Cloud-Native Workload Orchestration at the Edge

Subjects: Computer Science, Software Engineering

Contributor: Rafael Vaño , Ignacio Lacalle , Piotr Sowiński , Raúl S-Julián , Carlos E. Palau

Cloud-native computing principles such as virtualization and orchestration are key to transferring to the promising paradigm of edge computing. Challenges of containerization, operative models and scarce availability of established tools make a thorough review indispensable. Container virtualization and its orchestration through Kubernetes have dominated the cloud computing domain, while major efforts have been recently recorded focused on the adaptation of these technologies to the edge. Such initiatives have addressed either the reduction of container engines and the development of specific tailored operating systems or the development of smaller K8s distributions and edge-focused adaptations (such as KubeEdge). Finally, new workload virtualization approaches, such asWebAssembly modules together with the joint orchestration of these heterogeneous workloads, seem to be the topics to pay attention to in the short to medium term.

edge computing cloud computing edge-to-cloud computing continuum edge-native Kubernetes WebAssembly container container runtime microVM Unikernel

1. Introduction

Edge-native as a concept was first introduced by Satyanarayanan et al. in 2019 [1]. After that, it has been used in multiple fora by companies such as IBM, Cisco, and others, settling as the most popular term to refer to the application of cloud-native design principles on edge computing devices. These cornerstone principles are microservices-centricity (achieving modular distributed system deployment), containerization, dynamic management, orchestration and scheduling, and DevOps [2].

There is a growing interest in research on transferring technologies and tools to the edge to comply with such principles, applying agility, adaptability, affordability, and accessibility traits to edge scenarios. This interest is mostly manifested by the launching of so-called projects, particularly those promoted by renowned open-source initiatives like Linux Foundation (LF Edge line [3]) or Eclipse Foundation (Edge Native Working Group [4]). Especially relevant is here the Cloud Native Computing Foundation (CNCF), the vendor-neutral hub that has standardized and fostered cloud-native principles and technologies like Kubernetes, *etcd*, CoreDNS, gRPC, and many others. CNCF has launched a series of incubated and sandbox projects under their umbrella specifically pursuing edge-native goals [5]. In addition, ad hoc funding tenders have been scheduled by public entities such as the European Commission (EC) [6], such as the development of meta operating systems (meta-OS) to be installed in edge computing nodes [7], or distributed smart orchestration of various workloads across the edge and cloud computing fabric using Artificial Intelligence (AI) [8]. The former aims at realizing the so-called IoT-edge-cloud computing continuum, which roots its success on applying cloud-native techniques along the entire computational spectrum. Following the open-source spirit of Eclipse and CNCF, the EC has launched an ambitious initiative: EUCloudEdgeIoT.eu, that aims to bring together all European research actions seeking to devise such computing continuums with a common stack of open-source technologies [9].

2. Virtualization: From Cloud Computing to Constrained Environments

One of the main keystones behind the success of cloud computing is the capacity to virtualize the available resources and the ability to manage workloads (services/applications) in an efficient way over virtualized hardware [10]. Virtual Machines (VMs) were the first attempt, enabling the creation of smaller instances of machines over the same physical infrastructure. This paradigm dominated the cloud computing arena for several years until new approaches appeared aiming to solve several shortcomings of VMs. First, the overhead created by the need to install multiple OSs to support different VMs was hindering the potential to devote those resources to actual workloads. In addition, the workloads became very diverse and heterogeneous (video streaming, sensor data processing, AI…), requiring dynamicity in the installation and management, something that VM hypervisors were not able to provide [11]. Furthermore, the utilization of hypervisors required system owners and developers to be aware of the needed resources and the infrastructure to be selected to deploy their services or applications. Several initiatives appeared with the aim of overcoming those issues, having major success in the cloud computing field.

To solve the latter problem (required knowledge of underlying hardware and resources), a paradigm arose called *serverless*. *Serverless* architectures are widely adopted by public cloud providers, offering their customers the possibility of running their applications without having to take care of: (1) the infrastructure where it will be really deployed and (2) for how long to reserve it. This paradigm has been especially successful in the public cloud as it builds upon the assumption that the available computing resources are well identified (e.g., as in datacenters) and that the infrastructure is easily manageable and accessible in a uniform way by a single operator (cloud provider). However, these principles are exactly orthogonal to the reality of edge computing, where a software architect cannot omit the insights regarding the infrastructure when an application’s deployment must take into account the underlying heterogeneity of resources. Notwithstanding, some initiatives for applying *serverless* mechanisms to the edge have arisen. Especially relevant are those that rely on the temporal dynamicity of services deployment in order to downscale or upscale on demand the resources devoted to services, optimizing resource consumption (which is of paramount importance on the edge). Examples following this approach are projects such as OpenFaaS [12] and Knative [13], which take advantage of the Kubernetes technology. Knative (which is an incubating project of the CNCF) also provides a complete event-driven engine based on CloudEvents, a specification to standardize event data descriptions, which opens a wide range of possibilities in K8s-based architectures on the edge [14]. With the inclusion of this event-driven engine, the deployed services can trigger different events to activate some functionalities or workloads without modifying the source code.

To solve the former problem (overheads caused by installing one OS per each virtual machine), a paradigm arose called container virtualization. Container virtualization implies the use of containers to isolate applications and their dependencies into smaller units that contain the service to be deployed. The ruling principle of container virtualization is that no hypervisor (plus OS per machine) is needed; in contrast, the host's single OS is virtualized, allowing containers to be deployed isolated, but still sharing the same kernel [15]. This makes containers much more lightweight and scalable. Because of all these advantages, containers have become the de facto standard for running virtualized deployments in the cloud (cloud-native) and also are the natural evolution of the legacy VM-based deployments [16]. Several container technologies exist (e.g., LXC [17]—LinuX Containers—was one of the first developed container runtimes), but since Docker was released in 2013, this technology has been the standard solution for this virtualization technique. Docker has been successfully tested and adopted in production-grade deployments around the industry's public and private clouds for different purposes.

