a SC can foster environmental sustainability by reducing and preventing pollution using real-time systems. In [37], a Smart IoT Based Building and Town Disaster

Management System for SCs was designed, which allows efficient monitoring and management for public safety. Another solution [38] deployed a system of interconnected modules into a SC to monitor events. The solution aims to support disaster management and did not rely on any additional infrastructure. Systems like this contribute to a more efficient government. In regards to real-time analytics, [39] mention that smart traffic control systems have a positive influence on avoiding traffic jams and mitigating traffic congestion in the city. For this, the city was equipped with video surveillance technology. This improves the quality of living. As a further example, SC-related activities can foster new business models [40]. In [41], comprehensive data collection for commercial purposes was suggested. According to the density of smartphones in a particular area, a new price structure for billboards has been established. Furthermore, it is possible to organize parking spaces in a more efficient way [42]. Optimizing waste disposal is also mentioned as an area for further optimization [43]. Tackling climate change, various attempts have already been made to monitor pollution in SCs and launch countermeasures, e.g., by prohibiting particular car classes from driving into the inner-city [44, 45]. This helps improve the air quality and finally the healthiness of citizens in large cities. However, several challenges are accompanied by establishing SC architectures. As already indicated, the concept of SC is strongly associated with the IoT paradigm. Those devices undertake essential tasks to realize the characteristics a SC should have. Improving citizens' well-being needs a sophisticated provision of high-quality smart services [2, 31]. Popular application areas that contribute to the goals of SCs also imply challenges, which are discussed in the following.

Challenges. Fundamental challenges occurring by establishing SC architectures that handle a large set of workloads and requests are two-fold. First, general organizational issues come up because there are a lot of stakeholders and distributed responsibilities that result in complex decision-making. Multiple industry partners, institutes, and public services must collaborate to achieve an efficient solution. This collaboration also comprises the appropriate sharing and management of network infrastructure, computational resources, and eventually data centers and their derivatives in edge computing [1]. Secondly, besides sharing and agreeing on a common data platform, further challenges must be considered as soon the organizational aspects and core infrastructure is set up. A major concern is providing an efficient allocation of applications for low-latency use cases that are quite common in SC contexts. In addition, because critical areas like smart traffic control systems must be served, an appropriate fault-tolerance is inevitable [9]. Since the number of requesting devices is still increasing and needs to be managed, redundancy, reliability, and real-time capabilities are inevitable for an appropriate orchestration of large-scale deployments [10].

2.2. Principles of Edge Computing

Edge computing adds an additional layer to the cloud by placing computational resources close to data-generating devices, like sensors, actuators, or other entities. This new placement strategy aims to reduce latency and improve bandwidth capabilities [46, 47]. In contrast to the cloud, where all data is transmitted to and stored in a centralized way, edge computing provides additional resources to take the load from the cloud [48]. Especially low-latency application fields, like real-time analytics or video surveillance, can benefit from the increased bandwidth and reduced latency realized by edge computing [49, 50]. Also, the geographically distributed nature of large IoT networks is prone to unstable network connections [51]. The core requirement of edge computing, minimizing the latency and increasing bandwidth for real-time services, might be violated by long distances between client and servers. The length of the physical distance between clients and servers has a coherence with latency [19]. Edge computing as general technology has different types, also called edge technologies [52]. Common types are Mobile Edge Computing (MEC), Cloudlet, Micro Data Center (mDC), and Fog. All of these offer different provision models (Figure 2).

MEC was introduced by Nokia and placed computational resources (e.g., computing, network, and storage) mainly next to mobile Radio Access Network (RAN) stations. This type of edge computing aims to provide low-latency network access to low-power devices, often to process time-critical tasks, like real-time analytics. The placement next to mobile RAN stations enables fast and dynamic provisioning of applications, which offer real-time services. Accordingly, the provision of applications near the data-generating devices reduces the traffic sent to the cloud and reduces network congestion. There is no common understanding if MEC is a substitute for the cloud. It is unclear whether data must or must not be forwarded to the cloud in any case [52, 53].

Cloudlets form virtualized clusters with a set of decentralized devices for running low-latency applications. They are self-managing, fast and easy to deploy by local administrators and aim to provide computational resources close to data-generating devices. Workloads and applications are supposed to be transmitted as VM overlays. The overlays are executed on top of a base image that is already available on the target device. It is necessary to shift these VM overlays rapidly, for example, if the service-requesting devices change their position continuously, for example, in

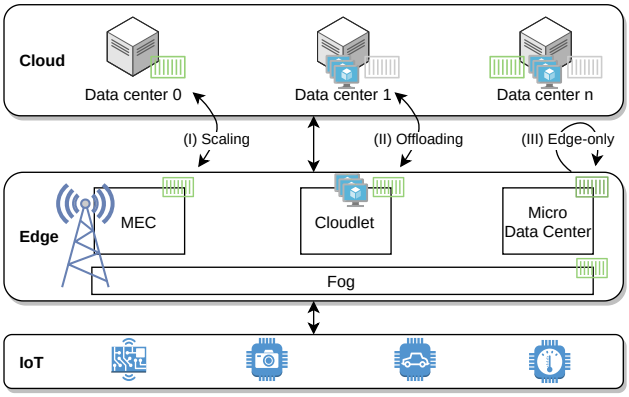

Figure 2. Cloud-Edge Architecture with Provision Models and Edge Technologies [58]

smart traffic control systems. Therefore, it is inevitable that cloudlets have a reliable network connection with a high bandwidth available [54]. Besides using VM overlays, [55] introduced lightweight containers for shifting workloads among devices in cloudlets. In addition, Linux containers were used [56, 57] to realize performance enhancements with a lightweight alternative.

Similar to cloudlets, mDCs want to reduce response times by collaborating with the cloud as an additional layer. mDCs support multi-tenancy and need, therefore, a strong hardware- and software-based protection against unauthorized access. They run in an isolated and secured unit in terms of physical and virtual access. For data exchange with the cloud, they usually have a reliable, fast, and durable connection.[2]

Fog computing, often also called edge computing [48, 51, 59], follows similar principles. Although there is a high similarity between fog computing and the formerly presented edge technologies, fog computing can be interpreted as one step closer to data-generating devices and is explicitly designed in a decentralized way [49]. In fog computing, large-scale networks of heterogeneous devices cooperate and communicate to realize low latency for the lower layers [11, 60, 61]. It is still an unanswered question if there is a substantial discrimination between edge and fog computing. Since edge and fog computing share common goals for many devices and act as an additional layer to the cloud, one can assume both terms can be treated equally. Many authors [47, 48, 51, 59, 60] follow this interpretation and do not introduce an explicit differentiation.

Figure 2 shows different ways to deploy applications with low-latency requirements. Utilizing the full architecture of cloud-edge environments, different techniques on how to deploy applications have emerged over time. The most common, so-called provision models, are (I) scaling, (II) offloading, and (III) edge-only deployments. For the last provision model, it is noteworthy that the edge is not supposed to replace the cloud side entirely [52, 53]. In the following, the different provision models are explained in detail:

- (I) Scaling: This type of provision model is also known under distributed offloading. Applications are running simultaneously in the cloud and edge, collaborating to achieve better performance. Both layers can scale out if a particular application runs out of resources (e.g., storage or latency) [62].
- (II) Offloading: The most frequent approach is offloading from cloud to edge and the other way round. For example, entire applications or application stacks are moved from the cloud to the edge layer, e.g., to meet a particular latency requirement. In addition, applications may be moved to the cloud if load decreases or the latency requirements get relaxed [63].
- (III) Edge-only: In this deployment type, applications are only placed and moved on the edge layer to fulfill the needed latency. Therefore, offloading is not required for this technique. However, many solutions use offloading techniques nonetheless to perform edge placement strategies [64].

2.3. Orchestration of Cloud-Edge Architectures

Using cloud-edge architectures for running a large set of different services comes along with new complexities. These complexities arise because of the additional edge layer, different edge technologies, and provision models. Cloud-edge orchestration is responsible for assigning workloads to the cloud, edge, and IoT layer based on a particular set of objectives (Section 2.2).

The most important aspect that must be covered is an efficient placement of applications dependent on the origin of potential requests. Also, the required real-time latency and bandwidth of devices must be considered to achieve the claimed application response times. For this, complex decision and orchestration models are required that consider demand and supply of resources like CPU, memory, disk, and network utilization [48]. Fault-tolerance and resilience is a further core requirement of cloud-edge architectures [65]. As already mentioned in the former section, cloud-edge systems are used in critical areas, like smart traffic control systems. Outages because of broken nodes in the architecture should not happen. The complexity of managing those architectures is further intensified due to the decentralized, distributed, and large-scale nature of the edge layer. In addition, a heterogeneous

[2] Microsoft researcher: Why Micro Datacenters really matter to mobile's future

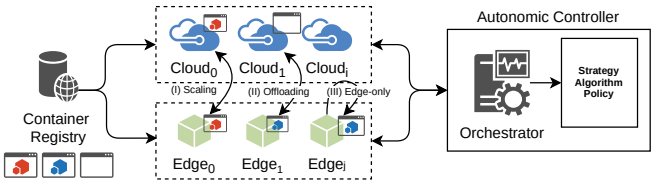

Figure 3. Generic Orchestration Architecture, based on [67]

set of devices, often low-power devices, needs to be managed appropriately [59, 63]. Dynamically scaling down and up the number and types of nodes in cloud-edge architectures must be supported to realize large-scale deployments in the right way [66]. Also noteworthy are security considerations like supporting multi-tenancy for edge technologies like cloudlets and mDCs, which are offering a shared model (Section 2.2). Security mechanisms like authentication and authorization are strongly required to operate cloud-edge systems [65], especially if they are publicly accessible [59].

