The Journal of Supercomputing (2022) 78:1144–1181
https://doi.org/10.1007/s11227-021-03914-1

Containerization technologies: taxonomies, applications and challenges

Ouafa Bentaleb[1,2,3] · Adam S. Z. Belloum[3] · Abderrazak Sebaa[4,5] · Aouaouche El-Maouhab[1]

Accepted: 22 May 2021 / Published online: 8 June 2021

Abstract

Modern scientific research challenges require new technologies, integrated tools, reusable and complex experiments in distributed computing infrastructures. But above all, computing power for efficient data processing and analyzing. Containers technologies have emerged as a new paradigm to address such intensive scientific applications problems. Their easy deployment in a reasonable amount of time and the few required computational resource make them more suitable. Containers are considered light virtualization solutions. They enable performance isolation and flexible deployment of complex, parallel, and high-performance systems. Moreover, they gained popularity to modernize and migrate scientific applications in computing infrastructure management. Additionally, they reduce computational time processing. In this paper, we first give an overview of virtualization and containerization technologies. We discuss the taxonomies of containerization technologies of the literature, and then we provide a new one that covers and completes those proposed in the literature. We identify the most important application domains of containerization and their technological progress. Furthermore, we discuss the performance metrics used in most containerization techniques. Finally, we point out research gaps in the related aspects of containerization technology that require more research.

Keywords Computing · Parallel · Scientific application · Micro-services · Virtualization · Container · Orchestrator

✉ Ouafa Bentaleb
ouafabentaleb@gmail.com

Extended author information available on the last page of the article

1 Introduction

In the last years, there has been a growing need for solving complex problems in a reasonable amount of time. This requirement has led to the implementation of rather complex, parallel, and high-performance systems. Virtualization technology is a common way to reduce this complexity and increase the system's flexibility [1]. Virtualization used as a basic strategy to incorporate servers in data and computing infrastructures. Virtualization allows for improving resource utilization and allocation. It represents a basic building block for deployments in cloud computing [2]. There are two main technologies used to implement virtualization, namely virtualization-based hypervisors, and virtualization-based containers technologies. However, virtualized computing infrastructures based on hypervisor [3] have not efficient deployments during resources management, where thus resources are needed to accelerate applications in terms of time processing.

Unlike hypervisor-based deployment, applications implemented in environment based-containers share the same operating system, while being isolated from the other applications. Therefore, this deployment is a significantly smaller memory footprint [4]. Moreover, the utilization of virtualization-based containers keeps growing. It provides a full middleware stack that offers mechanisms suitable for deploying, and running applications that allow features such as scalability, fault tolerance, and high availability. At the heart of the containerization approach, virtualization-based containers are a leading concept, responsible to provide services. These services allow running various environments sharing computing resources, across computing infrastructures [5].

Containerization technology was first proposed by IBM [6] in 1979. Implemented in UNIX operating system V7, and introducing a chroot [7] system call. This advance was the beginning process for isolation, running isolated groups on a single host. That isolation leverage several underlying technologies built on the Linux kernel: namespaces and cgroups [8]. The support of namespaces introduced in Linux kernel version 2.4.19, while cgroups released in Linux kernels version "2.6.24", known as the control group technology.

The emergence of micro-services architecture [9] based on containers technology, considering, Linux containers (LXC) [7], OpenVZ [10], Docker [11], Singularity [12], and uDocker [13] created a shift in the way we develop applications, from operations to programming. Many popular orchestration tools, such as Kubernetes [14], Docker Swarm [2], and Apache Mesos [15] used container lifecycle management. These orchestrators provided frameworks for managing containers within micro-services architecture. Moreover, these frameworks enable features suitable for scheduling containers with resources, which are under limits control, fault tolerance, and allow auto-scaling. Among the orchestration tools, Kubernetes [16] and OpenShift [17] are becoming widely adopted in computing systems including, industry and scientific domains. Thereafter, recently Rancher compliant orchestration management platforms evolved to manage orchestrators, and maintain efficiency features that ensure performance across the computing infrastructure.

Although cloud computing [18] is the most popular environment for the containerization of applications. Containerization strategy applied to diverse applications domains beyond cloud services, such as scientific computing, big data processing, Internet of Things and edge computing.