It should be noted that the area of systems based on cloud-native computing principles is evolving rapidly to meet the shifting demands of the industry. Current challenges and open issues include unified orchestration for heterogeneous systems, federated management domains, leveraging artificial intelligence for automatic resource management and data interoperability [18][19]. Interestingly, the surveys also acknowledge that one of the main challenges is applying the existing cloud-native principles to the entire edge-to-cloud continuum.

3. Current Commercial Approaches from Public Cloud Providers

Most public cloud providers have addressed edge computing as an extension of their own cloud infrastructure scope. Consequently, the majority of available solutions from those companies consist of their standard commercial cloud products, but assuming customers' premises as the central cloud locations. In addition, those providers have also devised and created their own hardware devices for the edge adapted to run their new (cloud) edge solutions. This way, cloud providers made sure that the selected equipment would be plug-and-play, prepared to run their products, keeping their advantages and service level with zero configuration needed from the final user. However, apart from creating huge vendor lock-in environments, most of these frameworks are not actually edge-native solutions responding to the relevant requirements; they only shift workloads outside the cloud infrastructure but without breaking a strong dependency on the cloud or adapting architectural paradigms to the edge.

Amazon Web Services (AWS) offers a solution named AWS IoT Greengrass for deploying processing capabilities over nodes at the edge tier of the architecture, especially over IoT devices that gather data from various attached sensors [20]. This solution includes *serverless*-based deployments in the devices using AWS Lambda, the *serverless* approach of AWS. It transforms models that were previously created in the cloud into Lambda functions, that are then shifted to the edge (where container virtualization and AI inference equipment exist) to be run. To avoid network connectivity loss problems, Greengrass creates a virtual twin of the device that ensures compliance with the desired state in the cloud, as long as there is a reliable connection; otherwise, it re-synchronizes whenever it is re-established and calibrates to reassure the desired state again. In addition, inter-device communication is allowed inside a local network without depending on the cloud. As for device management, AWS IoT Greengrass is highly configurable from a remote location, allowing the addition and removal of (even customized) hardware and software modules. Another solution by AWS for the edge is AWS Snowball Edge, a device type from the AWS Snowball family that is designed to work at the customer's premises [21]. Similar to Greengrass modules, Snowball edge devices can be managed locally and through the cloud. Finally, these devices can be classified as the top layer of the edge tier of the computing continuum because they provide great computing capabilities, approaching a small cloud datacenter.

Microsoft Azure provides a complete stack for edge computing under its Azure IoT Edge framework, including remote equipment management, edge-level virtualization, and remote workload allocation and control [22]. This technology is delivered with an open-source MIT license via the GitHub account of Azure, allowing developers to deploy and integrate Azure's edge stack in their own infrastructure. However, interoperability is not complete, as the installation depends on the Azure cloud stack. Both Azure and AWS have delivered several series of their own certified devices targeting the lowest and medium tier of the edge computing range (some of them are based on popular embedded boards, such as Raspberry Pi, NVIDIA Jetson and Intel boards). This certification consists of the validation by the Azure cloud engineers that a device "can connect with Azure IoT Hub and securely provision through the Device Provisioning Service (DPS)" [23]. Similar to Amazon, Microsoft offers a line of powerful equipment to bring the Azure services to the customer's premises and remote locations (near the sources of data), lowering the amount of data to be uploaded to the cloud by the Azure Stack Edge.

Google Cloud integrates edge computing solutions under its Google Distributed Cloud solution. Google devised a complete edge-to-cloud continuum infrastructure by offering services that can be deployed on all of their own-defined tiers of the edge-to-cloud architecture layers (Google's network edge, telecommunications operators' edge, customers' datacenters, and customer' remote edge), including the telecommunication service provider network layer and the possibility of virtualizing telcos' 5G network elements [24].

4. Container Virtualization: Deploying Container Workloads on the Edge

In 2015, the Open Container Initiative (OCI) was founded by the container technology leaders (Docker, CoreOS, …) in order to specify a standard for container virtualization. This action resulted in the creation of three specifications: the Runtime Specification (*runtime-spec*, the specification for running containers), the Image Specification (*image-spec*, for packaging containers in images) and the Distribution Specification (*distribution-spec*, for sharing and using container images) [25]. Two years later, when the Docker Engine was established as the technological reference for containers, Docker open-sourced the code of its container platform, divided into modules (including its high-level container runtime—*containerd* [26] and its low-level container runtime—*runC* [27]), with the purpose of bringing the capability of building fully customized container-based systems to system builders. This innovation received the name the Moby Project [28]. It allowed engineers to focus on the development of lighter container runtimes through the reduction of their internal components and the removal of unnecessary modules in order to increase their performance or even to design specific operating systems incorporating such runtimes, which appear to be interesting trends for both cloud and edge.

4.1. Container Engines

Container engines (e.g., Docker, Podman) are comprised of tools for building and running container images. To achieve the latter, a high-level container runtime (also referred to as the container manager, this block controls the transport and management of images) interacts with a low-level container runtime (the block that receives the unpackaged container image from the high-level runtime and finally runs the container inside the system interacting with the host's OS kernel), as shown in **Figure 1**. As mentioned in the previous paragraph, Docker Engine's runtime (de-facto standard) materializes in *containerd* and *runC*. However, there are other runtime implementations that have been developed with edge computing constraints in mind.

crun is a low-level container runtime fully compliant with the OCI runtime specification that is written in C, the same as LXC and the Linux kernel, unlike *runC*, which is written in Go. Here, *crun* improves upon *runC*, as C can bring better performance than Go. Rust is a programming language that has been established as a strong alternative to C and C++ due to remarkable improvements in memory safety. Rust also betters Go in the implementation of system calls, which has pushed its usage *to develop youki, an OCI-compliant low-level container runtime that shares with crun the reduction of binary size, making both suitable for environments with constrained resources*.