As already outlined in the former sections, the provision models present several challenges that must be met. Workloads should be fast and easy to deploy and moved across the layers in the architecture. Meanwhile, container technology is the de facto standard running workloads in cloud-edge architectures. Containers follow the principle of lightweight virtualization and contain the application, libraries, and the runtime environment. They are executed in an isolated way by a so-called container engine which restricts the amount and type of resources a particular container can use (i.e., CPU, memory, and storage). The container engine itself runs on an operating system and shares the kernel with the container instances. Container technology has been widely accepted for this area because containers are small in size, have a fast startup time compared to traditional VMs [24], and can be easily ported to other physical nodes [22, 23].

The eligibility of container technology for orchestration activities requires an appropriate architecture that is able to realize the provision models. In Figure 3, a general and generic architecture based on [67] is shown. A container registry provides containerized applications. In case of a deployment instruction, corresponding nodes download the container image and execute it with the required configuration. A so-called autonomic controller creates these deployment instructions. This controller includes an orchestrator that assigns containers to nodes given a strategy, algorithm, or policy. This workflow can be applied to all provision models that are covered in this paper.

2.4. Edge Computing in Smart City Contexts

Edge computing and its related technologies have meanwhile a large application field in SC contexts. Over the years, many architectures emerged, like the rainbow architecture from [68] that aims to provide easy development of SC applications for large sensor networks. The authors followed an agent-based approach using fog computing to implement their architecture. Also, for the sake of air quality monitoring, [69] proposed an architectural design for SCs. In the work of [70], a managed, programmable, and virtualized edge platform was presented using container technology as an underlying technology. The authors tested different applications, like air quality monitoring, sound classification, and image recognition. Other solutions consider similar tasks and proposals using edge computing or related approaches for real-time traffic monitoring [71–73] or video streaming [74]. Further approaches focus on real-time use cases as well [7, 14, 18]. Especially the latter proposal aims to move workloads during runtime from edge nodes to other edge nodes. In regards to the already introduced edge technologies, there are architectures that either require or support cloud, edge, MEC, fog [75, 76], or finally cloudlets [77] as fundamental edge technology. Distributing and shifting of workloads and tasks in SCs was covered in [78, 79]. They formulated complex task allocation algorithms to further reduce the end users' latency. Lastly, edge computing and related orchestration activities also contribute to better privacy and security [77, 80].

Edge computing and in specific edge orchestration is an essential component providing additional computational capacities for SCs. There are a lot of different use cases. However, most of them have an explicit requirement that must be met, especially in terms of latency and bandwidth to operate normally. The highly distributed nature of SC architectures can be efficiently supported by edge computing and further by edge orchestration, where an advanced and flexible shifting of workloads is possible and different provision models are fully supported.

3. Related Work

We first analyzed publications that defined and investigated characteristics of cloud-edge computing. Then, we performed an evaluation based on a set of criteria. In [81], a literature survey on fog computing and other edge technologies was performed. They stated limitations, research directions, and potential solutions for fog orchestration. Their study mapped these limitations and research directions to potential solutions and summarized plenty of aspects for future work. [61] conducted a literature survey to investigate orchestration challenges for edge and fog computing.

Also, the authors provided a mapping between challenges and discussed a set of potential solutions. [82] investigated requirements like infrastructure-, platform-, and application-related criteria. Following the qualitative study design, these criteria were used to evaluate established fog architectures. The authors collected a comprehensive set of resource allocation and scheduling algorithms for orchestration. These criteria were used to evaluate a set of solutions that try to tackle this problem domain. In [59], a discussion of motivations, challenges, and opportunities in edge computing was provided. The authors assembled a set of criteria for orchestration-related activities.

In a second step, we consider works that investigate cloud-edge or edge orchestration architectures in specific. [83] set a focus on requirements for orchestration in regards to the management of nodes (e.g., joining/leaving the cluster or scheduling). In specific, they evaluated container orchestration tools like Mesos, K8s, and Docker Swarm. They defined requirements for orchestration systems, however, without any comparison of cloud-edge architectures implemented with the different container technologies. In the work of [65], the state-of-the-art of fog orchestration was further investigated with a focus on the core requirements an architecture must comply with. Accordingly, they evaluated well-established fog orchestration architectures. They inferred that most of the considered architectures could deal with the general requirements of fog computing.

Lastly, plenty of contributions are in close relation to our study that investigate only one particular aspect of cloud-edge orchestration in detail: Architectural and algorithmic challenges for resource provisioning and scheduling were further investigated in [84]. This literature review showed a comprehensive set of limitations that have been matched to potential solutions. In addition, starting points have been mentioned for further investigations. [62] analyzed offloading strategies that are an essential part of cloud-edge architectures. Similarly, [85] presented an overview of several offloading algorithms and evaluated them based on a set of criteria.

As shown in the former paragraphs, several solutions have already investigated challenges, research directions, and potential solutions for issues in cloud-edge orchestration activities. However, there is no overview of K8s-based cloud-edge and edge architectures in specific. Furthermore, there has not been a detailed analysis of the requirements of SCs for cloud-edge environments and orchestration yet. Hence, this contribution aims to provide a detailed analysis of modifications made to K8s to make it ready for the requirements of edge environments in SCs.

4. Kubernetes as Edge Orchestration Platform

This section covers K8s as a candidate for running cloud-edge and edge orchestration environments. First, we explain the general architecture and functionality of K8s. In a second step, we discuss K8s-based orchestration architectures that we obtained from a literature review by using the search term $\texttt{Kubernetes} \wedge (\texttt{edge} \vee \texttt{fog}) \wedge (\texttt{computing} \vee \texttt{orchestration})$. For the literature review, we used the following popular databases: IEEE Explore, SpringerLink, and ArXiv. We considered only those papers which identified shortcomings of K8s and offered solutions for the identified issues. Finally, we cover general limitations and potential solutions for K8s as an edge orchestration platform.

4.1. Kubernetes as Container Platform

The container platform K8s is used to execute containerized workloads on a set of nodes. Furthermore, it implicitly implements the generic orchestration architecture shown in Section 2.3. Figure 4 shows a minimal working cluster, consisting of one master node and one worker node. The master node, also called control plane, runs all essential system services for the cluster. To run containerized workloads, at least one worker node is needed. It is possible to assign workloads to the master node that, however, this is not recommended.[3] In general, containerized workloads are executed in pods, the smallest deployable unit in K8s that provides the execution environment for containers.

Managing the set of worker nodes and assigning containerized workloads to the worker nodes is done by the master node. At this, the managing services are also running as containers and can be distributed to multiple nodes, e.g., to achieve high availability. In specific, a K8s cluster includes the following components: All running nodes in the cluster are observed by the controller-manager (*c-m*). This component also keeps track of the current state to plan future actions and deployments. For example, the controller-manager can restart workloads on other nodes if the currently used node is failing or not operational anymore. The Kube Scheduler (KS) is in charge of assigning workloads to worker nodes (named *sched* in Figure 4). This assignment is realized based on a scoring model to find the most suitable node. For example, manually set constraints and available resources can be regarded. Cluster data, for example, currently running assignments, are stored in a strongly consistent and distributed key-value store, named *etcd*. In case of a redundant deployment, either with multiple master nodes or a unique etcd cluster, the data will be replicated across all instances. The *api* is exposed

[3] Kubernetes documentation - Nodes

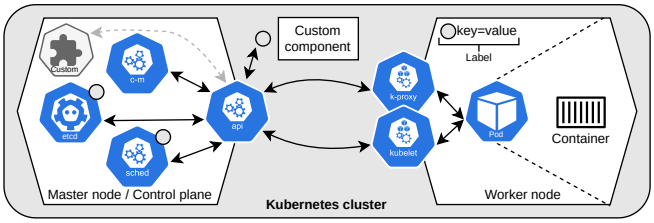

Figure 4. General Kubernetes Architecture [58]

as REST API and allows the interaction with the system components in the cluster. The Horizontal Pod Autoscaler (HPA), omitted in Figure 4, dynamically scales in and scales out the number of pods based on metrics like CPU and memory utilization.[4] The *kubelet* that is available on all worker nodes serves as a communication endpoint for the controller-manager. The major responsibility of this component is managing the lifecycle of nodes according to the commands received from the controller-manager. In addition, the current state of the node is transmitted to the controller-manager as well. The component *k-proxy* opens ports and forwards traffic according to the deployed workloads and the configuration.[5]

K8s is modularly built and allows replacing several components, as shown in Figure 4. For example, the scheduler component can be replaced completely. Furthermore, additional information in the form of labels can be attached to nodes. This might be helpful in enriching the data basis for complex scheduling processes in case the scheduler component has been replaced. External components, e.g., for advanced scheduling and scaling, can interact with the API or custom containerized components to achieve a particular outcome. These components might even be deployed on a K8s cluster as control plane components when the built-in scheduler is supposed to be replaced.[6]

4.2. Kubernetes-based Edge Orchestration Architectures

In our study, we classify K8s-based edge orchestration systems into three categories. The first category comprises open-source frameworks and solutions that aim to realize essential orchestration features in edge computing. Solutions that implement custom modifications and introduce new extensions to K8s are part of the second category. The last category reveals those solutions that tackle only the edge layer with a modified K8s.

