Anna-Sofia Susanna Paavonen

Container Orchestration in Edge Computing: A Mapping and Comparison of Container Orchestration Technologies

Bachelor's Thesis information and software technology

May 21, 2024

University of Jyväskylä

Faculty of Information Technology

Author: Anna-Sofia Susanna Paavonen

Contact information: anna-sofia.s.paavonen@jyu.fi

Supervisor: Tuomo Rossi

Title: Container Orchestration in Edge Computing: A Mapping and Comparison of Container Orchestration Technologies

Työn nimi: Konttien orkestrointi reunalaskennassa: konttien orkestrointiteknologioiden kartoitus ja vertailu

Project: Bachelor's Thesis

Page count: 33+0

Abstract: The requirements of edge computing in comparison to the centralized cloud have resulted in the development of container orchestration technologies that are more suitable for the resource constrained and heterogeneous edge environments. The aim of this thesis is to map these edge container orchestration technologies and to compare them against each other based on their features and resource utilization. This comparison will give an understanding of how well these technologies are suited for container orchestration at the edge.

Keywords: edge computing, container orchestration, Kubernetes, resource utilization comparison, feature comparison

Suomenkielinen tiivistelmä: Reunalaskennan mukanaan tuomat vaatimukset ovat aiheuttaneet paineen uusien konttien orkestrointiteknologioiden kehittämiselle. Tämän kandidaatintutkielman tarkoituksena on kartoittaa nämä reunalaskentaa varten kehitetyt konttien orkestrointiteknologiat ja vertailla niitä toisiinsa niiden ominaisuuksien ja resurssienkäytön suhteen. Tämä vertailu antaa kuvan siitä, mitä reunalaskentaan kehitettyjä konttien orkestrointiteknologioita on olemassa ja kuinka hyvin ne soveltuvat käytettäväksi reunalaskennassa.

Avainsanat: reunalaskenta, konttien orkestrointi, Kubernetes, resurssienkäytön vertailu, ominaisuuksien vertailu

List of Figures

List of Tables

Contents

1 Introduction

Today there is a huge number of different IoT-devices that generate a massive amount of data all the time. These IoT-devices are very heterogeneous and many of them need fast data processing to be able to react quickly to, for example, changes in the environment. Cloud computing will not alone be able to meet the requirements of these different IoT-devices. That is why a couple of new computing paradigms, edge computing and fog computing, have arisen. Edge computing and fog computing bring the computing power, applications, storage, and other services closer to the end user and edge devices that are dependent on fast connections and fast data processing (Khan et al. August 2019; Iorga et al. March 2018). There are no clear definitions for edge computing and fog computing, and it is not clear what the differences between these two computing paradigms are. Therefore, in this thesis, for simplicity, these two concepts will be seen as same and only edge computing will be used from now on to describe the aforementioned computing paradigms.

To be able to manage these massive networks of connected devices, their resources, and the applications they are running, orchestration is necessary. There are various container orchestration technologies, such as Kubernetes and Docker Swarm, that were developed for orchestrating cloud services. However, since these technologies have not been specifically designed for edge computing, they are not able to meet the requirements of edge computing and therefore may not be feasible for orchestrating the edge as they are (Hoque et al. July 2017). To solve this problem, some new orchestration technologies and tools have been developed with the requirements of edge computing in mind.

The objective of this thesis is to map these new technologies and tools and to define their strengths and limitations regarding how well they are suited for orchestrating containerized applications at the edge. Knowing the technologies available and their strengths and limitations will help with picking the most feasible orchestration tools for different use cases of edge computing. The research questions are:

1. What is edge computing?
2. What container orchestration technologies developed specifically for edge computing

are there?

3. What are the strengths and limitations of these technologies?

This thesis was conducted as a semi systematic mapping study. The reason for why it is not completely systematic is that a bachelor's thesis does not require following the guidelines and rules of conducting scientific literature reviews too strictly. However, the thesis aims for some systematicity in the research process since the objective of the thesis is to give an overview of the different container orchestration technologies and to minimize bias in the research and review process.

The process of conducting the research was as follows: First, the research questions were formulated. Second, relevant search terms were picked based on the research questions and they were formulated to an adequate search string. Third, this search string was used to conduct searches to some databases, such as ACM Digital Library, IEEEXplore, Scopus and SpringerLink. After doing the searches to the databases, the metadata of the found articles was collected on one excel-sheet, and the process of removing duplicates and including and excluding papers based on some defined criteria followed. Next step was to collect and synthesize relevant data form the included papers to answer the research questions. The findings from this research are reported in this thesis.

The structure of this thesis will be as follows: In chapter 2 edge computing, containerization and container orchestration are discussed, and an overview of Kubernetes is provided. Chapter 3 will list all found container orchestration technologies developed for edge computing and shortly describe their features and components. In chapter 4 these technologies are compared to each other based on their features and resource utilization and in chapter 5 the results from these comparisons are presented. Chapter 6, conclusion, wraps up the most important findings.

2 Edge computing and container orchestration

There are a couple of concepts that need to be clarified and defined before starting the discussion about container orchestration technologies developed for edge. In this chapter edge computing and its characteristics, as well as containerization and container orchestration, are discussed. At the end of the chapter, an overview of Kubernetes is provided since many of the container orchestration technologies analyzed in this thesis are based on Kubernetes.

2.1 Edge computing

Edge computing specializes cloud computing by moving computing capacity from the centralized datacenters closer to the decentralized edge. For this reason, edge computing has some special characteristics that differentiate it from cloud computing. These characteristics are low latency, geographical distribution, control (Iorga et al. March 2018), security, and privacy. These characteristics are described in Table 1. Bringing computing power, storage, and other services closer to the edge makes edge computing more suitable for localized processing, large-scale sensor networks, mobility support, real-time interactions, heterogeneity, interoperability, and interplay with the centralized cloud (Iorga et al. March 2018).

These characteristics are actualized in many different use-cases of edge computing, and smart factories are one of these use cases (Soori, Arezoo, and Dastres January 2023). In a smart factory there can be a lot of different devices and machines, distributed all over the factory. These devices can be connected to edge servers also located in the factory, near the devices that generate the data. Since these edge servers are located inside the factory and not in a remote data center, the staff of the factory has control over these servers. Moreover, the locality of these edge servers affects the security and privacy inside the factory, since, for example, sensitive data can be shared and processed inside the factory's local network. Lastly, many of the devices used in a smart factory can be time sensitive. This means that they require low latency in processing computational tasks. Some of the data generated by these devices may even have to be processed in real time to, for example, prevent accidents.