Focusing on the high-level runtimes, *CRI-O* appears to be one of the more interesting solutions. It is a lightweight container runtime specifically designed for Kubernetes [46]. Its main advantage is the reduction of resource consumption in comparison with Docker's *containerd*, Moby or *rkt* (also analyzed), and its compatibility with any OCI-compliant low-level runtime, an interesting feature that allows this runtime to run workloads not only based on containers

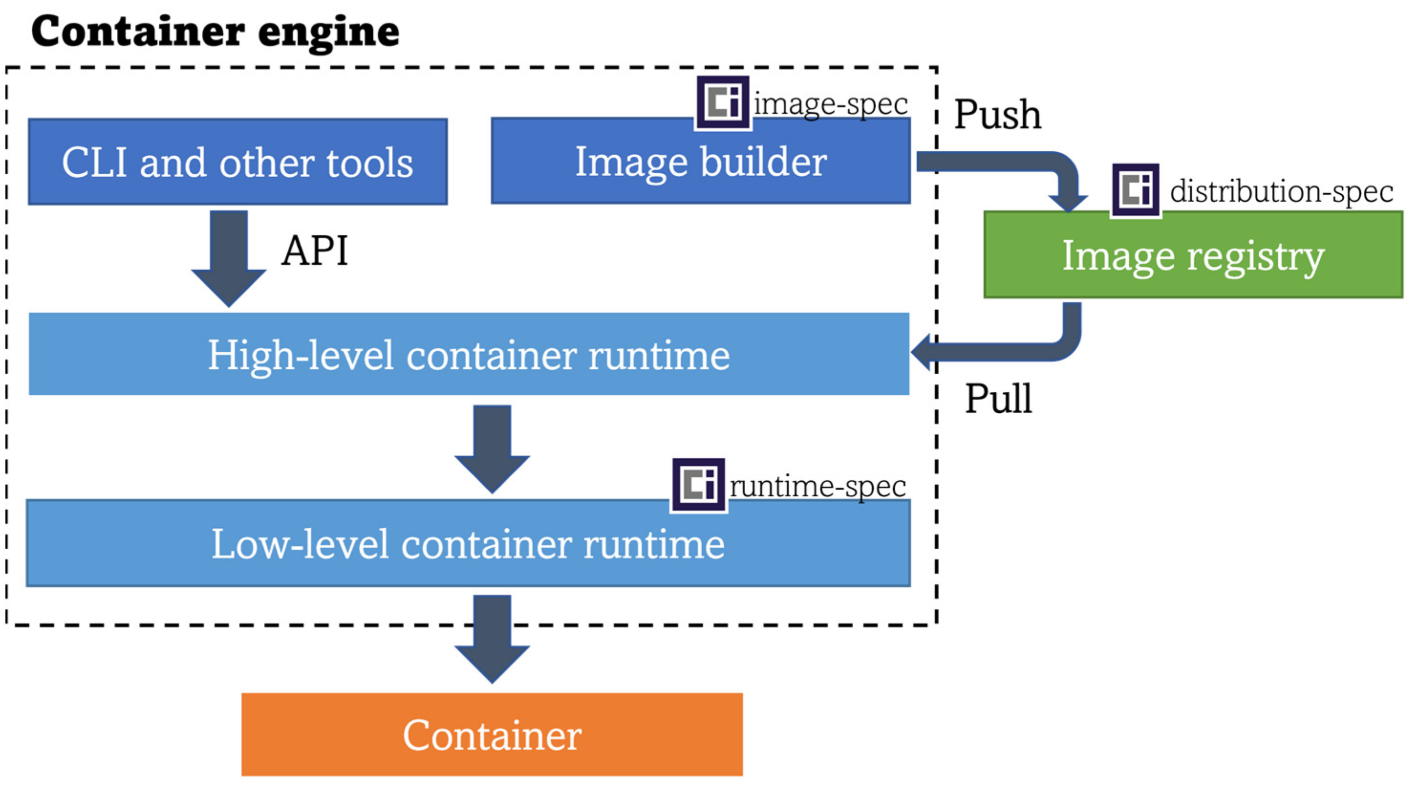

Figure 1. Container workflow block architecture.

4.2. Purpose-Built Operating Systems

In the same way that container runtimes are being reduced to target a wider range of systems, some specific OSs have appeared that take advantage of reduced and portable runtimes with the main goal of bringing container virtualization to increasingly constrained, as well as embedded devices.

EVE-OS is an operating system originally developed by ZEDEDA and then donated under an open-source license to the Linux Foundation that has included it inside its edge research projects stack (LF-Edge) [29]. The main purpose for the development of this OS is to provide edge computing devices with a universal, vendor-agnostic and standardized architecture OS, following the same strategy that Google used in the smartphone market when it delivered Android. EVE-OS has adopted well-known open-source projects such as Xen Project, LinuxKit and Alpine Linux for its development. Currently, the remote management of a fleet of devices running EVE-OS following a Zero Trust security model is possible using the Adam controller, the reference implementation of an LF-Edge API-compliant Controller. Furthermore, EVE-OS provides support for running containerized workloads using *containerd* and for running Kubernetes distributions on top of it. EVE's contributors have recently announced an architecture proposal for integrating EVE with Kubernetes at a control plane level [30]. Other interesting operating systems for running containers in constrained devices are *BalenaOS* [31], which is based on Yocto Linux and includes its own container engine -*balenaEngine*, based on the Moby project technology-, and *Pantavisor* [32], more focused on embedded devices because supports a wide range of CPU architecture.