Despite the substantial body of literature on containerization technologies, only a few studies review and survey [1–5, 19] this field. Thus, the surveys [1–5, 19] pointed out the challenges related to the adoption of containerization technologies, identifying the main issues that arise in containerization implementation, such as performance, orchestration, and security. However, none of these studies has provided a general overview of the related aspects of containerization. They are focusing only on containerization in cloud infrastructures. Moreover, a lot of work done afterward [14, 19–21] and many of the aspects related to containerization technologies have since evolved. It is thus necessary to extend these studies to cover the newly emerging relevant aspects of containerization, to deploy application quickly across distributed clusters and provide a complete taxonomy, effectiveness, and limitations of related containerization technologies. The primary motivation behind this survey is to provide a comprehensive understanding of containerization technologies, fundamentals for researchers interested in the field of virtualization, to complement, fill the gaps of previous efforts, a result of the fast evolution of container technology over the past several years, and give a view of these technologies, including recent advances and future trends.

Relevant literature was identified by querying scholarly databases for the terms "**Containerization Technologies**". Then, returned results were downloaded, read, and analyzed. The scholarly databases queried included: (1) Google Scholar. (2) ScienceDirect. (3) SpringerLink database. (4) IEEE Xplore. (5) ACM Digital Library. In all reviewed papers, each one was carefully read and analyzed to extract the research problem, the containers technologies, containers orchestration platforms, applications domains, most relevant container taxonomies in the literature, to obtain a complete classification and a comprehensive review of various performance evaluation metrics.

The main contributions of this work are summarized as follows:

- We discuss the previous taxonomies of containerization technologies, provide a new one that covers, and complete those proposed in the literature.
- We give an overview of containerization technology and define the most important concepts features of each concept. We also describe the most popular orchestration platforms of the literature.
- We overview several application domains in which containerization technologies have been proven successful and their evolution across these application domains in distributed computing infrastructures [22], including Grid, Cloud, IoT (Internet of things), Fog, and Edge computing. Such application domains have evolved over the past years towards a similar vision that focuses on delivering computational resources.
- We provide the main performance evaluation criteria considered when designing implementing a containerization technology.

- We identify research gaps of the related aspects, which partially explored or not addressed to inspire new research contributions.

The remainder of this work organized as follows. In Sects. 2 and 3, we describe various container concepts, their emergence, features offered by each technology while illustrating the most popular orchestration platforms. Then, in Sect. 4, we discuss containerization taxonomy and then propose a new taxonomy classification for containerization following the architecture-based containers model introduced in Sect. 3. In Sect. 5, we identify the application domains of containerization and the technological progress through these different application domains. Finally, in Sect. 6, we discuss performance parameters measurement for containerization techniques.

2 Overview

2.1 Virtualization

Virtualization is a technology for provisioning resources to end-users beyond the computing infrastructure. The main feature of virtualization is achieved through an isolation layer, built on top of the operating system level. For instance, provide users with an environment similar to a dedicated server. In the late 1960s and early 1970s, IBM [6] has introduced the virtualization concept in their systems. Since that time, there has been considerable evolution in thus virtualization ecosystems. There is a clear distinction, between relevant types of virtualization, which have emerged over time, para-virtualization and full-virtualization [1]. In 2005, VMware [22] proposes the para-virtualization (VMI), as a communication mechanism between the guest operating system and the hypervisor. This mechanism has enabled a new wave of transparent para-virtualization, in which a single binary version of the operating system can run either on bare hardware. However, the full-virtualization concept was implemented as a hardware module. This type of virtualization allows users to deploy virtual machines with full isolation from the operating system level, which can be performed directly on the top of the hardware.

The most important benefit of virtualization is to abstract the hardware. However, it gives an isolated [23] working environment for applications, by creating an aggregated pool of logical resources, such as CPU, memory, network, and disks. As shown in Fig. 1a, a full guest OS of the virtual machine (VM) instance runs as a single process on the host. This leads to high resource requirements, which elicit a slow during the start-up of the VM.

The efforts made on virtualization have spawned the largest computational systems, such as cloud computing, and data centers. Afterward, building environment-based virtualization for high-performance computing requires in-depth knowledge at least for one of the hypervisor technologies-based virtualization, such as VMware [24], Xen [25], KVM [26]. The hypervisor aims to virtualize the physical hardware, and present each VM with a standardized set of virtual devices, like using a Bridge (br0) via the network card, to translate the VM requests to the system hardware.