Table 1. Characteristics of edge computing

Characteristic	Description
Low Latency	Quicker response time for computation tasks due to the closer processing of data.
Geographical Distribution	Devices in an edge environment are in the close proximity of each other.
Control	The owner of the devices in an edge environment has complete control over these devices.
Security and Privacy	The data is processed in a local network and not sent to a centralized cloud over the Internet.

2.2 Container-based virtualization and container orchestration

There are different virtualization technologies, such as container-based virtualization and hypervisor-based virtualization. Container-based virtualization is the more lightweight and portable alternative of these two virtualization technologies for edge applications (Costa et al. February 2023). Containerization makes the development, testing and deployment of applications running in edge environments easier and faster (Ramalho and Neto June 2016) compared to hypervisor-based virtualization. In container-based virtualization, containers run on a common host kernel which optimizes the use of CPU, memory, and network resources, and makes the containers highly scalable and cost-effective (Watada et al. 2019). Since container-based virtualization is not dependent on any specific type of hardware, containers are an excellent alternative for running applications on the heterogeneous edge devices.

In order to manage multiple containers at the same time, container orchestration is needed. Container orchestration is used, among other things, for scaling containers up or down, migrating containers, allocating resources for containers, and load balancing (Malviya and Dwivedi March 2022). There are several open-source container orchestration platforms such

as Kubernetes, Docker Swarm, and Apache Mesos. However, these container orchestrators do not meet the requirements of edge computing, as was shown in the article by Hoque et al. (July 2017). Edge computing needs container orchestration technologies that take into consideration the requirements of edge computing, such as resource constraints, heterogeneity, and low-latency requirements of the edge devices.

2.3 Kubernetes

Since many of the container orchestration technologies that were found from the analyzed articles are in one way or another related to Kubernetes, this section will give an overview of Kubernetes and its most important components and functionalities. The architecture of Kubernetes is depicted in Figure 1.

Kubernetes is a widely used open-source container orchestration platform (*Overview,*). When Kubernetes is deployed a cluster is created, consisting of a set of nodes (*Kubernetes Components,*). There is at least one master node in a Kubernetes cluster and one to many worker nodes (Carrión December 2022). The master node, or the control plane, consists of the following components: an API server which receives commands for the worker nodes from outside of the cluster, a controller manager which monitors etcd and makes sure that the system is in the desired state (Carrión December 2022), etcd which is a key-value storage for storing all cluster data, and a scheduler which assigns newly created pods to nodes (*Kubernetes Components,*).

Worker nodes in a Kubernetes cluster usually run multiple pods (*Nodes,*). These pods contain the containerized application. According to the Kubernetes documentation about nodes, one of the components of a node is the container runtime, which is a software that is needed for running the containers. According to the documentation other components of a node are the kubelet, which is an agent responsible for making sure that the containers are running in a pod, and the kube-proxy, which is a network proxy.

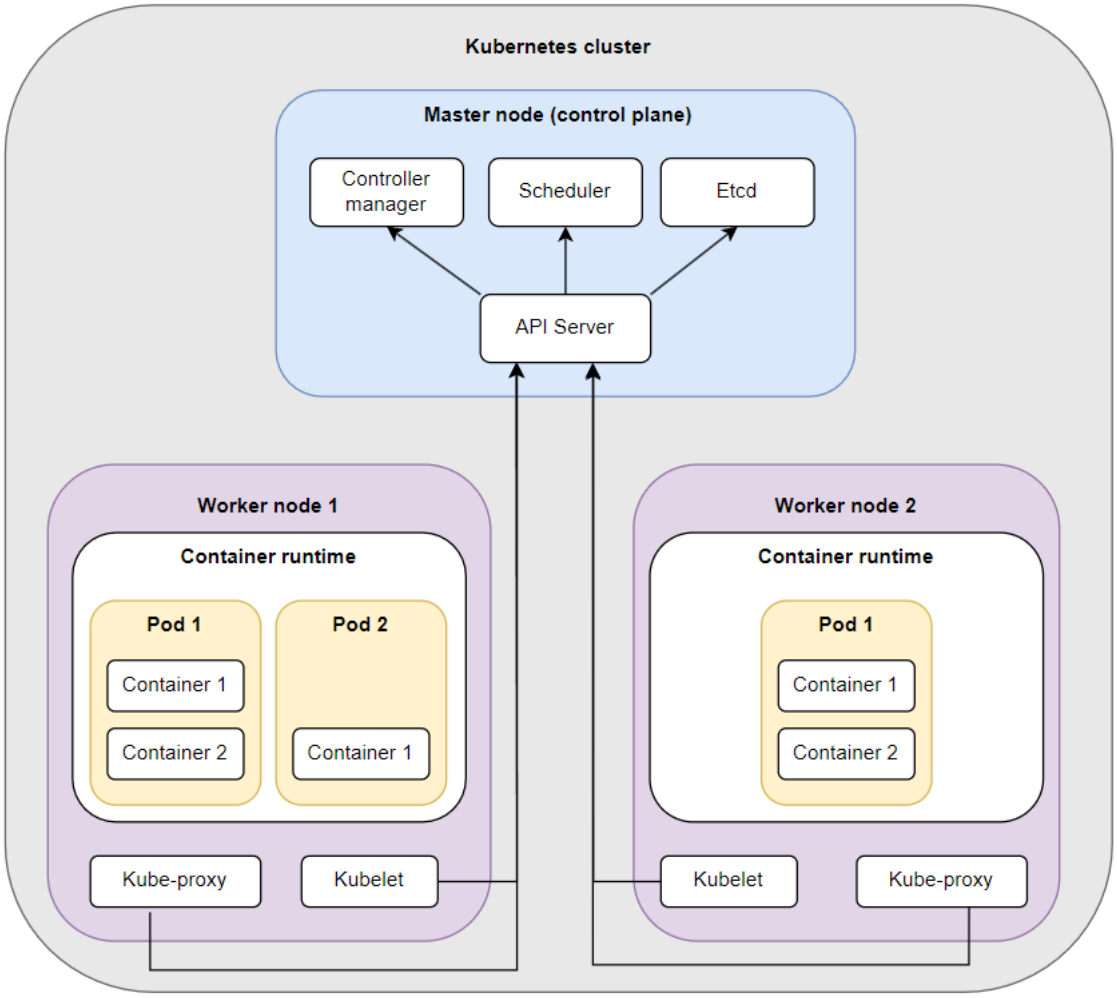

Figure 1. Kubernetes cluster. This figure is based on a figure shown on a YouTube-video by ByteByteGo (November 2023), the figure in Kubernetes documentation about Kubernetes components (*Kubernetes Components,*), and Kubernetes documentation about container runtimes (*Container Runtimes,*).