Although containers can be managed using their own interfaces, some tools have appeared on top of container virtualization technologies to better manage and orchestrate the deployment, allowing scheduling, scaling and control. Kubernetes (K8s) has become the standard for container and microservices orchestration in the cloud, gaining advantage over its competitors in recent years, such as Docker Swarm or Apache Mesos [33]. Consequently, as K8s is a highly customizable and open technology, most cloud providers have delivered their own K8s distributions that are fully compliant with the standard in order to optimize the integration of K8s with their systems. Bringing containers to edge computing deployments is possible, and it is in fact being implemented. Therefore, to transfer cloud-native traits provided by a *standardized* container orchestration tool (K8s) to the edge, an equivalent technology—adapted to edge computing—must be found. This is a relevant point, as Kubernetes itself does not seem to be the proper solution. Although its deployment is feasible in some computing nodes along the computing continuum [34][35] (those that can carry powerful workloads), it does not suit the capacities of more resource-constrained equipment such as far-edge nodes or leaf devices.

The solutions that have been discovered in state-of-the-art investigations are divided into two main groups. The first group focuses on replicating Kubernetes in edge environments by reducing the memory footprint of this orchestration platform to broaden the range of equipment in which K8s can be fully functional. Here, different lightweight versions of Kubernetes can co-live with a full K8s deployment in the cloud premises, creating a sort of multi-cluster environment so that communication and orchestration in the computing continuum is achieved. This approach is illustrated in **Figure 2**a. On the other hand, **Figure 2**b depicts the second trend of creating new frameworks specifically adapted to the edge that follow Kubernetes principles but are not direct simplifications or reductions of that technology. These solutions adapt K8s to the specific characteristics of the edge (unstable network connectivity, difficulty of managing heterogeneous and low-resource equipment), maintaining all its benefits achieved in the cloud and not just limiting themselves to creating another lightweight K8s distribution without reaching a real edge-to-cloud synergy.

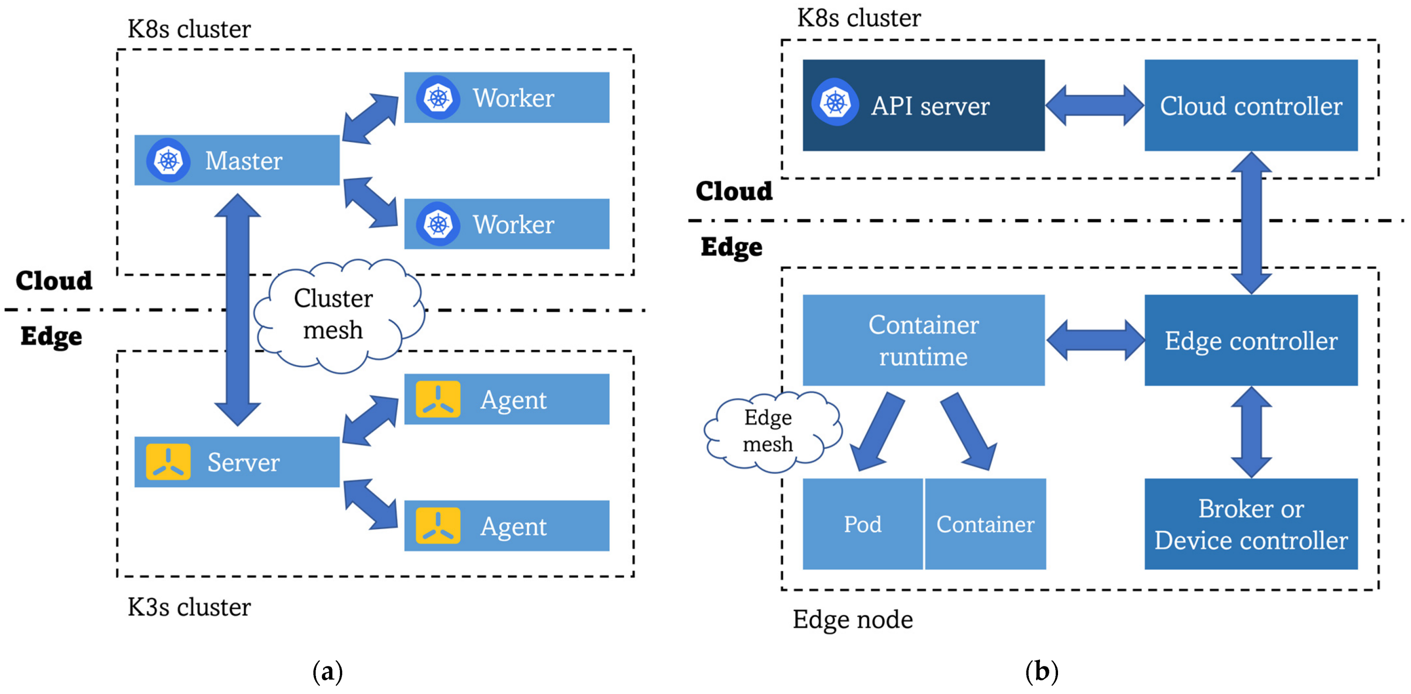

Figure 2. Comparison of architectures for K8s deployment at the edge: (**a**) K8s multi-cluster architecture using lightweight K8s distributions for the edge clusters; (**b**) K8s adapted to the edge computing requirements general architecture.