Platform-based Solutions. The first category comprises frameworks like KubeEdge (KE)[7], Baetyl[8], OpenYurt (OY)[9], or ioFog[10]. These platforms deploy custom components as containerized workloads to an unmodified K8s cluster to implement their orchestration logic. Usually, these platforms provide only mechanisms for an easy setup process of cloud-edge architectures. Easy-to-use routines are included to roll out the required infrastructure components, like software-defined networks, message brokers, service and event bus endpoints, and management capabilities. Setting up and running cloud-edge architectures is simplified such that users can easily deploy devices and monitor their state. In addition, platforms like KE advertise resource optimization that enables the usage of low-power devices. However, the platforms lack dynamic workload allocation capabilities. Support for different provision models of edge computing is somewhat limited. For example, the platforms might not be able to perform dynamic placement decisions or placement changes based on the current utilization of a set of devices. Furthermore, provision models like offloading from cloud to edge and vice versa, as well as scaling out to the edge, are not supported in particular [86]. To overcome this limitation, several solutions considered in this study used the modifiability and extensibility of K8s (Section 4.1). They implemented custom components to meet the requirements of more sophisticated cloud-edge and edge architectures.

Custom Cloud-Edge Architectures. The second category comprises architectures that consider the cloud and the edge layer in collaboration. [86] introduced *KaiS*. This framework improves the long-term rate of request processing and system overhead by using the edge in cooperation with the cloud. The orchestration activities are planned centralized in the cloud to assign workloads to edge nodes. The actual workload is dispatched only to the edge. The proposed solution uses advanced learning techniques and orchestration fundamentals. As a result, *KaiS* can increase the throughput rate and reduce the scheduling costs compared to K8s. Minimizing interference and energy consumption of deployments was covered in [87] as a multi-objective optimization problem and is called *KEIDS*. All nodes in the cluster were equipped with custom containers to fulfill the requirements of co-allocation of dependent workloads on a single node. This solution aims to reduce the carbon footprint of the orchestration architecture. The authors implemented *KEIDS* and a modified version of *KEIDS*. They

[4] Kubernetes documentation - Horizontal Pod Autoscaling
[5] Kubernetes documentation - Kubernetes Components
[6] Kubernetes documentation - Extending Kubernetes

[7] KubeEdge
[8] Baetyl
[9] OpenYurt
[10] Eclipse ioFog

compared the results to a First Come First Serve algorithm and inferred that *KEIDS* led to different improvements in carbon footprint, performance, and energy minimization. *Swirly*, a solution proposed by [88], is a scheduler that creates a service topology to simplify orchestration activities. It runs in the cloud and supports large-scale deployments by considering the topology with a small number of containers. Also, real-time resource utilization is considered, like CPU, memory, and finally latency for the end users. For that, they equipped all nodes with custom container components. Based on a benchmarking study, they concluded that the solution is able to handle up to 300000 devices. [26] suggested a cloud-edge solution with a focus on location-aware scheduling for an air monitoring service. To achieve location-aware scheduling, they implemented custom schedulers and modified K8s. They compared their location- and network-aware scheduler to the default KS and approaches based on integer linear programming. The architecture with the modified scheduler reveals a remarkable latency reduction. In [89], K8s has been prepared for geographically distributed clusters. Similar to the approaches before, they attached custom components to each node and added a custom scheduler component to consider the network latency. This solution uses the cloud and the edge layer for running workloads and calculates latency-aware deployments based on periodic latency checks. An evaluation revealed that the architecture could adjust deployments according to real-time conditions. [90] designed an architecture with support for fault-tolerance, application isolation, data transport, and multi-cluster management. Fault-tolerant message broker clusters, deployed in cloud and edge, were used to realize the storage layer. Master nodes deployed in the cloud were replicated and operated in high availability mode. Therefore, a two-node failure keeps the proposed architecture operational. [91] propose *Fogernetes* that provides network-aware and resource-oriented deployments based on a labeling system. The labeling system adds key-value pairs to nodes and refers to them during deployment. For verification, they implemented an architecture for video streaming with devices placed in edge and cloud and defined the target nodes for deployment in corresponding deployment manifests. K8s used the former defined key-value pairs and realized a location-aware deployment based on labels and the definitions in the deployment manifests.

Custom Edge Architectures. The third category reveals a conceptual overview of solutions that tackle the edge layer specifically for orchestration purposes. In [92], a fog architecture with K8s was proposed to deploy multi-container applications on low-power devices. Several plugins extended the default KS to accomplish an efficient and location-aware deployment of containers. Multi-container applications are deployed on neighboring nodes. An evaluation yielded that the service quality was not compromised. [93] suggested a decoupled and native modification for K8s to implement location-aware, latency-aware, and fault-tolerant deployments. Deployments are calculated by the usage of an external component that passes these deployments to an unmodified K8s cluster. A robust integration into K8s was achieved by running an additional component directly on K8s that interacts with the API of the cluster. To validate the solution, the authors performed experiments on allocation performance and failover time. [66] developed an edge solution for industrial IoT that reduces the scheduling time. They applied the technique of single-step scheduling via a custom scheduler to pursue latency-aware deployments, an improved deployment time, and a lower temperature of all nodes in the cluster. In an evaluation, they showed that the scheduling time could be reduced. Also, they monitored latency, jitter, and packet loss during the scheduling process. An agent-based approach for orchestration of fog architectures was considered in [94]. They addressed inherent issues of K8s, like high-load on the master node in the scheduling process. To overcome this problem, they replaced the default KS that addresses only the fog layer. Selected scheduling tasks (e.g., node filtering, sorting, and scoring) were shifted to several nodes in the cluster. They evaluated their identified solution for a small number of replicas (< 10) and obtained that it needed less deployment time compared to the default approach.

4.3. General Limitations of Kubernetes for Edge Orchestration

This section discusses the limitations of K8s as an orchestration platform for cloud-edge orchestration. We categorized these limitations in resource-awareness and architectural shortcomings. A graphical representation of all shortcomings is depicted in Figure 5 and is explained in detail in the following section with the corresponding numbers 1 - 8.

Resource-Awareness. A major challenge in cloud-edge orchestration is dealing with real-time resource demands and supplies. The most important resources that must be tracked are CPU, memory, storage, and network utilization. Circumstances in cloud-edge environments can rapidly change. Appropriate actions, like offloading and scaling, in the environment must be triggered as soon as possible. K8s shows several limitations in this regard, as shown in Figure 5. Limitation 1 reveals that the default KS (*sched*) considers only CPU and

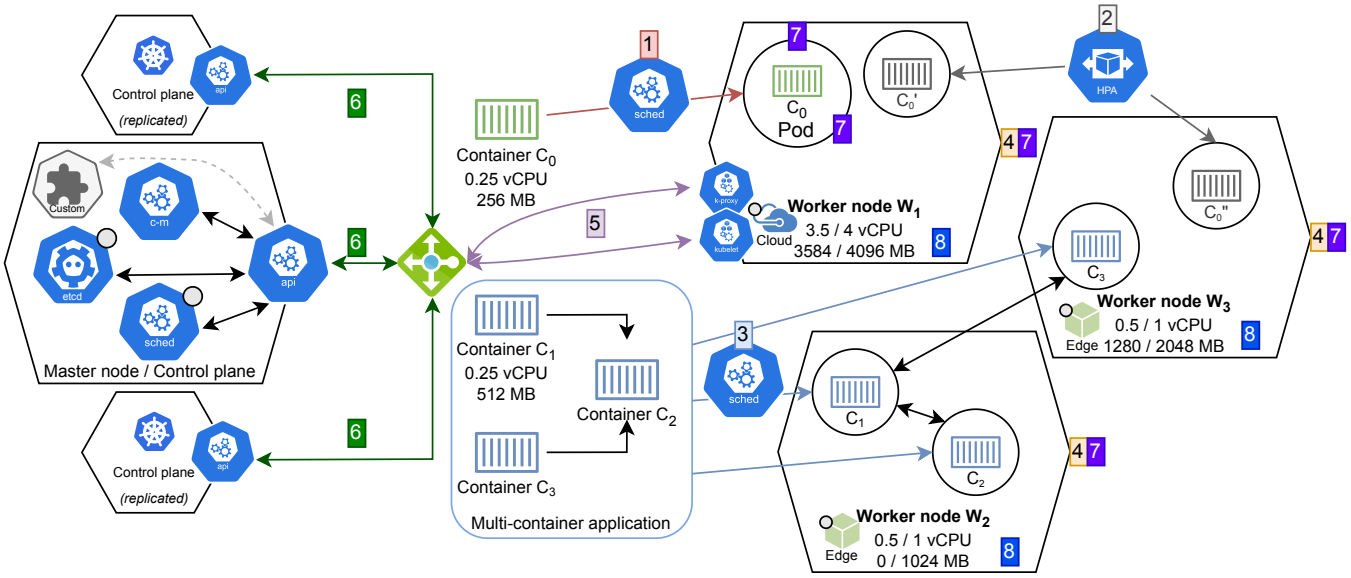

Figure 5. Selected Limitations of Kubernetes as Platform for Cloud-Edge Orchestration