3 Container orchestration technologies for edge

There are several container orchestration technologies that have been developed with the characteristics of edge computing in mind. This chapter gives a general overview of each of these orchestrators, their architecture, and their features. The found technologies will also be categorized based on how they relate to Kubernetes. The first section will explain how the categorization has been made.

3.1 Categorization of container orchestration technologies

The categories used for categorizing the orchestrators are edge extensions for Kubernetes, Kubernetes distributions, plugins for Kubernetes scheduler, and other container orchestrators. The category of edge extensions for Kubernetes includes KubeEdge and FLEDGE. They both need a Kubernetes master node to connect to and they are both deployed on the worker nodes, except for a couple of components that are placed on the master node to enable communication between the master and the edge nodes. FLEDGE consists of a FLEDGE Agent and KubeEdge consists of an edge core. These both edge components provide services that are needed to enable orchestration of containers in an edge environment.

The category of Kubernetes distributions includes K3s, K0s, Microk8s and MicroShift. These container orchestrators are based on the Kubernetes source code. However, by removing some unimportant components and by adding some edge orchestration enabling components Kubernetes has been modified to be more suitable for edge environments. For example, in K3s, etcd is replaced with sqlite3, which is a more lightweight datastore (*K3s - Lightweight Kubernetes | K3s* April 2024), and a tunnel proxy is added to enable the inclusion of devices without a public IP address to the cluster (Čilić et al. January 2023).

The category of plugins for Kubernetes scheduler includes KubeHICE and REACT. These orchestrators have been developed to solve a specific problem or to add some specific logic to the existing Kubernetes scheduler. They are mainly plugins to the Kubernetes scheduler with some additional components for the master node and the edge nodes to support and enable what the scheduling extension is meant to accomplish.

The last category, other container orchestrators, includes Nomad and RTCO. Nomad and RTCO has been assigned their own category because they are not related to Kubernetes like the other technologies are. This means that they are not, for example, any extension or plugin for Kubernetes, nor are they Kubernetes distributions. However, Nomad has a similar cluster structure as Kubernetes: in Nomad regions (or clusters) consist of servers (or master nodes) and clients (or worker nodes) (*Architecture | Nomad | HashiCorp Developer,*). RTCO, on the other hand, was implemented on Linux in the article by Struhár et al. (January 2024).

3.2 Edge extensions for Kubernetes

One of the container orchestration technologies presented in this category is **KubeEdge**. KubeEdge is an open-source edge computing platform built on Kubernetes, that extends container orchestration and device management to the edge (*Why KubeEdge | KubeEdge* August 2023). According to the documentation, KubeEdge can be used in both edge computing and cloud computing environments. KubeEdge supports both high-availability (*Deploying HA CloudCore | KubeEdge* August 2023) and multi-node clusters (*Why KubeEdge | KubeEdge* August 2023). The CPU architectures supported by KubeEdge are x86, ARMv7, and ARMv8 (*KubeEdge,*) and supported container runtimes are Docker, Containerd, Cri-o, and Virtlet (*Getting Started | KubeEdge* August 2023).

KubeEdge consists of two parts, a cloud component, and an edge component (*Why KubeEdge | KubeEdge* August 2023). According to the documentation, the cloud component manages the edge nodes in the cluster and interacts with the Kubernetes API server. It consists of an Edge Controller and a Device Controller that extend the existing Kubernetes controller (Čilić et al. January 2023). According to the authors these controllers take care of the data synchronization between the cloud and the edge. The authors mention that the cloud part also includes a Cloud Hub, which is responsible for enabling communication between cloud and the edge devices.

The edge component, which is aimed at the edge node, manages the containerized applications that run on the edge nodes and reports status changes to the cloud part (*Why KubeEdge | KubeEdge* August 2023). One of the components of the edge part is Edge Hub, which

is connected to the Cloud Hub mentioned above. It reports edge-side status changes to the cloud and syncs the cloud-side resource updates to edge. Other components of the edge part are Edged, which manages the containerized applications, Meta Manager, which is the processor for messages between Edged and Edge Hub, Device Twin, which stores the device status and syncs it to the cloud, Event Bus, which enables interaction with MQTT servers, and Service Bus, which enables interaction with HTTP servers. All the information in this paragraph is based on the KubeEdge documentation *Why KubeEdge | KubeEdge* (August 2023).

The other container orchestration technology presented in this category is **FLEDGE**. The information about FLEDGE presented in this overview is completely based on the article by Goethals, De Turck, and Volckaert (2020). FLEDGE is a container orchestrator developed for low resource devices at the edge. It uses modified Virtual Kubelet to connect to a Kubernetes cluster and OpenVPN and encryption to ensure secured connection between the edge and the cloud. FLEDGE consists of a Kubernetes master node and a FLEDGE agent which is deployed on every edge device in the cluster. The master node consists of the Virtual Kubelet and a FLEDGE broker server. The Virtual Kubelet acts as a proxy for Kubernetes and passes API calls to the FLEDGE broker server, which in turn sends these calls to the edge.

On an edge node, a FLEDGE agent and a container runtime are deployed. The FLEDGE agent consists of the FLEDGE broker client which receives API calls from the master node, a container runtime interface, a cgroup and namespace manager, and components for container networking and resource monitoring. Since FLEDGE has its own container networking component integrated to the system there is no need for installing a separate container network interface (CNI) plugin for container networking.

With FLEDGE there are no specific requirements regarding container runtime. Goethals, De Turck, and Volckaert (2020) tested both Docker and Containerd with FLEDGE. According to the test results Containerd is a more resource friendly option among the aforementioned container runtimes. In addition, it is not said clearly in the article what CPU architectures FLEDGE supports. However, according to the authors, the evaluations of FLEDGE were run on armfh and x64.

3.3 Kubernetes distributions

The first container orchestration technology presented in this category is **K3s**. K3s is a lightweight Kubernetes distribution packaged as a single binary, and it is half the size compared to Kubernetes in terms of memory requirements (*K3s - Lightweight Kubernetes | K3s* April 2024). According to the documentation K3s is targeted for edge computing environments, IoT, and many more, and it uses Cri-o and Containerd as default container runtimes. K3s supports high-availability and multi-node clusters (*Requirements | K3s* April 2024). In addition, according to the requirements documentation, the CPU architectures supported by K3s are x86_64, armhf, arm64/aarch64, and s390x.

The architecture of K3s is very similar to the architecture of Kubernetes. A K3s control plane consists of the API server, scheduler, controller manager and a control plane datastore (*Architecture | K3s* April 2024). However, one difference between Kubernetes and K3s is that etcd, which is the control plane storage used in Kubernetes, has been replaced with sqlite3 in K3s (*K3s - Lightweight Kubernetes | K3s* April 2024).