In the first group, *K3s* stands out. *K3s* is a lightweight, fully compliant Kubernetes distribution developed by Rancher. It was created for running in constrained devices, bearing a much lower memory footprint than other available K8s distributions [36]. *K3s* modifies the K8s paradigm of master and worker nodes, converting them into server and agent nodes. On another note, it offers three possible architectures: (1) a single server with an embedded SQLite database and (2) high-availability servers using an embedded or (3) an external database (SQL-based or *etcd*). This distribution is also optimized for several CPU architectures, such as ARM32, ARM64, and ARMv7. Henceforth, it becomes preferrable for edge environments (which often use embedded systems as nodes, like Raspberry Pis or NVIDIA Jetson boards). *K3s*'s minimum requirements are 256 MB of RAM for an agent node, and 512 MB for a server node with some workloads running in the agent node. Rancher also delivered an OS for edge nodes optimized for running *K3s*, maintaining only minimal resources of the underlying OS: *k3OS* [37].

In the second group, the design of the deployment differs ostensibly. Some of the devised new frameworks that adapt K8s principles to the edge are based on maintaining the control plane of the system in the cloud, and moving only the needed workloads and specific controllers to the edge, converting it into an autonomous node of the system regarding the application plane. So minimized, essential information from the control plane is cached on the edge or in intermediate nodes in order to be accessible in case of network connectivity issues. Among the analyzed solutions of this type, *KubeEdge* stands out. *KubeEdge* [38] is an open-source framework based on the Kubernetes architecture with the main purpose of bringing the full functionalities of Kubernetes to the edge. This ongoing deployment project is officially under the umbrella of the CNCF. The main underlying idea of this technology is to keep the entire control plane in the cloud, where the computing resources are more plentiful, and leave the workloads (running the containers) or the application plane to the edge in order to dedicate all the constrained computing resources of this tier for this purpose. A remarkable feature of *KubeEdge* specifically intended for edge-native deployments is the capability of coping with poor network connection between the cloud and the edge. Synchronization is handled only under stable network conditions, thus achieving a smooth management of the environment. This is a key issue for edge-native applicationsthat is not natively solved by K8s. Finally, *KubeEdge* also provides service mesh capabilities for the edge workloads.

5. Future Directions: The Horizon beyond Containers-Only

A new approach for workload virtualization has arisen, keeping the isolation trait while skipping some inconveniences of the VMs. This is reached by reducing the size and requirements of VMs, achieving the so-called microVMs or lightweight VMs that intend to be even lighter and faster than containers. Kata Containers is an OpenStack tool whose main goal is to run lighter VMs instead of containers while remaining compliant with the OCI runtime and image specifications. By sticking to OCI, Kata allows Docker images previously built from Dockerfiles to be deployed as microVMs (Kata containers [39]).

Another approach to replace or complement containers with microVMs are Unikernels. According to A. Madhavapeddy and D.J. Scott [40], Unikernels are single-purpose appliances that are compile-time specialized into standalone kernels and sealed against modification when deployed to a cloud platform". The main idea beyond Unikernels is to use only the strictly necessary part of the user and kernel space of an operating system to obtain a customized OS that will be run by a type 1 hypervisor. Such customized OS is achieved through the usage of a library operating system (library OS or libOS), removing the need for a whole guest OS [41]. This is translated into a reduction of image size and their booting time, as well as their footprint and possible attack surface. However, this virtualization technology has many disadvantages, the most prominent being the lack of standardization as compared with containers, in addition to the need of a complete library operating system rebuild for every new application with any minimal changes, followed by the limitation of debugging and monitoring capabilities[] Unikernel applications are also language-specific (there are Unikernel development systems only for a few languages), being also a considerable limitation for developers. *Unikraft* [42][43] is a Linux-based "automated system for building Unikernels" under the umbrella of the Linux Foundation (inside its Xen Project) and partially funded by the EU H2020 project UNICORE [44]. *Unikraft* presents an improved performance in comparison to its peers and other virtualization technologies, such as Docker (regarding boot time, image size, and memory consumption). In addition, *Unikraft* leverages an OCI-compliant runtime (*runu*) along with *libvirt* (a toolkit for interacting with the hypervisor) for running Unikernels. The novelty is that *runu* natively supports the execution of workloads previously packaged following the OCI Image Specification (unlike Nabla), enabling Unikraft-built Unikernels' interaction with containers and cloud-native platforms like K8s.