memory utilization during the scheduling process. However, as outlined in the previous sections, network-related metrics [21, 26, 66, 87, 92, 93] and energy consumption [26, 87] are important to meet the requirements in edge and IoT computing. Especially latency plays a vital role in scheduling containerized applications because the primary goal of edge computing is to respond faster than comparable cloud deployments. K8s does not provide a built-in mechanism to run deployments based on latency and bandwidth. This might limit the applicability in cloud-edge environments [26, 87, 89, 92, 93]. Furthermore, the default KS assigns pods to nodes by the usage of node filtering, sorting, and scoring. New workloads are scheduled one by one at a time. Potentially desirable priorities assigned to pods are not considered [21, 26, 86, 92]. In addition, the default KS assigns pods to nodes by using static resource demand and supply. Resource requirements of a containerized application are defined statically by the developers. During the scheduling process, eligible nodes are selected based on the remaining static resources. After the successful assignment, the available resources are reduced by the demanded resources of the container. Figure 5 shows an example of the static resource assignment: Container C_0 requires 0.25 vCPU and 256 MB of memory. As next step, the KS applies node filtering, sorting, and scoring and finally elects worker node W_1 in this example. The available resources on W_1 are reduced by the demanded resources of C_0. This temporarily leads to 3.75 vCPU and 3840 MB of memory (not shown in W_1). This static resource-based procedure could lead to unassigned pods if no suitable node can be found for an assignment. K8s will never move pods from one node to another in order to free up resources. In addition, this procedure may cause under-utilization of particular nodes if resource assignments are far away from the real usage [26, 92]. Limitation 2 shows how the HPA is acting if particular thresholds are exceeded. In this example, the HPA scales out to two additional instances, C_0' and C_0". Depending on the available resources of the other nodes, pods are assigned equally to the remaining nodes. This results in one additional pod with C_0' on W_1 and one additional pod with C_0" on W_3, which finally leads to 3.5 vCPU and 3584 MB of memory on W_1. As already mentioned, the scaling approach does only work based on CPU and memory utilization that is tracked in real-time. Therefore, these and other metrics (e.g., latency and bandwidth) could even be used to trigger offloading and scaling in cloud-edge systems. However, it is noteworthy that the costs for shifting applications or tasks from cloud to edge or vice versa should be regarded in scaling and offloading activities. This might prevent unnecessary offloading actions that are also reducing carbon footprint [26, 87]. Since optimizing latency is one of the primary goals in cloud-edge orchestrations, multi-container applications in different pods are supposed to be placed on the same node or on nodes close to each other [26, 92]. As shown in limitation 3, K8s does not necessarily follow this requirement because it tries to achieve a uniform distribution of applications among nodes. In this example, only C_1 and C_2 are deployed in pods on the same node (W_2), whereas C_3 is deployed in one pod on W_3. This finally leads to 0.5 vCPU and

0 MB of memory on W_2 and 0.5 vCPU and 1280 MB of memory on W_3. The last downside in regards to resource-awareness is the missing understanding of network topology. K8s treats all nodes as homogeneous with similar capabilities even if they are placed at geographically different locations. For example, W_2 and W_3 are low-performance edge devices with a weak CPU and slow main memory and are placed at the edge, as depicted in limitation [4]. However, cloud-edge architectures are usually consisting of heterogeneous nodes in a geographically distributed environment [26, 86, 87, 89, 91–93, 95].

Architectural Shortcomings. The second category of shortcomings of K8s as cloud-edge orchestration systems are architecture-related. First, all worker nodes in K8s are running k-proxy and kubelet as system components to accept network traffic for serving network services and to interact with the master node to receive new instructions. However, as shown in limitation [5], both components are permanently requested, which might burden low-end devices and mitigate performance. K8s reveals a centralized organization where the control plane is managing the set of worker nodes. In edge computing, the decentralized nature is one core characteristic that might break if K8s is used to assign workloads to nodes. Even if the control plane is fully replicated and running in high availability mode [90], as shown in limitation [6], the requirement providing computational resources in a decentralized manner is violated [86, 92, 94]. Limitation [7] covers that edge computing networks consist of a large number of devices, as already discussed in Section 2.2. However, one single K8s cluster can manage at most 5000 nodes with 150000 pods and 300000 containers.[11] Very large edge computing networks must have additional concepts to overcome this limitation, e.g., by the usage of cluster federation [88, 95]. Lastly, limitation [8] reveals that K8s does not hold a network topology of nodes in the cluster and can not deploy to specific nodes. Hence, the distributed nature of cloud-edge clusters is degraded because all nodes in the cluster are assumed to be homogeneous [86].

4.4. Potential Solutions for Kubernetes as Edge Orchestration Platform

The former section discussed several shortcomings of K8s for cloud-edge orchestration. This section presents potential solutions which help to overcome these shortcomings.

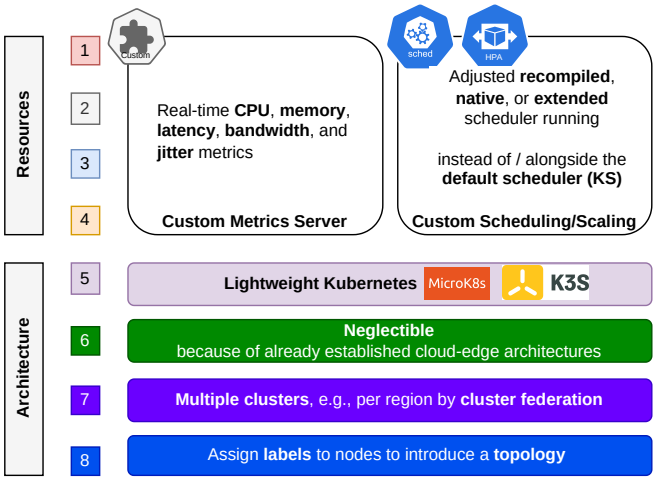

Figure 6. Potential Solutions for Kubernetes as Cloud-Edge Orchestration System

Providing Resource-Awareness. Figure 6 shows potential solutions for the shortcomings of K8s for edge orchestration. The solutions are mapped to the shortcomings described in Figure 5 with the corresponding numbers [1]-[8]. The most apparent limitation of K8s for edge orchestration is the missing capability to consider other metrics like CPU and memory resources, as referenced by [1] and [2]. This applies especially to latency, bandwidth, energy consumption, and costs for offloading and scheduling activities, also in regards to real-time monitoring capabilities. Several authors implemented the collection of custom metrics, like latency and bandwidth, with additional containerized applications, which are running alongside the default metrics server. The collected metrics are further analyzed by custom schedulers that are running alongside the default KS [86, 88, 89, 94, 95]. In addition, the expandability of K8s allows for an implementation of custom and native schedulers that can run in collaboration with the default scheduler or exclusively.[12] Also, there are already implementations that are using the KS with so-called scheduler extenders to alter the scheduling algorithm and outcome of the default KS [26, 92]. For example, the filter and scoring step based on predicates can be modified. Recompiling the default KS with modified and additional predicates leads to overcoming the already outlined shortcomings, as shown by [92]. Another possibility to alter the workload allocation policy is to calculate new deployments with custom orchestrators running as containers in the cluster, as done by [91, 93]. Moreover, the issue with unscheduled pods in case of an inefficient distribution of pods on

[11] Kubernetes documentation - Considerations for large clusters

[12] Kubernetes documentation - Configure Multiple Schedulers

nodes must be addressed. In this case, schedulers can divide the deployment and distribute the containers to different nodes [92]. Moving containers might be an option as well [89]. However, this solution lets the issue of static resource assignments unresolved. A potential under- and over-utilization is still at risk. For this, custom schedulers and custom metrics servers can be implemented to allow for real-time metrics, even during scheduling. Placing multi-container applications on different nodes is a further issue, as shown in limitation 3. To optimize latency, applications consisting of multiple containers should be actually deployed on the same node, respectively, on nodes near to each other. Based on labels, so-called affinities, and custom schedulers, K8s can be forced to perform the deployment only on a subset of nodes [89, 91–93]. This strategy is also used to introduce a network topology covered as an architectural shortcoming. Lastly, K8s assumes all nodes to be homogeneous as summarized by limitation 4. This is not a problem in the first place since scheduling and scaling with the HPA works with the nodes' individual resource supply. However, as already discussed, the performance may suffer if devices with weak specifications are treated equally as powerful devices. Since cloud-edge architectures consist of heterogeneous devices, deployments must be fairly distributed. For this, labels, affinities, and custom schedulers can be used to provide efficient deployments [26].