The second container orchestrator presented in this category is **K0s**. The information of K0s presented in this paragraph is based on the K0s documentation *Documentation* (). K0s is a Kubernetes distribution that consists of all the important features of Kubernetes to build a Kubernetes cluster. According to the documentation K0s is well suited for running applications in centralized cloud, on bare metal computers, in edge environments and on IoT-devices. K0s is distributed as a single binary, and it can run on any Linux system. K0s supports Containerd as default container runtime, but custom container runtime interface plugins are also supported. K0s also supports high-availability and multi-node clusters. The CPU architectures supported by K0s are x86-64, ARM64, and ARMv7.

Like for K3s, the architecture of K0s does not differ very much from Kubernetes' architecture. In addition to the main Kubernetes control plane components (control plane datastore, API server, scheduler, and controller manager), a K0s control plane also includes a Konnectivity-server, which acts as a proxy that forwards traffic from the API server to the worker nodes (*Networking (CNI) - Documentation,*). The components of the worker nodes include Kubelet and a container runtime (*Architecture - Documentation,*).

The third container orchestration technology presented in this category is **Microk8s**. Microk8s is a lightweight Kubernetes distribution developed for automated deployment, scaling, and management of containerized applications (*MicroK8s - MicroK8s documentation - home | MicroK8s,*). According to Bohm and Wirtz (no date) Microk8s provides only the most important Kubernetes components (the API server, scheduler, and controller manager among other things) making it lightweight and suitable to use for IoT applications. In addition, the authors mention that instead of etcd Microk8s uses Dqlite as its control plane data store.

Microk8s can be used for IoT-applications and in minimal environments (*MicroK8s - MicroK8s documentation - home | MicroK8s,*), such as edge computing. It supports Containerd and Kata container runtimes (*MicroK8s vs k3s vs Minikube | MicroK8s,*), as well as multi-node and high-availability clusters (*MicroK8s - MicroK8s documentation - home | MicroK8s,*). The CPU architectures supported by Microk8s are x86, ARM64, s390x, and POWER9 (*MicroK8s vs k3s vs Minikube | MicroK8s,*).

The last container orchestration technology presented in this category is **MicroShift**. MicroShift is a container orchestration platform based on OpenShift Kubernetes and developed for edge computing environments (*openshift/microshift* April 2024). It consists of only the most important components, and it requires only minimal configuration, making it lightweight and easy to use (*microshift/docs/contributor/design.md at main · openshift/microshift,*). According to the design documentation, MicroShift has a small resource footprint, and it works well with poor or no network connectivity. Microshift does not support multi-node clusters or high-availability clusters (*microshift/docs/contributor/design.md at main · openshift/microshift,*). The CPU architectures supported by MicroShift are amd64/ x86_64, arm64, and riscv64 (*Getting Started,*). According to the documentation, MicroShift requires CRI-O to be installed as a container runtime. There is almost no information available about MicroShift's architecture, and therefore, its architecture will not be presented in this overview.

3.4 Plugins for Kubernetes scheduler

One of the container orchestrators presented in this category is **KubeHICE**. The information about KubeHICE presented in this overview is completely based on an article by Yang et al. (September 2021). KubeHICE is a container orchestrator for devices with heterogeneous Instruction Set Architecture (ISA) in Cloud-Edge platforms, and it can be integrated into the Kubernetes scheduler. The main goals of the KubeHICE orchestrator are to be able to automatically match a node with a specific ISA with a containerized application that supports the ISA of the node, to be compatible with existing container ecosystems, and to provide performance-aware container orchestration for improved utilization of CPU.

KubeHICE is composed of two core methods: Automatic Instruction Set Architecture Matching (AIM) and Performance Aware Scheduling (PAS). AIM is responsible for finding a node with a suitable ISA for a container whereas PAS is responsible for scheduling containers according to the computing capability of the nodes. AIM and PAS are deployed on the Kubernetes master node and plugged into the Kubernetes scheduler. A Monitor Client running on each node reports the CPU usage of the containers to the Monitor Server which stores this data to an external database. The Performance Analyzer running on the master node then reads this data form the database to make estimations of the performance of each node.

The other container orchestrator presented in this category is **REACT**. The information about REACT presented in this overview is completely based on an article by Struhar et al. (September 2021). REACT is a container orchestration extension for Kubernetes, developed for orchestrating containerized applications with real-time (RT) requirements. The article does not directly mention any name for the orchestrator but since "REACT" is mentioned in the title of the article, it will be used in this thesis to address the orchestrator in question. Furthermore, it is not stated clearly in the article that REACT would be targeted specifically for edge computing. According to the article, REACT supports multi-node and high-availability clusters.

REACT provides these following functionalities: placement of real-time containers on suitable nodes according to the containers' real-time requirements, and continuous monitoring of the resource usage of the containers. This information about resource usage is then con-

sidered in the next scheduling decision. The REACT extension consists of a RT Scheduler Extender on the master node, and a RT Manager on the worker nodes. The RT Scheduler Extender is an additional scheduler that performs a secondary filtering of feasible nodes for the real-time containers that is performed after Kubernetes' default scheduling process. The RT Manager deploys the containers onto the worker nodes and reports their real-time performance to the master node.

3.5 Other container orchestrators

One of the container orchestration technologies presented in this category is **Nomad**. Nomad is a simple, flexible, and scalable workload orchestrator for deploying and managing containerized, non-containerized, microservice based and batch applications (*Introduction | Nomad | HashiCorp Developer,*). According to the documentation it runs as a single binary with no external services needed for deployment. Nomad supports high-availability (*Nomad reference architecture | Nomad | HashiCorp Developer,*) and multi-node clusters (*Introduction | Nomad | HashiCorp Developer,*).

In this paragraph the architecture of Nomad is described. The information in this paragraph is completely based on the architecture documentation *Architecture | Nomad | HashiCorp Developer* () for Nomad. When Nomad is deployed a region is created, consisting of clients and servers. Servers are responsible for receiving and accepting jobs from users, managing the clients, and making decisions about task placements in the region. A job consists of a set of tasks that should be run. A region consists of multiple servers among which a leader server is elected. This leader server is responsible for processing all queries to the region and for coordinating the scheduling being carried out by the servers in parallel. Clients communicate with their regional servers using remote procedure calls to register themselves and to report their status and the status of the allocations to the servers.

With Nomad multiple regions can be connected to form a single big cluster, to ensure availability or scalability (*Architecture | Nomad | HashiCorp Developer,*). According to the architecture documentation regions do not share jobs, clients, or state, but users can submit jobs or query the state of any region in the cluster through a gossip protocol.