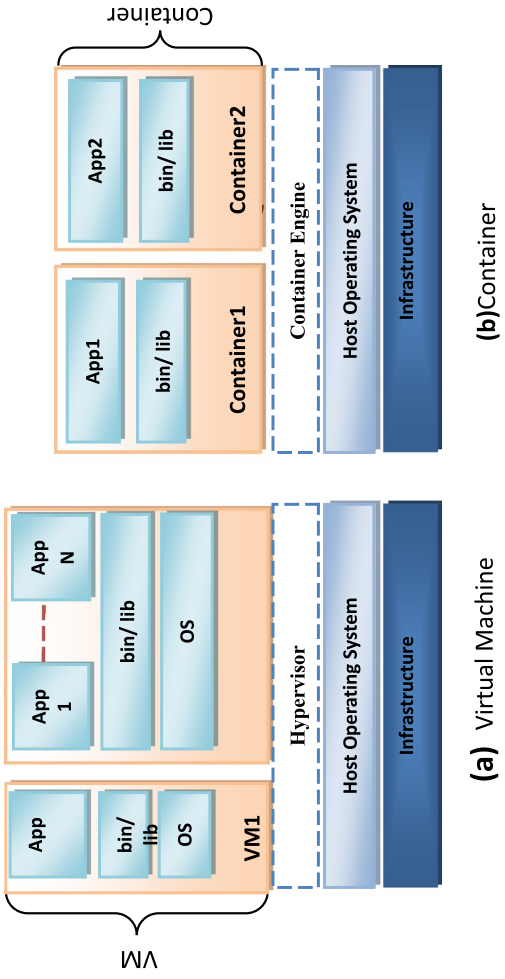

Fig. 1 System architecture-based virtualization comparison

Virtualization involves the routing of I/O to manage requests between virtual devices and the shared physical hardware. Instead of managing resources, the migration of virtual machines between physical machines spawned security issues [27]. This issue makes the system more vulnerable, and deploying systems based on virtualization became much harder. The most common type of virtualization is the operating system-level virtualization, which enables using isolation mechanisms. The isolation mechanism provides users with virtual environments similar to a dedicated server. This isolated virtual environment illustrated in Fig. 1b is called **Container**.

2.2 Containerization alternative

Containerization technology [18] is a light alternative to virtualization. As shown in Fig. 1, containerization focus to abstract the operating system (OS) level, instead of virtualizing the hardware stack with virtual machines [28, 29]. Containerization is a major trend through which applications can run. Essentially, containerization is carried out on the ability to develop, test, and deploy applications inside containers. Furthermore, ensure the efficiency of interconnection among containers, where containers are lightweight and portable. However, they have a lower start-up time compared to one of the virtual machines (Vms), due to their ability to share resources with the host machines. These features can allow them to enhance resource utilization, and using fewer of them, such as CPU, memory and storage disks. Moreover, containers handle independently for each running instance the provision the spatial isolation, processes, file systems, namespace, and hardware resources.

Due to their feature of promoting the fast deployment of applications, container paradigms have created a shift in the way computing operations are made for scientific programming. They can reach a near-native performance when tested against bar metal execution CPU-intensive applications [30]. They have gained increasing attention in high-performance computing applications [31].

Additionally, there are many disadvantages of hypervisor-based virtualization. Some of these drawbacks have been avoided through container technologies. For instance, remove the hypervisor dependency just-in-time compilation, performance degradation, and slow booting times of VMs.

Many studies [1, 6, 20, 32], and [33] have compared the hypervisor of virtual machines with containers. For instance, the study [34] compared KVM with Docker containers, by evaluating the performances of resources including CPU speed, memory bandwidth, disk space, and network throughput. This evaluation of performances was deducted using various environments, such as bare-metal, virtual machines, Linux containers, cloud computing [35], and Internet of Things (IoT) [36]. Their results indicate that containers are more beneficial than virtual machines. Besides, KVM hypervisor uses a high bandwidth of the memory. Indeed, while Docker virtualizes only the application level, the hypervisor virtualizes the whole operating system. Moreover, the performances during the startup time of a containerized application in bare-metal enhanced by about 50%, which is better than virtual machines. Furthermore, virtualization within orchestration

platforms is challenging, due to the scaling of various heterogeneous resources [37].

Besides previous disadvantages of hypervisors, container technologies provide an isolation mechanism through Linux kernel features, including namespaces and cgroups. They can secure the run-time environment for applications [38] and network resources management [39]. Thus, mechanisms process applications at a large scale [40] by packaging the code of an application with its dependencies into an encapsulated environment [41]. Also, they offer many features, including easy replication, reproducibility, and portability across the computing infrastructure.