Implementing Cloud-Edge Architectures. K8s requires the nodes to run additional components to operate as a cluster member, as depicted in limitation 5. This involves some additional overhead on the single cluster members. However, there are plenty of studies that have benchmarked K8s and lightweight distributions that are especially suitable for the edge and IoT devices. As a result, the additional components do only have a small impact on the performance, especially if lightweight K8s is used [96]. Actually, cloud-edge architectures should be in line with the characteristics of a decentralized architecture, as referenced by 6. Therefore, they should not use any centralized architecture, as is the case with K8s. However, this requirement can be relaxed because there are a lot of examples already that are using centralized architectures for all essential activities of orchestration [86, 89, 91]. Especially agent-based approaches have already been implemented with K8s in the field of edge computing [86, 94]. Limitation 7 shows that single K8s clusters have a limitation of at least 5000 nodes with 150000 pods and 300000 containers in total. This limitation is solvable by introducing multiple independent clusters (e.g., by region) or by cluster federation, as shown by [88, 95]. The missing network topology of K8s can be seen as the most serious issue for running cloud-edge orchestrations and is listed as limitation 8. Section 2.2 discussed various provision models that must be supported by edge orchestration. For example, edge offloading requires detailed knowledge about the location and assignments of nodes to layers. Hence, for orchestration of cloud-edge environments, K8s must be aware of the network topology to support the different edge technologies and provision models. As a potential solution, labels can be assigned to nodes, and with so-called affinities and anti-affinities, the assignment of pods to nodes can be controlled. The solutions that have been investigated in this paper use affinities to address a particular layer and add supplementary context information, like the geographical target location [89, 91–93] or the device type [26].

5. Limitations of the Proposed Solutions

The investigated implementations showed plenty of solutions to overcome the mentioned resource- and architecture-related limitations. However, these solutions still have unresolved issues and shortcomings that must be regarded when running cloud-edge architectures. This section presents an overview of the shortcomings we identified in the considered solutions. In line with Section 4.3 and Section 4.4, we present the limitations divided in resource and architectural solutions. Table 1 shows an overview of the resource and architectural requirements and their degree of achievement. We differentiate between fully supported (●), partially supported (❍), not supported (*no circle*), or where no details could be found (✲).

5.1. Resource-related Solutions

Most of the considered solutions are regarding the *cloud* and the *edge* layer. The most frequent architectural design consists of a cloud node and a set of worker nodes located in the edge layer. The cloud node runs the control plane with modified components and the edge nodes are in charge of running the actual workloads. However, there are plenty of solutions that include only the edge layer in their architecture. Most of the authors have used *custom* containerized schedulers that are replacing the *native* KS. [93] and [26] are running a *custom* scheduler [93] or an *extender* [26] alongside an unmodified default KS (❍). Some proposals are working without any modifications on the scheduling component and have used implicit scheduling [90, 91]. Another approach used only affinities and anti-affinities to modify the scheduling behavior [91]. Only one solution provided a *recompiled* scheduling component that is responsible for the entire cloud-edge orchestration [92]. Almost all solutions considered at least *CPU*, *memory*, and *disk* resources. Most of the solutions are only aware of static

Table 1. Comparison of Resource-Awareness and Architectural Capabilities in Different Kubernetes Implementations for Cloud-Edge Architectures [58]

		Resource-Awareness														Architectural Capabilities										
		Cloud	*Edge*	*IoT*	*Recompiled*	*Native*	*Extender*	*Custom*	*CPU*	*Memory*	*Disk*	*Energy*	*Latency*	*Bandwidth*	*K8s API*	*Custom*	*Cloud*	*Edge*	*IoT*	*Scaling*	*Offloading*	*Edge-only*	*Cluster*	*Control plane*	*Cluster storage*	*Cloud*	*Edge*	*Replicated*
Authors	**Year**	**Layer**			**Scheduler**				**Resources**				**Network**		**Metrics**		**Topology**			**Provision model**			**Fault-tolerance**			**Container registry**		
[86] Han	2021	●	●					●	●	●	●		●			●	●	●		●	●	●	●			●		
[92] Kayal	2020		●		●				❍	❍	❍		❍		❍			●				●				●		
[87] Kaur	2020	●	●					●	●	●		●		●		✲	❍	❍		❍	❍	●	●				✲	
[93] Eidenbenz	2020		●			❍		●	❍	❍	❍		●		●			●				●					✲	
[88] Goethals	2020	●	●					●	❍	❍	❍		●		❍	●		●				●					✲	
[21] Ogbuachi	2020		●					●	●	●	●		●			●		●				●				●		
[26] Santos	2019	●	●			❍	●	●	❍	❍	❍		❍	❍	❍		●	●		●	❍	●				●	●	●
[94] Casquero	2019		●					●	❍	❍	❍				❍			●				●				●		
[89] Haja	2019	●	●					●	❍	❍	❍		●		❍	●	●	●			●	●					✲	
[90] Javed	2018	●	●			❍			❍	❍	❍				❍		❍	❍		❍	❍	❍		●	❍		✲	
[91] Wöbker	2018	●	●			❍			❍	❍	❍				❍		●	●		●	❍	●					●	

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

resource assignments (❍) for scheduling decisions. A few solutions used real-time metrics (●) even for scheduling decisions. [87] also integrated the *energy* consumption for such decisions that might be important when devices are running on battery. Latency-aware deployments are inevitable in edge computing. For this, *latency* and *bandwidth* must be considered properly. Most solutions have used periodic latency measurements (●) to achieve those deployments. Other solutions relied on predefined and static assignments (❍). For bandwidth, which was rarely considered in the investigated solutions, there are periodic (●) checks [87] as well as static (❍) definitions [26]. For measuring the current CPU and memory utilization, usually, an unmodified version of the *K8s API* (❍) is used. To extend the set of metrics that can be used for scheduling and scaling, plenty of the investigated solutions just used the unmodified K8s API. [88] and [89] enriched this API with *custom* (●) containers and functionalities. One solution [93] is replacing the K8s API entirely with a version that is fully compatible to K8s and measures the latency between nodes. Missing explanations were also the case (✲).

5.2. Architectural Solutions

After reviewing resource-awareness, we will evaluate the architectural capabilities. Topology-awareness can be seen as one of the most important requirements. Only a few authors allow for deployment and scheduling to *cloud* and *edge* nodes explicitly (●). The solutions presented by [87, 90] take cloud and edge nodes as one single topology (❍) to run workloads. Mostly, only the edge layer is fully considered. Also, workloads are not supposed to be shifted among the cloud and edge layers. Solutions proposed by [26, 86, 91] support the *scaling* model where workloads can be run on cloud and edge simultaneously (●). The implemented schedulers of [87, 90] support scaling to edge only implicitly (❍). Explicit *offloading* (●) is seldom supported because, as already outlined, the cloud layer is usually used for the cluster managers and orchestrators and not workloads. Implicit *offloading* (❍) occurs if nodes are failing and workloads are shifted and restarted on other nodes by K8s automatically. This is only possible for architectures where workloads are potentially running on cloud and edge. Nearly all architectures fully facilitate *edge-only* deployments where workloads are deployed on the edge layer to reduce latency (●). Perhaps, workloads are also moved across the edge layer during application runtime to improve the latency further. In case users or the HPA are increasing the number of replicas and only edge nodes are available as computational resources, workloads are partially scaled out (❍). Fault-tolerance of K8s itself was poorly covered in the orchestration architectures. Even if the deployments of applications offer high availability, crashing master nodes that are carrying the *control plane* with essential components for scheduling, scaling, monitoring, and resiliency features mitigate the fault-tolerance of the entire architecture and limit the application in critical areas. [86] and [87] are using geographically independent (●) K8s *clusters* for fault-tolerance. The control plane, however, is not replicated using at least three master nodes (●) for production

environments.[13] A fault-tolerant architecture was only developed by [90] with three master nodes that are running replicas of the distributed key-value store etcd as *cluster storage*. This kind of deployment is also called stacked cluster storage setup (❍).[14] As a general recommendation, it is worth considering decoupling etcd from the master nodes and providing an independent external cluster storage (●). This external cluster storage achieves better resiliency and reduces the load on the master nodes as well. Particularly, this is recommended for environments that need to handle a large number of nodes.[15] Container registries are mainly located in the cloud in a non-geographically *replicated* manner. However, the geographical location of these registries is essential because the highly distributed edge nodes need to download containerized workloads in a small amount of time, for example, if workloads are supposed to be scaled, offloaded, or moved following the edge-only provision model. It is inevitable to follow a reasonable strategy by placing container registries at different locations to follow the superior goal of latency reduction in edge computing and edge orchestration. In the set of implementations, only one implementation [26] has used a fully replicated container registry across *cloud* (●) and *edge* (●). One further solution, provided by [91], placed the container registry on the edge layer (●) to accelerate the deployment process. In most of the works considered in this survey, the placement of container registries is neglected and not specified or further discussed (✲).

6. Discussion

This section discusses the evaluation of the K8s-based solutions in the former section. In Section 6.1, we present a summary of our results by answering the research questions. Afterward, Section 6.2 discusses the limitations of our study. We conclude this section with a short assessment if the efforts making K8s ready for the edge computing and edge orchestration, even for SCs, should be retained.

6.1. Findings

In this work, we performed an analysis of the capabilities of K8s to orchestrate cloud-edge architectures, also for SCs. For this, we obtained fundamental characteristics of the SC concept, edge computing, edge orchestration, and K8s as container orchestration platform.

We analyzed several studies that considered K8s as a basis for cloud-edge orchestration activities. The findings from these steps allow us to answer **RQ1**, in which we want to identify the **most critical requirements for cloud-edge orchestration and their coverage by K8s**.