The other container orchestration technology presented in this category is a real-time container orchestration framework developed by Struhár et al. (January 2024). The authors propose a real-time container virtualization and orchestration framework (**RTCO**) which was developed to support real-time containers in a multi-container environment. According to the article the orchestrator consists of two phases: The offline phase is responsible for making an initial reservation of resources for containers and selecting a node for the containers. The online phase, on the other hand, is responsible for distributing CPU resources among real-time and regular containers by continuously evaluating the real-time performance of the containers. According to the article, the system model of the orchestrator is based on the REACT orchestrator (see section 3.4).

In the article it is not clearly said what container runtime can be used with RTCO. However, in the experiment conducted in the article Docker containers were used. In addition to container runtime, it is also not said clearly whether the orchestrator has been developed specifically for edge computing.

4 Comparison of container orchestration technologies

In this chapter the container orchestration technologies presented in the previous chapter will be compared against each other based on their features and their resource utilization. However, since the orchestration technologies are very different from each other, they are not directly comparable.

4.1 Feature comparison

In this section the found orchestration technologies will be compared against each other in terms of their features. The feature comparison of these container orchestration technologies is depicted in Table 2. The "Edge" column shows which orchestrators are developed for edge computing environments. The "HA", which stands for high availability, and the "Multi-node" columns show which container orchestrators support high-availability and multi-node clusters respectively. The "Heterogeneity" column shows all the CPU architectures each technology supports. The "Real-time" column shows which technologies support real-time containers. The last column ("ISA matching") shows which technologies provide automatic Instruction Set Architecture (ISA) matching between containers and worker nodes. The abbreviation “n/a” used in the table stands for “not available/applicable” and is used in case the information about a specific feature for a technology is not available.

Table 2 shows that almost all the listed container orchestration technologies have been developed specifically for edge computing. The only exceptions are REACT, RTCO and Nomad. For Nomad the documentation does not say that Nomad would have been developed specifically for edge computing. For REACT and RTCO the articles do mention edge computing but whether the orchestrators in question were developed for edge is not stated directly.

In the context of these container orchestration technologies, high-availability means that there are more than one node in the cluster and the control plane is deployed on more than one node (*MicroK8s - High Availability (HA) | MicroK8s,*). According to the documentation of Microk8s, high availability allows for the cluster to function and continue serving workloads even if one node in the cluster would go down. The container orchestration technologies that

Table 2. Feature comparison of container orchestration technologies developed for edge. The meanings of the abbreviations are following: HA stands for high availability of the master node, ISA stands for instruction set architecture, and n/a stands for "not available/applicable".

Technology	Edge	HA	Multi-node	Heterogeneity	Real-time	ISA matching
KubeEdge	yes	yes	yes	x86, ARMv7, ARMv8	n/a	n/a
FLEDGE	yes	n/a	yes	n/a (armhf and x64 were used in the evaluation)	n/a	n/a
K3s	yes	yes	yes	x86_64, armhf, arm64/aarch64, s390x	n/a	n/a
K0s	yes	yes	yes	x86-64, ARM64, ARMv7	n/a	n/a
Microk8s	yes	yes	yes	x86, ARM64, s390x, POWER9	n/a	n/a
Microshift	yes	no	no	amd64/x86_64, arm64, riscv64	n/a	n/a
KubeHICE	yes	n/a	yes	same as Kubernetes	no	yes
REACT	n/a	n/a	yes	same as Kubernetes	yes	no
Nomad	no	yes	yes	n/a	n/a	n/a
RTCO	n/a	n/a	yes	n/a	yes	no

do support high-availability clusters are KubeEdge, K3s, K0s, MicroShift, and Nomad. For FLEDGE, KubeHICE, REACT, and RTCO it is usure whether they support high-availability clusters or not. MicroShift on the other hand does not support high-availability clusters since it does not support multi-node clusters either. KubeEdge, K3s, K0s, Microk8s, REACT, and Nomad, on the other hand, do support multi-node clusters. FLEDGE (Goethals, De Turck, and Volckaert 2020), KubeHICE (Yang et al. September 2021), and RTCO (Struhár et al. January 2024) support multi-node clusters, too.

KubeEdge, K3s, K0s, Microk8s, and MicroShift run on both x86 and ARM machines. In addition, K3s and Microk8s run on s390x machines. Microk8s also supports the POWER9 architecture and MicroShift supports the riscv64 architecture. For Nomad and RTCO it is unknown what CPU architectures they support. In the articles that present KubeHICE and REACT it is not said directly what architectures they support either. However, since they are extensions for the Kubernetes scheduler, they can be assumed to support the same CPU architectures as Kubernetes.

Real-time containers are supported only by REACT and RTCO. This is because these technologies were developed specifically to solve the problem of scheduling containers with real-time requirements to worker nodes. For all the other container orchestrators it is unsure whether they support real-time containers or not, except for KubeHICE, which does not support real-time containers. This is because it is an extension for the Kubernetes scheduler, like REACT and RTCO, that was developed to solve the problem about automatic ISA matching between a containerized application and a worker node (see section 3.4).

Based on this feature comparison the best container orchestrators to use in edge computing are K3s and Microk8s, since they both support multi-node and high-availability clusters, and they both support four different CPU architectures. The support for multi-node clusters and different CPU architectures are especially important for a container orchestration technology that is meant to be used in edge environments. This is because edge environments usually consist of multiple edge devices that often have very heterogeneous hardware. Based on the feature comparison KubeEdge and K0s are also good options for orchestration at edge environments since they also support many of the features listed in Table 2.

KubeHICE is well suited for edge computing environments since it was developed to make Kubernetes more suitable for edge computing. In addition, REACT and RTCO are also well suited for edge computing environments since many edge devices require real-time processing of data. Furthermore, since KubeHICE and REACT are plugins for the Kubernetes scheduler they could be used with some of the other container orchestration technologies, such as K3s, K0s, Microk8s, MicroShift, FLEDGE, and KubeEdge, to improve their feasibility for edge environments. However, this is not possible in practice since KubeHICE and REACT are experimental projects and they are not available for use.

MicroShift, on the other hand, is not very well suited for edge computing since it does not support multi-node clusters. Furthermore, it does not support high-availability clusters either. According to the MicroShift documentation *microshift/docs/contributor/design.md at main · openshift/microshift* () high-availability of the control plane increases the complexity of deployment and upgrade processes, which conflicts with MicroShift's design principles about minimal configuration and ease-of-use. Based on this feature comparison it is hard to say for FLEDGE and Nomad how well-suited they are for edge computing. For Nomad the reason is that it is not specifically developed for edge computing. For FLEDGE, on the other hand, the reason is that a lot of important information about its features is not available as can be seen in Table 2.