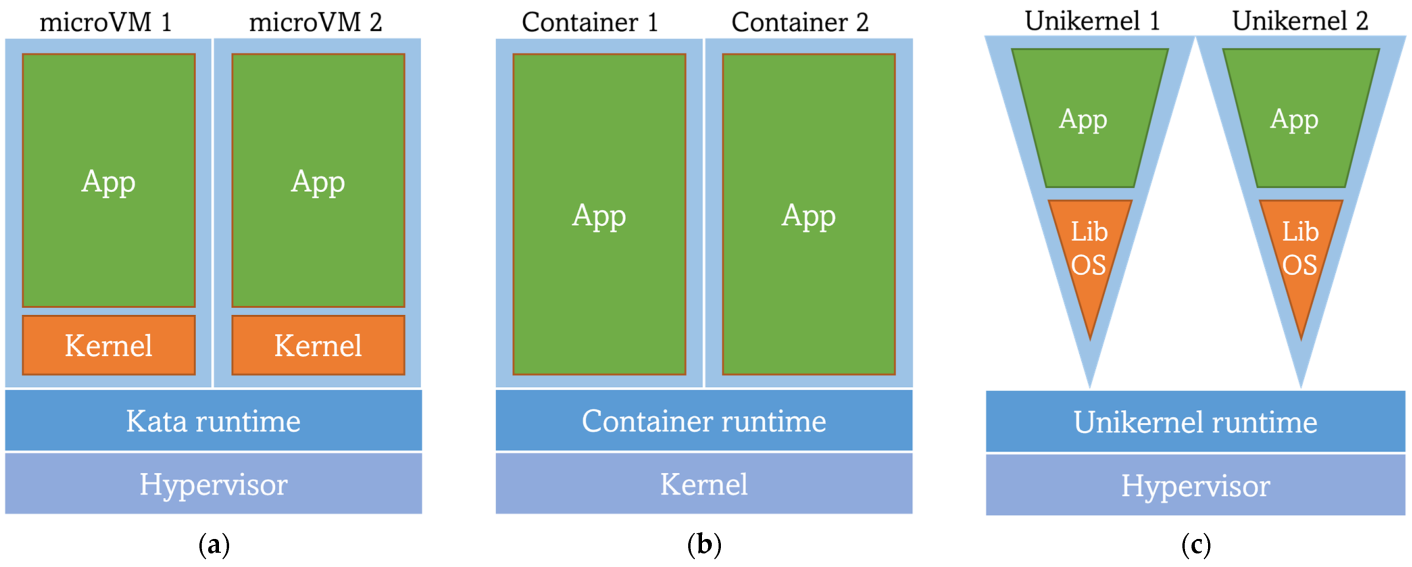

Figure 3. Comparison of the architectures of different virtualization techniques: (a) MicroVMs lightweight VM; (b) Containers; (c) Unikernels.

One step ahead of MicroVMs and Unikernels is the most recent and most promising trend in workload virtualization: WebAssembly (Wasm). Wasm is "*a binary instruction format for stack-based virtual machines*" developed as an open standard by the World Wide Web Consortium (W3C), that seeks to establish itself as a strong alternative to containers. Wasm allows software to be written in several high-level languages (C++, Go, Kotlin, …) to be compiled and executed with near-native performance in web applications that are designed to run in web browsers [45][46]. At a high level, the functioning of Wasm is simple: native code functions are compiled into Wasm modules that then are loaded and used by a web application (in JavaScript code) in a similar way to how ES6 modules are managed.

In recent times, developers have attempted to move Wasm outside web browsers, or, in other words, from the (web) client side to the server side for a standalone mode of operation. Wasm binaries' tiny size, low memory footprint, great isolation, fast booting (up to 100 times faster than containers), and response times make Wasm perfect for running workloads in edge and IoT devices using Wasm runtimes such as *Wasmedge*, *Wasmer* and *Wasmtime*. Adopting Wasm in edge environments (where resource-constrained nodes struggle to run container workloads) could lead to an increased number of simultaneously running services as compared with the current container throughput in the same environment [47][48]. This research led to the creation of the WebAssembly System Interface (WASI), a "modular system interface for WebAssembly" with the main purpose of enabling the execution of Wasm on the server side through the creation and standardization of APIs.

The hype about this technology is such that Docker cofounder Solomon Hykes stated in 2019, that if Wasm along with WASI had existed in 2008, they would not have needed to create Docker [49]. However, this does not mean that these two types of workloads are incompatible or that one technology is expected to replace the other in the future; both can be combined, and the choice of the working technology should be made depending on the final business use case. Therefore, the coexistence of different types of workloads puts into the scene the requirement of a common orchestration mechanism, so Kubernetes stands out as the most interesting candidate. Observing the general architecture of it, the *kubelet* component interacts with a high-level container runtime that implements the Kubernetes Container Runtime Interface (CRI) acting as a gRPC API client for launching pods and their containers. Looking deeper into K8s functioning, a resource named *Runtime Class* is the key underlying component. This class is “a feature for selecting the container runtime configuration” [50], which can be leveraged for mapping different workload types with their corresponding low-level runtime (same architectural spot as the low-level runtime in **Figure 1**). Therefore, the installed high-level container runtime in the cluster, for instance, *CRI-O* or *containerd*, will be able to send a request to the proper low-level runtime in order to actually run the requested workload, as shown in **Figure 4**. The only requirement is packaging the application into an OCI-compliant image previously stored in an OCI-compliant image registry (same pre-condition as for the Docker Engine).