RQ1. We categorized the essential requirements of cloud-edge orchestration in resource- and architecture-related ones. First, we identified the set of *layers* a cloud-edge orchestrator should manage. Second, we obtained that considering real-time *resource* utilization and providing *network* awareness are a primary focus, especially with dynamic changes over time. For the architectural requirements, it is important to consider the network *topology* as most important aspect. Further, the support of different *provision models* and the implementation of *fault-tolerance*, also for all core services, are essential for cloud-edge environments with production readiness. K8s already provides some of these required aspects. Scheduling and horizontal scaling based on CPU and memory are supported. In addition, K8s allows for setting up clusters in high availability mode. As stated by [83], cloud-edge architectures are subject to dynamic changes in the infrastructure. In specific, it is a common event that the number of nodes changes over time by adding and removing nodes [66]. However, this was not covered in our study in detail because K8s supports adding and removing nodes during runtime by default.[16] Security was also not in the scope of our study because the communication of the system components of K8s are secured via HTTPS by default and provide a built-in system for authentication and authorization.[17] As already indicated, there are various shortcomings of K8s for cloud-edge orchestration. First, resource-awareness is only partially covered. K8s is mainly built for the cloud and not designed for working on a heterogeneous structure [93]. A further issue is the missing support for real-time resource utilization during scheduling. The most severe shortcoming is that no network-related metrics in scheduling and scaling tasks are considered. Regarding architectural shortcomings, the missing topology-awareness limits the orchestration capabilities significantly. Hence, the usage of K8s might be restricted to edge-only deployments exclusively. K8s implements a centralized scheduler to perform assignments of workloads to nodes and to manage the overall cluster. This violates the decentralized notion of edge computing. We argue that this issue can be relaxed because there are already established non-K8s-based cloud-edge architectures, even with centralized components that take care of the architecture [97, 98]. For orchestration activities like

[13] Kubernetes documentation - Production environment
[14] Kubernetes documentation - Options for Highly Available Topology
[15] Kubernetes documentation - Production environment
[16] Kubernetes documentation - Safely Drain a Node
[17] Kubernetes documentation - Controlling Access to the Kubernetes API

task offloading, a centralized management is used very frequently, as shown in a comprehensive survey by [85].

This work also analyzed the state-of-the-art of K8s-based cloud-edge orchestration. In addition, we discussed general limitations for edge orchestration. These steps allow answering **RQ2** that aims to **investigate the benefits and drawbacks of cloud-edge architectures based on K8s**.

RQ2. There are many solutions that solve essential shortcomings of K8s for edge computing. Especially the capabilities in regards to resource-awareness have been significantly improved by adding custom schedulers that regard other resources like CPU and memory. In specific, the capabilities for network-aware deployments were targeted. Besides that, topology-awareness has been added to get K8s ready for the edge by overcoming architectural shortcomings. Nonetheless, there is still room for improvement. Support for multiple provision models, fault-tolerance of the cluster architecture, and the placement of container registries must be implemented.

Based on the qualitative analysis of K8s-based implementations for edge orchestration, we **identified drawbacks and are able to assess the solvability and the corresponding amount of effort** to answer **RQ3**.

RQ3. From our understanding, most of the problems are solvable in an appropriate amount of time because partial solutions can be combined to derive a unified solution. We are convinced that resource-related issues can be solved. Our survey showed that many solutions considered only the edge layer to be managed by K8s. However, if the number of nodes and devices is increasing nonetheless, multiple clusters can be connected by cluster federation[18]. Furthermore, many components of K8s can be replaced or extended, for example, the scheduler to implement the resource-related shortcomings. Implementing a native scheduler with custom scheduling algorithms and scaling policies for the edge is a complex task [93]. This might be the reason why we could not obtain a large number of native implementations. Lastly, K8s allows modifying the metrics sever to add further metrics that can enhance the overall placement decisions.

We conclude that essential architectural shortcomings can be solved as well. In specific, network topology, support for different provision models, deployments, and cluster setups with high availability and placement of container registries were not a major focus in the investigated solutions. Essential capabilities like fault-tolerance by replicating clusters, their core services, or geographically distributed container registries can be implemented quickly. However, this requires accepting a trade-off. In favor of a unified and universal cloud-edge orchestration, the criteria following a decentralized architecture might be relaxed.

Finally, this work presented an overview of essential aspects of SCs. Also, a short review of already established approaches for edge computing in SC contexts has been covered. This supports us to answer **RQ4** appropriately where we need to discuss if **K8s is an eligible candidate for providing demand- and supply-aware deployments for a SC context**.

RQ4. As outlined in Section 2.1, a SC aims to improve the quality of living in urban life. From an IT-related perspective, this is mostly done by the usage of IoT technology. However, new challenges arise due to the rising number of devices and the emergence of critical services, like smart traffic control systems. Based on our theoretical investigation, we can argue that multi-tenancy, security, fault-tolerance, and providing low-latency for real-time systems are major concerns that are not fully resolved by the existing solutions so far. A revised usage of K8s can positively contribute to overcoming these issues. Also, in SCs, the number of applications is continuously increasing. The types of applications are also changing over time, which requires a flexible management of an IT infrastructure at scale. For such large-scale deployments, an efficient orchestration system based on K8s might be beneficial. From our perspective, K8s fulfills several essential requirements already that can be further improved. As discussed, real-time resource supplies and demands, network-related metrics, and the missing understanding of network topology are major issues that should be solved if K8s is working as a unified solution for deployments in SCs.

6.2. Threats to Validity

We aligned the set of evaluation criteria to critical requirements of SCs, edge computing, edge orchestration, and the shortcomings of K8s. As a matter of fact, the set of evaluation criteria might be incomplete and a simplification. Also, it is still to question if the custom implementations are comparable because they follow different approaches and objectives by solving the shortcomings of K8s. In addition, we investigate centralized, decentralized, and mixed architectures equally without further differentiation. Nonetheless, we argue that mission-critical requirements in edge computing for SC contexts must be met. Our evaluation assumed that all solutions must support the essential provision models in edge computing. According to our study, edge-only deployments are the most frequent provision model. Therefore, it is to question whether all solutions must cover all provision models. The same applies to the layer aspect, where the layer might depend on the goals of the proposed solutions. Lastly,

[18] GitHub - kubefed

we considered a general K8s-based solution for edge orchestration and did not filter the set of solutions by those solutions that explicitly attempt to be used for SCs. Some solutions [26, 88] were explicitly used in SC environments and contributed to the validity of our evaluation. Nonetheless, the principles, core contributions, and requirements of related cloud-edge and edge orchestration architectures are not significantly different from architectures that cover SCs in specific.

7. Conclusion and Future Work

The vision of SCs equips urban environments with ICT to improve citizens' quality of life. However, the large number of heterogeneous applications and devices with different resource requirements constitute new challenges. Cloud-edge orchestration offers an efficient way to distribute workloads based on resource demands and supplies. Especially reducing the communication latency for real-time applications is one of the main incentives to focus on these activities. Mostly, container technology is used to distribute those workloads on a set of nodes. For managing large and complex container orchestration, K8s is considered to be the state-of-the-art solution. However, many authors claimed that K8s, which is mainly built for cloud computing, lacks significant features. Therefore, they contributed several improvements to make it ready for the edge.

This paper evaluated the most recent architectural proposals that tackle the most significant issues of edge orchestration. To perform our evaluation, we based our survey on the essential requirements of SCs, edge computing, edge orchestration, and finally native K8s. This paper contributes a state-of-the-art overview of established cloud-edge architectures that are also suitable to manage the complexities of SC architectures. It can be seen as an overview of requirements that are already solved by these solutions and issues that are still unresolved.

We identified plenty of benefits and drawbacks of the investigated architectures, which also influence the applicability of K8s to SC contexts. Issues like real-time resource utilization, network-awareness, and network topology have been solved quite well so far. However, aspects like providing multiple provision models (i.e., offloading, scaling, and edge-only deployments) still have room for improvement. In addition, fault-tolerant cluster architectures for managing cloud-edge environments are still in the early stages. The ideal placement strategy for container registries was also not in focus of the cloud-edge environments we considered. Furthermore, we assessed if the shortcomings of K8s are solvable with an appropriate amount of effort to enable a long-term cloud-edge orchestration with K8s. In conclusion, since there are already partial solutions for most of these issues, K8s should still be retained as a container orchestration platform for cloud-edge systems. Since K8s offers solutions for essential requirements of cloud-edge orchestration and SC environments, further research should still be in focus. Using K8s as an advanced orchestration system in SC context can foster the implementation of valuable services that have a positive impact on people's and society's well-being. Furthermore, the government can be released through these developments since more and more administrative tasks can be automated.

We still consider K8s as container orchestration platform for cloud-edge architectures and plan to implement a unified orchestration platform. A high degree in standardization (e.g., custom schedulers, custom metric APIs, and architectural recommendations) can support the relevance of K8s in cloud-edge architectures. In the future, we plan to follow this defined goal by providing architectural blueprints that help to deploy K8s in production, in specific for a SC context that requires a high degree of flexibility for different demands. Furthermore, we want to foster the standardization of generic cloud-edge orchestration strategies, algorithms, and policies. Detailed instructions and providing standardized APIs can contribute to a broader usage of K8s, even for standalone edge orchestration algorithms, strategies, and policies that are using their own container orchestration solution. The public availability of architectural blueprints in a centralized repository with established standalone solutions can foster the popularity of K8s for edge orchestration. In order to examine the feasibility of the proposed architectural blueprints, we want to provide reference implementations that are tested at scale to evaluate if K8s can finally run large-scale cloud-edge architectures.

References

[1] Sebrechts, M., Borny, S., Wauters, T., Volckaert, B. and Turck, F.D. (2021) Service relationship orchestration: Lessons learned from running large scale smart city platforms on kubernetes. *IEEE Access* **9**: 133387–133401.