2.3 Containerization technologies

Container-based virtualization technology [39] has been widely investigated over the last years [42]. In this section, we briefly depicted some container technologies, including Docker, Singularity, and uDocker. These technologies are, respectively, representative of container-based virtualization techniques. Currently, they are adopted in scientific computing communities. Some of the container technologies focus on specific applicability in the industry, such as Docker. Others focus on the portability of containers across HPC environments, such as Singularity. Each of the technologies mentioned above implements its methods to achieve process hardware and network isolation. In terms of container security, isolation is the main important concept needed to enhance security level in all solutions, such as cloud computing, GRID, and HPC clusters. A user space defines a separate instance of a running container.

2.3.1 Docker

As illustrated in Fig. 1b, Docker [11] is a lightweight container-based virtualization platform. Extend functionalities from Linux containers LXC [7], such as sharing the same operating system among containers, built from the same container image, where the processes inside a container seem like they own the entire system.

Docker container provides methods for security level, by using namespaces, and Cgroups mechanisms, to achieve process hardware, and isolation mechanisms. There are many types of namespaces, like the user, net, PID, mnt, Cgroup, time, etc., which limits the user's space and provides isolated Linux kernel resources (user management, file system, network, hostname) for the container. Therefore, Cgroups kernel mechanism manages to process subsets by enforcing resource consumption.

Xiaolian Li et al. [43] perform a measurement study on Linux container security using real exploits that can break the isolation launched to attack containers, by proposing a defense mechanism to defeat all identified privilege escalation attacks. Sergei et al. [44] enhanced container security by providing a secure container mechanism to protect the container processes from outside attacks using the SGX trusted execution support of Intel CPUs. Aside from that, Docker is the most popular

platform for containerization for different operating systems (Linux and Windows). Docker allows an easy way for running and managing containers among users and data centers. Moreover, Docker images can easily build with a Dockerfile, which specifies initial tasks, then used to construct container images on top of an existing one by adding another layer. Docker-Hub [45] is the main container registry for sharing applications with automated builds. It contains all requirements related to the software application, dependencies, and libraries needed to run the application on the Docker engine. Docker-compose is the engine that provides an efficient layer to run micro-services. Indeed, micro-services are encapsulated into multicontainers of one composed application. They communicate via the network and interact with volumes for data storage. Noting that, the density of a Docker can increase to handle more workloads.

2.3.2 Singularity

Singularity [12] is a pattern developed by Lawrence Berkeley National Laboratory (Sylabs Inc.), which focuses on Linux container approaches. Allow users to create and deploy their execution environments, designed for computational science. Singularity images can build starting from specific languages and can be a result of the conversion of Docker images even layered differently from Docker. Moreover, Singularity allows the configuration of namespaces for containers. It can minimize the number of virtualized namespaces. The goal is to achieve mobility, network and resource performances. Singularity container platform provides a container with the same privileges (inside and outside it), which ensures users operate freely in their working environment. Singularity provides software stacks into a single configuration file for building and distributing containers on other different platforms [46]. This file has a special format (Singularity Image Format 'SIF'), enables new features that guaranteed reproducibility, portability, and security related to cryptography signing, i.e., (PGP key).

Singularity aims to provide a containerization engine for a security model that differs from the other models proposed by the others container platforms, which enable untrusted users to run untrusted containers safely. Singularity enables sharing the entire user namespaces with the host system except for the mnt namespace, to disable running a host file with another unprivileged user in a container. This security model enforces security policies at runtime, by leveraging kernel security modules (SELinux, second, AppArmor). Additionally, Singularity gives Admins the ability to limits attacks and prevents them from outside the containers. It prevents attacks from outside the container, which disallows the changing of users' context with sudo.

Furthermore, Singularity has proved its efficiency and changed the way to do development for computational science, using a distributed image format with an efficient and complete environment. Moreover, Singularity is very promising in production within high-performance computing, because it supports various features of resources management, such as file systems, schedulers (LSF, SLURM, HTCondor), InfiniBand, Parallelism mechanism (MPI), and NVIDIA GPUs [47].