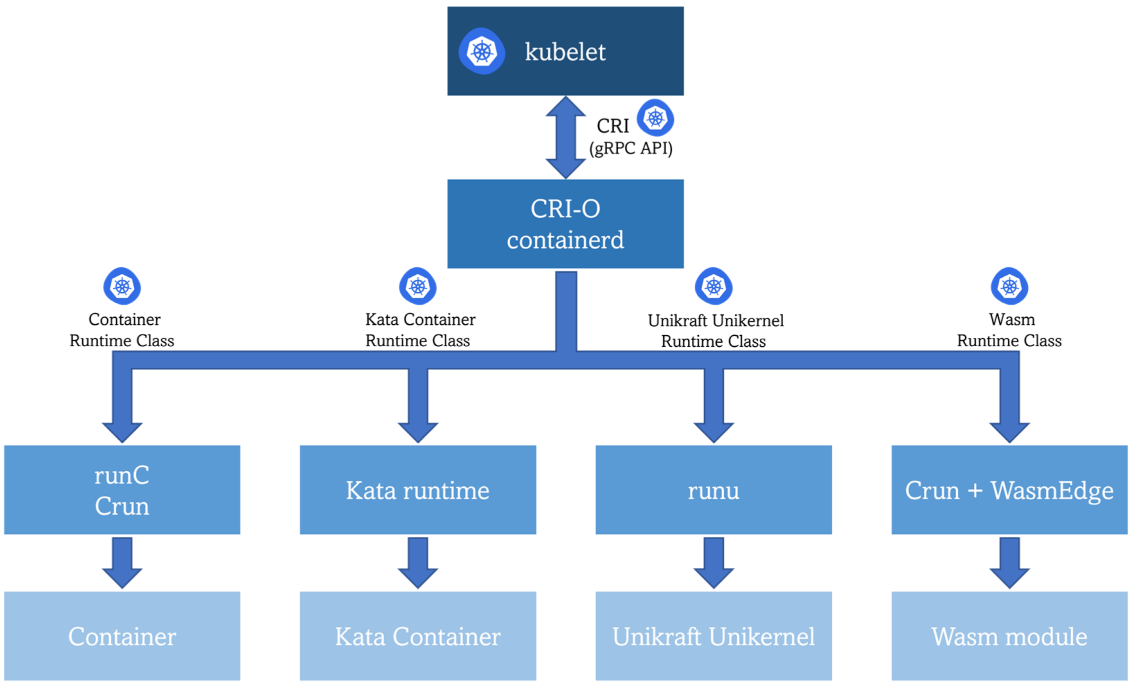

Figure 4. Running different types of workloads in K8s.

References

1. Satyanarayanan, M.; Klas, G.; Silva, M.; Mangiante, S. The Seminal Role of Edge-Native Applications. In Proceedings of the 2019 IEEE International Conference on Edge Computing,

EDGE 2019—Part of the 2019 IEEE World Congress on Services, Milan, Italy, 1 July 2019; Institute of Electrical and Electronics Engineers Inc.: Milan, Italy, 2019; pp. 33–40.

2. Raj, P.; Vanga, S.; Chaudhary, A. Delineating Cloud-Native Edge Computing. In Cloud-Native Computing: How to Design, Develop, and Secure Microservices and Event-Driven Applications; Wiley-IEEE Press: Piscataway, NJ, USA, 2023; pp. 171–201.
3. LF Edge Projects. Available online: https://www.lfedge.org/projects/ (accessed on 9 January 2023).
4. The Eclipse Foundation: Edge Native Working Group. Available online: https://edgenative.eclipse.org/ (accessed on 9 January 2023).
5. CNCF Cloud Native Landscape. Available online: https://landscape.cncf.io/ (accessed on 9 January 2023).
6. Shaping Europe's Digital Future: IoT and the Future of Edge Computing in Europe. Available online: https://digital-strategy.ec.europa.eu/en/news/iot-and-future-edge-computing-europe (accessed on 9 January 2023).
7. European Comission Funding & Tender Opportunities: "Future European Platforms for the Edge: Meta Operating Systems (RIA)". Available online: https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/topic-details/horizon-cl4-2021-data-01-05 (accessed on 9 January 2023).
8. European Comission Funding & Tender Opportunities: "Cognitive Cloud: AI-Enabled Computing Continuum from Cloud to Edge (RIA)". Available online: https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/topic-details/horizon-cl4-2022-data-01-02 (accessed on 9 January 2023).
9. EUCloudEdgeIOT: Building the European Cloud Edge IoT Continuum for Business and Research. Available online: https://eucloudedgeiot.eu/ (accessed on 9 January 2023).
10. Malhotra, L.; Agarwal, D.; Jaiswal, A. Virtualization in Cloud Computing. J. Inf. Technol. Softw. Eng. 2014, 4, 1–3.
11. Bhardwaj, A.; Krishna, C.R. Virtualization in Cloud Computing: Moving from Hypervisor to Containerization—A Survey. Arab. J. Sci. Eng. 2021, 46, 8585–8601.
12. OpenFaaS: Serverless Functions Made Simple. Available online: https://www.openfaas.com/ (accessed on 9 January 2023).
13. Knative: An Open-Source Enterprise-Level Solution to Build Serverless and Event Driven Applications. Available online: https://knative.dev/docs/ (accessed on 9 January 2023).
14. CloudEvents: A Specification for Describing Event Data in a Common Way. Available online: https://cloudevents.io/ (accessed on 9 January 2023).

15. Maenhaut, P.J.; Volckaert, B.; Ongenae, V.; de Turck, F. Resource Management in a Containerized Cloud: Status and Challenges. J. Netw. Syst. Manag. 2020, 28, 197–246.
16. Yadav, A.K.; Garg, M.L. Ritika Docker Containers versus Virtual Machine-Based Virtualization. Adv. Intell. Syst. Comput. 2019, 814, 141–150.
17. Canonical Ltd. Linux Containers: LXC. Available online: https://linuxcontainers.org/lxc/introduction/ (accessed on 9 January 2023).
18. Duan, Q. Intelligent and Autonomous Management in Cloud-Native Future Networks—A Survey on Related Standards from an Architectural Perspective. Future Internet 2021, 13, 42.
19. Alonso, J.; Orue-Echevarria, L.; Casola, V.; Torre, A.I.; Huarte, M.; Osaba, E.; Lobo, J.L. Understanding the Challenges and Novel Architectural Models of Multi-Cloud Native Applications—A Systematic Literature Review. J. Cloud Comput. 2023, 12, 1–34.
20. AWS IoT Greengrass. Available online: https://aws.amazon.com/greengrass/features/ (accessed on 9 January 2023).
21. What Is AWS Snowball Edge?—AWS Snowball Edge Developer Guide. Available online: https://docs.aws.amazon.com/snowball/latest/developer-guide/whatisedge.html (accessed on 9 January 2023).
22. Microsoft Azure IoT Edge. Available online: https://azure.microsoft.com/en-us/products/iot-edge/#iotedge-overview (accessed on 9 January 2023).
23. Certifying IoT Devices: Azure Certified Device Program. Available online: https://www.microsoft.com/azure/partners/azure-certified-device (accessed on 9 January 2023).
24. Google Distributed Cloud. Available online: https://cloud.google.com/distributed-cloud (accessed on 9 January 2023).
25. Open Container Initiative. Available online: https://opencontainers.org/ (accessed on 9 January 2023).
26. containerd: An Industry-Standard Container Runtime with an Emphasis on Simplicity, Robustness and Portability. Available online: https://containerd.io/ (accessed on 9 January 2023).
27. runC: CLI Tool for Spawning and Running Containers According to the OCI Specification. Available online: https://github.com/opencontainers/runc (accessed on 9 January 2023).
28. Moby Project. Available online: https://mobyproject.org/ (accessed on 9 January 2023).
29. EVE—LF Edge. Available online: https://www.lfedge.org/projects/eve/ (accessed on 9 January 2023).
30. EVE Kubernetes Control Plane Integration—Draft. Available online: https://wiki.lfedge.org/display/EVE/EVE+Kubernetes+Control+Plane+Integration+-+Draft