[2] Ramaprasad, A., Sánchez-Ortiz, A. and Syn, T. (2017) A unified definition of a smart city. In *Lecture Notes in Computer Science*, 13–24.

[3] Santos, J., Vanhove, T., Sebrechts, M., Dupont, T., Kerckhove, W., Braem, B., Seghbroeck, G.V. *et al.* (2018) City of things: Enabling resource provisioning in smart cities. *IEEE Communications Magazine* **56**(7): 177–183.

[4] Su, K., Li, J. and Fu, H. (2011) Smart city and the applications. In *2011 International Conference on Electronics, Communications and Control (ICECC)* (IEEE).

[5] Gaur, A., Scotney, B., Parr, G. and McClean, S. (2015) Smart city architecture and its applications based on IoT. *Procedia Computer Science* **52**: 1089–1094.

[6] hoon Kim, T., Ramos, C. and Mohammed, S. (2017) Smart city and IoT. *Future Generation Computer Systems* **76**: 159–162.

[7] Eremia, M., Toma, L. and Sanduleac, M. (2017) The smart city concept in the 21st century. *Procedia Engineering* **181**: 12–19.

[8] Atzori, L., Iera, A. and Morabito, G. (2010) The internet of things: A survey. *Computer Networks* **54**(15): 2787–2805.

[9] Mohamed, N., Al-Jaroodi, J. and Jawhar, I. (2019) Towards fault tolerant fog computing for IoT-based smart city applications. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)* (IEEE).

[10] Tcholtchev, N. and Schieferdecker, I. (2021) Sustainable and reliable information and communication technology for resilient smart cities. *Smart Cities* **4**(1): 156–176.

[11] Kumar, V., Laghari, A.A., Karim, S., Shakir, M. and Brohi, A.A. (2019) Comparison of fog computing & cloud computing. *Int. J. Math. Sci. Comput* **1**: 31–41.

[12] Lau, B.P.L., Marakkalage, S.H., Zhou, Y., Hassan, N.U., Yuen, C., Zhang, M. and Tan, U.X. (2019) A survey of data fusion in smart city applications. *Information Fusion* **52**: 357–374.

[13] Shahrour, I. and Xie, X. (2021) Role of internet of things (IoT) and crowdsourcing in smart city projects. *Smart Cities* **4**(4): 1276–1292.

[14] Balakrishna, C. (2012) Enabling technologies for smart city services and applications. In *2012 Sixth International Conference on Next Generation Mobile Applications, Services and Technologies* (IEEE).

[15] Laghari, A.A., He, H., Shafiq, M. and Khan, A. (2016) Assessing effect of cloud distance on end user's quality of experience (QoE). In *2016 2nd IEEE International Conference on Computer and Communications (ICCC)* (IEEE).

[16] Cao, J., Zhang, Q. and Shi, W. (2018) *Edge Computing: A Primer* (Springer International Publishing).

[17] Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L. (2016) Edge computing: Vision and challenges. *IEEE Internet of Things Journal* **3**(5): 637–646.

[18] Caragliu, A., Bo, C.D. and Nijkamp, P. (2011) Smart cities in europe. *Journal of Urban Technology* **18**(2): 65–82.

[19] Satyanarayanan, M. (2017) Edge computing. *Computer* **50**(10): 36–38.

[20] Laghari, A.A., He, H., Memon, K.A., Laghari, R.A., Halepoto, I.A. and Khan, A. (2019) Quality of experience (QoE) in cloud gaming models: A review. *Multiagent and Grid Systems* **15**(3): 289–304.

[21] Ogbuachi, M.C., Reale, A., Suskovics, P. and Kovács, B. (2020) Context-aware kubernetes scheduler for edge-native applications on 5g. *Journal of communications software and systems* **16**(1): 85–94.

[22] Morabito, R. (2017) Virtualization on internet of things edge devices with container technologies: A performance evaluation. *IEEE Access* **5**: 8835–8850.

[23] Pahl, C. and Lee, B. (2015) Containers and clusters for edge cloud architectures – a technology review. In *2015 3rd International Conference on Future Internet of Things and Cloud* (IEEE).

[24] Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M. and Steinder, M. (2015) Performance evaluation of microservices architectures using containers. In *2015 IEEE 14th International Symposium on Network Computing and Applications*: 27–34.

[25] Barika, M., Garg, S., Zomaya, A.Y., Wang, L., Moorsel, A.V. and Ranjan, R. (2019) Orchestrating big data analysis workflows in the cloud. *ACM Computing Surveys* **52**(5): 1–41.

[26] Santos, J., Wauters, T., Volckaert, B. and Turck, F.D. (2019) Resource provisioning in fog computing: From theory to practice †. *Sensors* **19**(10): 2238.

[27] Barriga, J.J., Sulca, J., León, J., Ulloa, A., Portero, D., García, J. and Yoo, S.G. (2020) A smart parking solution architecture based on LoRaWAN and kubernetes. *Applied Sciences* **10**(13): 4674.

[28] Ogawa, K., Kanai, K., Nakamura, K., Kanemitsu, H., Katto, J. and Nakazato, H. (2019) IoT device virtualization for efficient resource utilization in smart city IoT platform. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)* (IEEE).

[29] Kristiani, E., Yang, C.T., Huang, C.Y., Wang, Y.T. and Ko, P.C. (2020) The implementation of a cloud-edge computing architecture using OpenStack and kubernetes for air quality monitoring application. *Mobile Networks and Applications* : 1–23.

[30] Albino, V., Berardi, U. and Dangelico, R.M. (2015) Smart cities: Definitions, dimensions, performance, and initiatives. *Journal of Urban Technology* **22**(1): 3–21.

[31] Dameri, R. and Cocchia, A. (2013) Smart city and digital city: twenty years of terminology evolution. *ITAIS* : 1–8.

[32] Wenge, R., Zhang, X., Dave, C., Chao, L. and Hao, S. (2014) Smart city architecture: A technology guide for implementation and design challenges. *China Communications* **11**(3): 56–69.

[33] Anthopoulos, L. (2015) Defining smart city architecture for sustainability.

[34] Yin, C., Xiong, Z., Chen, H., Wang, J., Cooper, D. and David, B. (2015) A literature survey on smart cities. *Science China Information Sciences* **58**(10): 1–18.

[35] Al-Farabi, M., Chowdhury, M., Readuzzaman, M., Hossain, M., Sabuj, S. and Hossain, M. (2018) Smart environment monitoring system using unmanned aerial vehicle in bangladesh. *EAI Endorsed Transactions on Smart Cities* .

[36] Hollands, R.G. (2008) Will the real smart city please stand up? *City* **12**(3): 303–320.

[37] Park, S., Park, S., Park, L., Park, S., Lee, S., Lee, T., Lee, S. *et al.* (2018) Design and implementation of a smart IoT based building and town disaster management system in smart city infrastructure. *Applied Sciences* **8**(11): 2239.

[38] Sakhardande, P., Hanagal, S. and Kulkarni, S. (2016) Design of disaster management system using IoT based interconnected network with smart city monitoring. In *2016 International Conference on Internet of Things and Applications (IOTA)* (IEEE).

[39] Tönjes, R., Barnaghi, P., Ali, M., Mileo, A., Hauswirth, M., Ganz, F., Ganea, S. *et al.* (2014) Real time iot stream processing and large-scale data analytics for smart city applications. In *European Conference on Networks and*

Communications (sn): 10.

[40] Hafez, R. (2018) A methodical plan towards smart economy in new egyptian cities. *EAI Endorsed Transactions on Smart Cities* .

[41] Mulligan, C.E.A. and Olsson, M. (2013) Architectural implications of smart city business models: an evolutionary perspective. *IEEE Communications Magazine* **51**(6): 80–85.

[42] Saharan, S., Kumar, N. and Bawa, S. (2020) An efficient smart parking pricing system for smart city environment: A machine-learning based approach. *Future Generation Computer Systems* **106**: 622–640.

[43] Baker, N., Szabo-Müller, P. and Handmann, U. (2018) Transfer learning-based method for automated e-waste recycling in smart cities. *EAI Endorsed Transactions on Smart Cities* .

[44] Siregar, B., Nasution, A.B.A. and Fahmi, F. (2016) Integrated pollution monitoring system for smart city. In *2016 International Conference on ICT For Smart Society (ICISS)* (IEEE).

[45] Honarvar, A.R. and Sami, A. (2019) Towards sustainable smart city by particulate matter prediction using urban big data, excluding expensive air pollution infrastructures. *Big Data Research* **17**: 56–65.

[46] Bagchi, S., Siddiqui, M.B., Wood, P. and Zhang, H. (2019) Dependability in edge computing. *Communications of the ACM* **63**(1): 58–66.

[47] Laghari, A.A., Jumani, A.K. and Laghari, R.A. (2021) Review and state of art of fog computing. *Archives of Computational Methods in Engineering* **28**(5): 3631–3643.

[48] Svorobej, S., Bendechache, M., Griesinger, F. and Domaschka, J. (2020) Orchestration from the cloud to the edge. In *The Cloud-to-Thing Continuum*, 61–77.

[49] Laghari, A.A., Wu, K., Laghari, R.A., Ali, M. and Khan, A.A. (2021) A review and state of art of internet of things (IoT). *Archives of Computational Methods in Engineering* **29**(3): 1395–1413.