4.2 Resource utilization comparison

In this section the found orchestration technologies will be compared against each other in terms of their resource utilization. KubeHICE, REACT, and RTCO will not be included in this comparison since there is no information available about their resource utilization.

For this comparison five articles have been analyzed. Bahy et al. (September 2023), Bohm and Wirtz (no date), and Telenyk et al. (September 2021) tested CPU, memory, and storage utilization of the found container orchestrators. However, Koziolek and Eskandani (April 2023) tested only CPU and memory utilization. Goethals, De Turck, and Volckaert (2020) compared FLEDGE to Kubernetes and K3s in terms of memory and storage requirements. Furthermore, some of the found container orchestrators have been researched more than

others. For instance, K3s has been analyzed in all articles whereas KubeEdge, Nomad, K0s, FLEDGE, and MicroShift have been analyzed in only one article.

In all these articles the technologies have been tested in different test scenarios. Bahy et al. (September 2023) tested Nomad, K3s, and KubeEdge in three different test scenarios: when the node was created, when the orchestrator was deployed onto the node, and when an application was deployed. Koziolek and Eskandani (April 2023), on the other hand, analyzed Microk8s, K3s, K0s, and MicroShift in the idle condition of the orchestrator. The authors conducted the experiment both for the control plane and the worker nodes for each orchestrator. Lastly, Bohm and Wirtz (no date) and Telenyk et al. (September 2021) both tested and compared K3s, Microk8s and Kubernetes against each other. Bohm and Wirtz (no date) tested the resource utilization of each technology during starting, adding, running, draining, and stopping of nodes. In addition, the test scenarios also included applying, running, and deleting a web server deployment. Telenyk et al. (September 2021), on the other hand, tested the technologies in the scenarios of starting and stopping the master node, adding and deleting a worker node, applying and removing a deployment, and in idle cluster state. Goethals, De Turck, and Volckaert (2020) compared FLEDGE to Kubernetes when FLEDGE was running with kube-proxy and FLEDGE to K3s when FLEDGE was not running with kube-proxy.

In the CPU utilization tests there is no clear winner among the found container orchestrators developed for edge computing. Bahy et al. (September 2023) report that Nomad performed best in the CPU utilization tests. However, the authors report that K3s performed only slightly worse that Nomad. On the other hand, Koziolek and Eskandani (April 2023) report that K0s performed significantly better in the CPU utilization tests compared to the other technologies analyzed in the article. Bohm and Wirtz (no date) and Telenyk et al. (September 2021) compared K3s and Microk8s against Kubernetes, and for both articles the authors report that Kubernetes was most efficient in CPU utilization. However, according to Bohm and Wirtz (no date), there was only small difference in the CPU utilization between K3s and Kubernetes.

Bahy et al. (September 2023) report that KubeEdge performed worst in the CPU utilization test. According to the authors, KubeEdge performed significantly worse compared to Nomad

and K3s. Telenyk et al. (September 2021) report that Microk8s and K3s showed worst CPU utilization on average. According to the authors, in some test scenarios K3s performed better and in some scenarios Microk8s performed better. In addition, in some cases K3s is reported even to have performed better than Kubernetes. Koziolek and Eskandani (April 2023) report that K3s and Microk8s performed worst in the CPU utilization tests. According to the authors, there were not very big differences in the CPU utilization between Microk8s, K3s, and MicroShift. Bohm and Wirtz (no date) report that Microk8s performed significantly worse in the CPU utilization tests than Kubernetes and K3s. These results indicate that Microk8s does not utilize CPU resources very efficiently.

In the memory utilization tests there is no clear winner among the edge container orchestrators. Both Bohm and Wirtz (no date) and Telenyk et al. (September 2021) report that Kubernetes performed best in the memory utilization tests. However, according to Bohm and Wirtz (no date) the difference between Kubernetes' and K3s' memory utilization was very small. Bahy et al. (September 2023) report that Nomad performed best in the memory utilization tests on average. According to the authors there were not very big differences between the technologies in the memory utilization tests on the ARM and x86 worker nodes. On the master node, on the other hand, the authors report that Nomad performed significantly better compared to KubeEdge and K3s. Lastly, Koziolek and Eskandani (April 2023) report that MicroShift and K3s performed best in the memory utilization tests: MicroShift performed best for the master node and K3s performed best for the worker node. Goethals, De Turck, and Volckaert (2020) report that FLEDGE performed significantly better in the memory usage test compared to Kubernetes and K3s.

Koziolek and Eskandani (April 2023) report that Microk8s performed worst in the memory utilization tests for the master node, whereas K0s performed worst for the worker node. According to Bohm and Wirtz (no date) Microk8s performed significantly worse in the memory utilization tests compared to K3s and Kubernetes. According to Telenyk et al. (September 2021) both Microk8s and K3s performed worst in the memory utilization tests. According to the authors, in some testing scenarios Microk8s performed worst and in some scenarios K3s performed worst. Bahy et al. (September 2023) report that KubeEdge had worst memory utilization. However, according to the authors, the difference between the technologies in

memory utilization was quite small for the worker nodes. For the master node the difference between KubeEdge and Nomad was significant.

Bahy et al. (September 2023), Bohm and Wirtz (no date), and Telenyk et al. (September 2021) report that K3s performed best in the storage utilization tests. According to Telenyk et al. (September 2021) K3s performed significantly better in the storage utilization test compared to Kubernetes and Microk8s in all test scenarios. According to the authors the reason for this may be because K3s uses Sqlite3 instead of Kubernetes' default etcd. However, according to Goethals, De Turck, and Volckaert (2020) FLEDGE performed better compared to K3s in the storage usage test.

Bahy et al. (September 2023) report that KubeEdge had the worst storage utilization. Bohm and Wirtz (no date) and Telenyk et al. (September 2021), on the other hand, report that Microk8s performed worst in the storage utilization tests on average. In the storage utilization tests the authors for both of these articles report that Microk8s performed significantly worse than the other technologies analyzed in these articles.

Based on this resource utilization comparison there is no one clear winner among the analyzed container orchestration technologies. When comparing the results of the resource utilization tests for K3s between all four analyzed articles, the results vary. However, one constant result is that Bahy et al. (September 2023), Bohm and Wirtz (no date), and Telenyk et al. (September 2021) all report that K3s performed best in the storage utilization tests in all test scenarios. According to Goethals, De Turck, and Volckaert (2020), however, FLEDGE performed better compared to K3s in both memory and storage usage tests. Based on this resource utilization comparison Microk8s does not utilize resources very efficiently. Since the remaining technologies (K0s, MicroShift, Nomad, and KubeEdge) have all been analyzed in only one article, and some of them even in different articles, it is hard to say how efficient they are in resource utilization compared to each other but also compared to K3s and Microk8s.