(accessed on 9 January 2023).

31. BalenaOS—Run Docker Containers on Embedded IoT Devices. Available online: https://www.balena.io/os/ (accessed on 9 January 2023).
32. Pantavisor: A framework for containerized embedded Linux. Available online: https://pantavisor.io/ (accessed on 9 January 2023).
33. Truyen, E.; van Landuyt, D.; Preuveneers, D.; Lagaisse, B.; Joosen, W. A Comprehensive Feature Comparison Study of Open-Source Container Orchestration Frameworks. Appl. Sci. 2019, 9, 931.
34. Paszkiewicz, A.; Bolanowski, M.; Ćwikła, C.; Ganzha, M.; Paprzycki, M.; Palau, C.E.; Lacalle Úbeda, I. Network Load Balancing for Edge-Cloud Continuum Ecosystems. In Innovations in Electrical and Electronic Engineering; Mekhilef, S., Shaw, R.N., Siano, P., Eds.; Springer Science and Business Media Deutschland GmbH: Antalya, Turkey, 2022; Volume 894 LNEE, pp. 638–651.
35. Balouek-Thomert, D.; Renart, E.G.; Zamani, A.R.; Simonet, A.; Parashar, M. Towards a Computing Continuum: Enabling Edge-to-Cloud Integration for Data-Driven Workflows. Int. J. High Perform. Comput. Appl. 2019, 33, 1159–1174.
36. K3s: Lightweight Kubernetes. Available online: https://k3s.io/ (accessed on 9 January 2023).
37. K3OS: The Kubernetes Operating System. Available online: https://k3os.io/ (accessed on 9 January 2023).
38. KubeEdge: A Kubernetes Native Edge Computing Framework. Available online: https://kubeedge.io/en/ (accessed on 9 January 2023).
39. Kata Containers: Open-Source Container Runtime, Building Lightweight Virtual Machines That Seamlessly Plug into the Containers Ecosystem. Available online: https://katacontainers.io/ (accessed on 9 January 2023)
40. Madhavapeddy, A.; Scott, D.J. Unikernels: Rise of the Virtual Library Operating System. Queue 2013, 11, 30–44.
41. Sarrigiannis, I.; Contreras, L.M.; Ramantas, K.; Antonopoulos, A.; Verikoukis, C. Fog-Enabled Scalable C-V2X Architecture for Distributed 5G and beyond Applications. IEEE Netw. 2020, 34, 120–126. [Google Scholar] [CrossRef]
42. Kuenzer, S.; Bădoiu, V.-A.; Lefeuvre, H.; Santhanam, S.; Jung, A.; Gain, G.; Soldani, C.; Lupu, C.; Răducanu, C.; Banu, C.; et al. Unikraft: Fast, Specialized Unikernels the Easy Way. In Proceedings of the Sixteenth European Conference on Computer Systems, Online Event, UK, 26–28 April 2021; pp. 376–394.
43. Unikraft: A Fast, Secure and Open-Source Unikernel Development Kit. Available online: https://unikraft.org/ (accessed on 9 January 2023).

44. UNICORE EU H2020 Project. Available online: https://unicore-project.eu/ (accessed on 9 January 2023).

45. WebAssembly. Available online: https://webassembly.org/ (accessed on 9 January 2023).

46. WebAssembly Community Group WebAssembly Specification Release 2.0 (Draft 2023-02-13); 2023. Available online: https://webassembly.github.io/spec/core/_download/WebAssembly.pdf (accessed on 13 February 2023).

47. Ménétrey, J.; Pasin, M.; Felber, P.; Schiavoni, V. WebAssembly as a Common Layer for the Cloud-Edge Continuum. In Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge, FRAME 2022, Co-Located with HPDC 2022, Minneapolis, MN, USA, 1 July 2022; Association for Computing Machinery, Inc.: Minneapolis, MN, USA, 2022; pp. 3–8.

48. Mäkitalo, N.; Mikkonen, T.; Pautasso, C.; Bankowski, V.; Daubaris, P.; Mikkola, R.; Beletski, O. WebAssembly Modules as Lightweight Containers for Liquid IoT Applications. In Proceedings of the Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Biarritz, France, 18–21 May 2021; Springer Science and Business Media Deutschland GmbH: Berlin, Germany, 2021; Volume 12706 LNCS, pp. 328–336.

49. Charboneau, T. Why Containers and WebAssembly Work Well Together. Available online: https://www.docker.com/blog/why-containers-and-webassembly-work-well-together/ (accessed on 9 January 2023).

50. Kubernetes Reference Documentation: Runtime Class. Available online: https://kubernetes.io/docs/concepts/containers/runtime-class/ (accessed on 9 January 2023).

Retrieved from https://encyclopedia.pub/entry/history/show/94010