[50] Laghari, A., Laghari, R., Wagan, A. and Umrani, A. (2018) Effect of packet loss and reorder on quality of audio streaming. *ICST Transactions on Scalable Information Systems* .

[51] Premsankar, G., Francesco, M.D. and Taleb, T. (2018) Edge computing for the internet of things: A case study. *IEEE Internet of Things Journal* **5**(2): 1275–1284.

[52] Bilal, K., Khalid, O., Erbad, A. and Khan, S.U. (2018) Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Computer Networks* **130**: 94–120.

[53] Ahmed, E. and Rehmani, M.H. (2017) Mobile edge computing: Opportunities, solutions, and challenges. *Future Generation Computer Systems* **70**: 59–63.

[54] Satyanarayanan, M., Bahl, P., Caceres, R. and Davies, N. (2009) The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing* **8**(4): 14–23.

[55] Al-Tarawneh, M.A.B. (2020) Mobility-aware container migration in cloudlet-enabled IoT systems using integrated muticriteria decision making. *International Journal of Advanced Computer Science and Applications* **11**(9).

[56] Qiu, Y., Lung, C.H., Ajila, S. and Srivastava, P. (2017) LXC container migration in cloudlets under multipath TCP. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)* (IEEE).

[57] Qiu, Y., Lung, C.H., Ajila, S. and Srivastava, P. (2019) Experimental evaluation of LXC container migration for cloudlets using multipath TCP. *Computer Networks* **164**: 106900.

[58] Böhm, S. and Wirtz, G. (2021) Towards orchestration of cloud-edge architectures with kubernetes. *EAI Edge-IoT 2021 - 2nd EAI International Conference on Intelligent Edge Processing in the IoT Era* .

[59] Varghese, B., Wang, N., Barbhuiya, S., Kilpatrick, P. and Nikolopoulos, D.S. (2016) Challenges and opportunities in edge computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*: 20–26.

[60] Bonomi, F., Milito, R., Zhu, J. and Addepalli, S. (2012) 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 '12* (ACM Press).

[61] Vaquero, L.M., Cuadrado, F., Elkhatib, Y., Bernal-Bernabe, J., Srirama, S.N. and Zhani, M.F. (2019) Research challenges in nextgen service orchestration. *Future Generation Computer Systems* **90**: 20–38.

[62] Aazam, M., Zeadally, S. and Harras, K.A. (2018) Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities. *Future Generation Computer Systems* **87**: 278–289.

[63] Hong, C.H. and Varghese, B. (2019) Resource management in fog/edge computing. *ACM Computing Surveys* **52**(5): 1–37.

[64] da Silva, D.M.A., Asaamoning, G., Orrillo, H., Sofia, R.C. and Mendes, P.M. (2019) 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).

[65] Velasquez, K., Abreu, D.P., Assis, M.R.M., Senna, C., Aranha, D.F., Bittencourt, L.F., Laranjeiro, N. *et al.* (2018) Fog orchestration for the internet of everything: state-of-the-art and research challenges. *Journal of Internet Services and Applications* **9**(1).

[66] Pahl, C., Ioini, N.E., Helmer, S. and Lee, B. (2018) An architecture pattern for trusted orchestration in IoT edge clouds. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)* (IEEE).

[67] Casalicchio, E. (2017) Autonomic orchestration of containers: Problem definition and research challenges. In *Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools* (ACM).

[68] Giordano, A., Spezzano, G. and Vinci, A. (2016) Smart agents and fog computing for smart city applications. In *Smart Cities*, 137–146.

[69] Cicirelli, F., Guerrieri, A., Spezzano, G. and Vinci, A. (2017) An edge-based platform for dynamic smart city applications. *Future Generation Computer Systems* **76**: 106–118.

[70] Hsieh, Y.C., Hong, H.J., Tsai, P.H., Wang, Y.R., Zhu, Q., Uddin, M.Y.S., Venkatasubramanian, N. *et al.* (2018) Managed edge computing on internet-of-things devices for smart city applications. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium* (IEEE).

[71] Barthélemy, J., Verstaevel, N., Forehead, H. and Perez, P. (2019) Edge-computing video analytics for real-time traffic monitoring in a smart city. *Sensors* **19**(9): 2048.

[72] Hossain, S.A., Rahman, M.A. and Hossain, M.A. (2018) Edge computing framework for enabling situation awareness in IoT based smart city. *Journal of Parallel and Distributed Computing* **122**: 226–237.

[73] Ahsaan, S. and Mourya, A. (2018) Prognostic modelling for smart cities using smart agents and IoT: A proposed solution for sustainable development. *EAI Endorsed Transactions on Smart Cities* .

[74] Taleb, T., Dutta, S., Ksentini, A., Iqbal, M. and Flinck, H. (2017) Mobile edge computing potential in making cities smarter. *IEEE Communications Magazine* **55**(3): 38–43.

[75] Al-gaashani, M., Muthanna, M.S.A., Abdukodir, K., Muthanna, A. and Kirichek, R. (2020) Intelligent system architecture for smart city and its applications based edge computing. In *2020 12th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)* (IEEE).

[76] Hussain, M.M., Alam, M.S. and Beg, M.S. (2018) Computational viability of fog methodologies in IoT enabled smart city architectures-a smart grid case study. *EAI Endorsed Transactions on Smart Cities* .

[77] Gheisari, M., Pham, Q.V., Alazab, M., Zhang, X., Fernandez-Campusano, C. and Srivastava, G. (2019) ECA: An edge computing architecture for privacy-preserving in IoT-based smart city. *IEEE Access* **7**: 155779–155786.

[78] Deng, Y., Chen, Z., Yao, X., Hassan, S. and Wu, J. (2019) Task scheduling for smart city applications based on multi-server mobile edge computing. *IEEE Access* **7**: 14410–14421.

[79] Zheng, X., Li, M. and Guo, J. (2020) Task scheduling using edge computing system in smart city. *International Journal of Communication Systems* **34**(6).

[80] Wang, D., Bai, B., Lei, K., Zhao, W., Yang, Y. and Han, Z. (2019) Enhancing information security via physical layer approaches in heterogeneous IoT with multiple access mobile edge computing in smart city. *IEEE Access* **7**: 54508–54521.

[81] Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J. *et al.* (2019) All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture* **98**: 289–330.

[82] Naha, R.K., Garg, S., Georgakopoulos, D., Jayaraman, P.P., Gao, L., Xiang, Y. and Ranjan, R. (2018) Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE Access* **6**: 47980–48009.

[83] Hoque, S., Brito, M.S.D., Willner, A., Keil, O. and Magedanz, T. (2017) Towards container orchestration in fog computing infrastructures. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)* (IEEE).

[84] Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R.H., Morrow, M.J. and Polakos, P.A. (2018) A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials* **20**(1): 416–464.

[85] Wang, J., Pan, J., Esposito, F., Calyam, P., Yang, Z. and Mohapatra, P. (2019) Edge cloud offloading algorithms. *ACM Computing Surveys* **52**(1): 1–23.

[86] Han, Y., Shen, S., Wang, X., Wang, S. and Leung, V.C.M. (2021) Tailored learning-based scheduling for kubernetes-oriented edge-cloud system. *CoRR* .

[87] Kaur, K., Garg, S., Kaddoum, G., Ahmed, S.H. and Atiquzzaman, M. (2020) KEIDS: Kubernetes-based energy and interference driven scheduler for industrial IoT in edge-cloud ecosystem. *IEEE Internet of Things Journal* **7**(5): 4228–4237.

[88] Goethals, T., Volckaert, B. and de Turck, F. (2020) 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*.

[89] Haja, D., Szalay, M., Sonkoly, B., Pongracz, G. and Toka, L. (2019) Sharpening kubernetes for the edge. In *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos on - SIGCOMM Posters and Demos '19* (ACM Press).

[90] Javed, A., Heljanko, K., Buda, A. and Framling, K. (2018) CEFIoT: A fault-tolerant IoT architecture for edge and cloud. In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)* (IEEE): 813–818.

[91] Wöbker, C., Seitz, A., Mueller, H. and Bruegge, B. (2018) Fogernetes: Deployment and management of fog computing applications. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium* (IEEE).

[92] Kayal, P. (2020) 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): 1–6.

[93] Eidenbenz, R., Pignolet, Y.A. and Ryser, A. (2020) Latency-aware industrial fog application orchestration with kubernetes. In *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)* (IEEE).

[94] Casquero, O., Armentia, A., Sarachaga, I., Perez, F., Orive, D. and Marcos, M. (2019) 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).

[95] Goethals, T., DeTurck, F. and Volckaert, B. (2020) Extending kubernetes clusters to low-resource edge devices using virtual kubelets. *IEEE Transactions on Cloud Computing* : 1–1.

[96] Böhm, S. and Wirtz, G. (2021) Profiling lightweight container platforms: Microk8s and k3s in comparison to kubernetes. In *CEUR workshop proceedings* (RWTH Aachen): 65–73.

[97] Taherizadeh, S., Stankovski, V. and Grobelnik, M. (2018) A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers. *Sensors* **18**(9): 2938.

[98] Yu, Z., Wang, J., Qi, Q., Liao, J. and Xu, J. (2018) Boundless application and resource based on container technology. In *Edge Computing – EDGE 2018*, 34–48.

All links were last followed on February, 28, 2022.