5 Results

Based on the feature comparison and the resource utilization comparison of the container orchestration technologies K3s is a very well-suited container orchestration platform for edge computing environments. There are many reasons for this. First, K3s supports both multiple different CPU architectures and multi-node clusters. Second, K3s uses storage resources very efficiently compared to the other orchestrators, which was discovered in the resource utilization comparison (see section 4.2. Third, the results of the resource utilization tests indicate that K3s is on average fairly efficient in utilizing CPU and memory, too.

Based on the feature comparison and resource utilization comparison Microk8s and MicroShift are less well-suited for edge computing environments. This is because MicroShift does not support multi-node clusters and based on the results of the resource utilization comparison Microk8s does not utilize resources very efficiently. KubeHICE, REACT, and RTCO, on the other hand, are not even options when it comes to deciding what container orchestrator to use at the edge, since they are experimental orchestrators and not available for use. FLEDGE, on the other hand, is not only an experimental orchestrator. Based on the results from resource utlization comparison and feature comparison it could be considered as a container orchestrator well-suited for edge environments. However, it cannot be used for container orchestration at the edge since it is not available either. For Nomad, KubeEdge, and K0s, on the other hand, it is hard to say how well-suited they are for edge environments. The most important reason for this is that Nomad, K0s, and KubeEdge have been analyzed in only one article and therefore no conclusions can be drawn about how well they utilize resources in comparison to the other orchestrators.

The container orchestration technologies that were presented and analyzed in this thesis are different from each other in many ways. In addition, in the articles that were analyzed in section 4.2 some of the technologies were analyzed in more than one article, whereas many of them were analyzed in only one of the articles. These are the reasons why the feature and resource utilization comparisons between the container orchestration technologies analyzed in this thesis are not fair. Another thing that can affect the results in the resource utilization comparison is that not all of the technologies were tested in an actual edge computing en-

vironment. For instance, in the article by Bohm and Wirtz (no date) the experimental setup consisted of one master node and three worker nodes, all virtual machines hosted on one single physical machine.

6 Conclusion

In this thesis ten container orchestration technologies were presented and categorized based on how they relate to Kubernetes. These technologies were also analyzed based on their features and resource utilization. Based on the comparison results the technologies' feasibility for edge computing environments was discussed. The results indicate that K3s is well suited for edge computing environments. On the other hand, Microk8s and MicroShift are less well-suited for edge computing. This is because Microk8s is not very resource efficient and MicroShift does not support multi-node clusters. KubeHICE, REACT, FLEDGE, and RTCO, on the other hand, are not available for use.

The container orchestrators analyzed in this thesis are different from each other. In addition, some of them have been researched more than others, and some of them have not been tested in an actual edge computing environment. For these reasons the feature comparison and the resource utilization comparison cannot give an absolutely fair comparison between these technologies or a completely realistic idea about how well these technologies are suited for edge computing environments.

Bibliography

Architecture - Documentation. Visited on May 4, 2024. https://docs.k0sproject.io/stable/architecture/#worker-node.

Architecture | K3s [**inlang**en]. April 2024. Visited on April 12, 2024. https://docs.k3s.io/architecture.

Architecture | Nomad | HashiCorp Developer [**inlang**en]. Visited on April 27, 2024. https://developer.hashicorp.com/nomad/docs/concepts/architecture.

Bahy, Muhammad Bintang, Nur Rahmat Dwi Riyanto, Muhammad Zain Fawwaz Nuruddin Siswantoro, and Bagus Jati Santoso. September 2023. "Resource Utilization Comparison of KubeEdge, K3s, and Nomad for Edge Computing" [**inlang**en]. In *2023 10th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI),* 321–327. Palembang, Indonesia: IEEE. ISBN: 9798350306866, visited on March 21, 2024. https://doi.org/10.1109/EECSI59885.2023.10295642. https://ieeexplore.ieee.org/document/10295642/.

Bohm, Sebastian, and Guido Wirtz. No date. "Profiling Lightweight Container Platforms: MicroK8s and K3s in Comparison to Kubernetes" [**inlang**en].

ByteByteGo. November 2023. *Kubernetes Explained in 6 Minutes | k8s Architecture.* Visited on May 6, 2024. https://www.youtube.com/watch?v=TlHvYWVUZyc.

Carrión, Carmen. December 2022. "Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges". *ACM Computing Surveys* 55, number 7 (): 138:1–138:37. ISSN: 0360-0300, visited on April 8, 2024. https://doi.org/10.1145/3539606.

Čilić, Ivan, Petar Krivić, Ivana Podnar Žarko, and Mario Kušek. January 2023. "Performance Evaluation of Container Orchestration Tools in Edge Computing Environments" [**inlang**en]. Number: 8 Publisher: Multidisciplinary Digital Publishing Institute, *Sensors* 23, number 8 (): 4008. ISSN: 1424-8220, visited on March 12, 2024. https://doi.org/10.3390/s23084008. https://www.mdpi.com/1424-8220/23/8/4008.

Container Runtimes [**inlang**en]. Section: docs. Visited on May 6, 2024. https://kubernetes.io/docs/setup/production-environment/container-runtimes/.

Costa, Breno, Joao Bachiega, Leonardo Rebouças De Carvalho, and Aleteia P. F. Araujo. February 2023. "Orchestration in Fog Computing: A Comprehensive Survey" [**inlang**en]. *ACM Computing Surveys* 55, number 2 (): 1–34. ISSN: 0360-0300, 1557-7341, visited on January 22, 2024. https://doi.org/10.1145/3486221.

Deploying HA CloudCore | KubeEdge [**inlang**en-GB]. August 2023. Visited on April 27, 2024. https://kubeedge.io/docs/setup/deploy-ha/.

Documentation. Visited on April 12, 2024. https://docs.k0sproject.io/stable/.

Getting Started | KubeEdge [**inlang**en-GB]. August 2023. Visited on April 17, 2024. https://kubeedge.io/docs/welcome/getting-started/.

microshift/docs/contributor/design.md at main · openshift/microshift [**inlang**en]. Visited on May 2, 2024. https://github.com/openshift/microshift/blob/main/docs/contributor/design.md.

Goethals, Tom, Filip De Turck, and Bruno Volckaert. 2020. "FLEDGE: Kubernetes Compatible Container Orchestration on Low-Resource Edge Devices" [**inlang**en]. In *Internet of Vehicles. Technologies and Services Toward Smart Cities,* edited by Ching-Hsien Hsu, Sondès Kallel, Kun-Chan Lan, and Zibin Zheng, 174–189. Lecture Notes in Computer Science. Cham: Springer International Publishing. ISBN: 978-3-030-38651-1. https://doi.org/10.1007/978-3-030-38651-1_16.

Hoque, Saiful, Mathias Santos De Brito, Alexander Willner, Oliver Keil, and Thomas Magedanz. July 2017. "Towards Container Orchestration in Fog Computing Infrastructures". In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC),* 2:294–299. ISSN: 0730-3157. Visited on April 27, 2024. https://doi.org/10.1109/COMPSAC.2017.248. https://ieeexplore.ieee.org/abstract/document/8029944.

Introduction | Nomad | HashiCorp Developer [**inlang**en]. Visited on April 27, 2024. https://developer.hashicorp.com/nomad/intro.

Iorga, Michaela, Larry Feldman, Robert Barton, Michael J Martin, Ned Goren, and Charif Mahmoudi. March 2018. *Fog computing conceptual model* [**inlang**en]. Technical report NIST SP 500-325. Gaithersburg, MD: National Institute of Standards and Technology. Visited on March 12, 2024. https://doi.org/10.6028/NIST.SP.500-325.

K3s - Lightweight Kubernetes | K3s [**inlang**en]. April 2024. Visited on April 12, 2024. https://docs.k3s.io/.

Khan, Wazir Zada, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. August 2019. "Edge computing: A survey". *Future Generation Computer Systems* 97 (): 219–235. ISSN: 0167-739X, visited on March 1, 2024. https://doi.org/10.1016/j.future.2019.02.050.

Koziolek, Heiko, and Nafise Eskandani. April 2023. "Lightweight Kubernetes Distributions: A Performance Comparison of MicroK8s, k3s, k0s, and Microshift" [**inlang**en]. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering,* 17–29. Coimbra Portugal: ACM. ISBN: 9798400700682, visited on March 12, 2024. https://doi.org/10.1145/3578244.3583737.

KubeEdge [**inlang**en-GB]. Visited on April 27, 2024. https://kubeedge.io/.

Kubernetes Components [**inlang**en]. Section: docs. Visited on April 8, 2024. https://kubernetes.io/docs/concepts/overview/components/.

Malviya, Anshita, and Rajendra Kumar Dwivedi. March 2022. "A Comparative Analysis of Container Orchestration Tools in Cloud Computing". In *2022 9th International Conference on Computing for Sustainable Global Development (INDIACom),* 698–703. Visited on April 8, 2024. https://doi.org/10.23919/INDIACom54597.2022.9763171.

MicroK8s - High Availability (HA) | MicroK8s [**inlang**en]. Visited on May 4, 2024. http://microk8s.io.

MicroK8s - MicroK8s documentation - home | MicroK8s [**inlang**en]. Visited on April 12, 2024. http://microk8s.io.

MicroK8s vs k3s vs Minikube | MicroK8s [**inlang**en]. Visited on April 12, 2024. http://microk8s.io.

Getting Started [**inlang**en]. Visited on May 2, 2024. https://microshift.io/docs/getting-started/.

Networking (CNI) - Documentation. Visited on April 12, 2024. https://docs.k0sproject.io/stable/networking/.

Nodes [**inlang**en]. Section: docs. Visited on April 8, 2024. https://kubernetes.io/docs/concepts/architecture/nodes/.

Nomad reference architecture | Nomad | HashiCorp Developer [**inlang**en]. Visited on April 27, 2024. https://developer.hashicorp.com/nomad/tutorials/enterprise/production-reference-architecture-vm-with-consul.

openshift/microshift. April 2024. Original-date: 2021-04-26T14:39:38Z. Visited on April 27, 2024. https://github.com/openshift/microshift.

Overview [**inlang**en]. Visited on April 8, 2024. https://kubernetes.io/docs/concepts/overview/.

Ramalho, Flávio, and Augusto Neto. June 2016. "Virtualization at the network edge: A performance comparison". In *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM),* 1–6. Visited on March 12, 2024. https://doi.org/10.1109/WoWMoM.2016.7523584. https://ieeexplore.ieee.org/abstract/document/7523584.

Requirements | K3s [**inlang**en]. April 2024. Visited on April 27, 2024. https://docs.k3s.io/installation/requirements.

Soori, Mohsen, Behrooz Arezoo, and Roza Dastres. January 2023. "Internet of things for smart factories in industry 4.0, a review". *Internet of Things and Cyber-Physical Systems* 3 (): 192–204. ISSN: 2667-3452, visited on May 6, 2024. https://doi.org/10.1016/j.iotcps.2023.04.006.

Struhar, Vaclav, Silviu S. Craciunas, Mohammad Ashjaei, Moris Behnam, and Alessandro V. Papadopoulos. September 2021. “REACT: Enabling Real-Time Container Orchestration” [**inlang**en]. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA),* 1–8. Vasteras, Sweden: IEEE. ISBN: 978-1-72812-989-1, visited on March 18, 2024. https://doi.org/10.1109/ETFA45728.2021.9613685.

Struhár, Václav, Silviu S. Craciunas, Mohammad Ashjaei, Moris Behnam, and Alessandro V. Papadopoulos. January 2024. “Hierarchical Resource Orchestration Framework for Real-time Containers”. *ACM Transactions on Embedded Computing Systems* 23, number 1 (): 4:1–4:24. ISSN: 1539-9087, visited on March 12, 2024. https://doi.org/10.1145/3592856.

Telenyk, Sergii, Oleksii Sopov, Eduard Zharikov, and Grzegorz Nowakowski. September 2021. “A Comparison of Kubernetes and Kubernetes-Compatible Platforms” [**inlang**en]. In *2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS),* 313–317. Cracow, Poland: IEEE. ISBN: 978-1-66544-209-1 978-1-66542-605-3, visited on April 10, 2024. https://doi.org/10.1109/IDAACS53288.2021.9660392.

Watada, Junzo, Arunava Roy, Ruturaj Kadikar, Hoang Pham, and Bing Xu. 2019. “Emerging Trends, Techniques and Open Issues of Containerization: A Review” [**inlang**en]. *IEEE Access* 7:152443–152472. ISSN: 2169-3536, visited on March 12, 2024. https://doi.org/10.1109/ACCESS.2019.2945930.

Why KubeEdge | KubeEdge [**inlang**en-GB]. August 2023. Visited on April 12, 2024. https://kubeedge.io/docs/.

Yang, Saqing, Yi Ren, Jianfeng Zhang, Jianbo Guan, and Bao Li. September 2021. “Kube-HICE: Performance-aware Container Orchestration on Heterogeneous-ISA Architectures in Cloud-Edge Platforms” [**inlang**en]. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom),* 81–91. New York City, NY, USA: IEEE. ISBN: 978-1-66543-574-1, visited on March 18, 2024. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00025.