2.3.3 uDocker

uDocker [13] is a technology introduced in 2016, oriented to provide a user-space runtime environment for containers execution under Linux operating system. The main benefit that promotes using uDocker containers is the possibility to acquire customized and isolated environments, which allow running determined micro-services. Moreover, uDocker environment can be installed without using additional software. It does not require root privileges and any administrator interventions to set up the environments.

uDdocker container images created based on Docker images using Dockerfile. Afterward, push to Docker-Hub or a private Docker registry. The main capabilities of uDocker container are capable to use process controls as a host application. Those process controls are used to access the network and enable interactive accounting, also managed by many batch systems. uDocker provides tools to access transparently to external computing environments. In addition, it is convenient to run uDocker containers into various computing infrastructures, namely cloud computing and HPC [48]. However**,** uDocker implements a large set of commands (CLI) that are the same as Docker. Also, uDocker promotes easing the use of those tools that are already familiar with Docker. uDocker engine interacts directly with Dockerhub to instantiate and run containers, using the Docker format.

Most of the scientific applications are usually developed to be executed in multi-user shared environments. This environment owns unprivileged users. Additionally, due to the ability to mount host directories to access data inside the container, it does not require an isolation feature like Docker, as long as the applications are executed without privileges [49].

2.3.4 Comparison

To get an overview of how some mechanisms and environments interact with the three presented containerization technologies, Docker, Singularity, and uDocker, Table 1 depicts the main differences.

As is shown in Table 1, Docker and uDocker are lightweight virtualization tools. They use namespaces, and cgroups mechanisms to isolate the computing environments, and spinning up the amount number of containers. Moreover, they have inherent advantages over virtual machines (VMs), considering the start-up time reduced. They are near to the native performance, and convenient for migration. Besides, Singularity supports Docker images, there are some features in common with them. For instance, it is necessary during the image creation, to specify the HOST PATH in the mount point. As it is illustrated in the comparison table (1), the three containers have their corresponding registry for images. Indeed, the ability of Singularity and uDocker to have multi-users accessed the system. Fundamentally, root permission is not required, and neither escalation is allowed during running a container. These represent the ideal container technologies for running scientific applications across distributed computing resources, such as high-performance Computing, and GRID computing infrastructures. They offer features focusing on

Table 1 Summary of containers technologies comparison [50]

	Docker	Singularity	uDocker
Community adoption	Enterprise	Scientific (DCI)	Scientific (DCI)
Isolation	Namespace, resource, network	Namespace, host network	LXC, Docker
Privilege model	Root Daemon	SUID	SUID
Trivial HPC install (one package, zero conf)	Yes with security implication,	Yes	Yes
Internal image build/bootstrap	No depend on upstream	Yes	Yes
Additional hardware	Maybe	No	No
Additional network configuration	Yes	No	No
Native support for GPU	No	Yes	Yes
Native support for MPI	No	Yes	Yes
Native support for Infiniband	Yes	Yes	Yes
Works with all schedulers	No	Yes	Yes
Admin can control and limit capabilities	No	Yes	Yes
File system access	Mount/volume	mount	Mount
Portability	Multi layered image	Single file image	Single file image
Storage	OverlayFS	ext3, ext4	OverlayFS

system security, by minimizing the number of virtualized namespaces, performing mobility, freedom, and performance. Thus, features are used to simplify their architectural design. Therefore, Docker has been very restricted, and not adopted in these infrastructures.

3 Container features

This section aims to provide an overview of container features, namely the container architecture, lifecycle, and orchestration. At the end of the section, we describe the proposed taxonomies of containerization technologies and provide a new one, which extends these taxonomies to cover concepts that emerged after their publication.

3.1 Container architecture

Since 2013, container-based [51] architecture models changed from software components-oriented architecture to service-oriented architecture [52]. Particularly, micro-services architecture is an implementation that inheriting concepts from service-oriented architecture "SOA", aims to build services independently over distributed systems. Container introduced as a concept composed of Unix-like pipelines [32]. The pipeline mechanism allows inter-process communication within Linux operating system. Micro-services are smaller autonomous components, encapsulated in an application that performs a specific task. Moreover, it encapsulates everything, considering the operating system, the runtime environments with all dependencies, packaged as one unit of application, ready for execution. Micro-services architecture has successfully attracted a lot of attention. They possess diverse opportunities related to creating a predictable environment for applications, introduced as services. Each service has a specific functionality as opposed to a monolithic architecture, application functionality usually wrapped into a single process. However, distributed computing [9], and cloud computing [53] platforms have adopted micro-services architecture, to deploy and scale [54] applications as services. They are structured as a set of loosely coupled services.

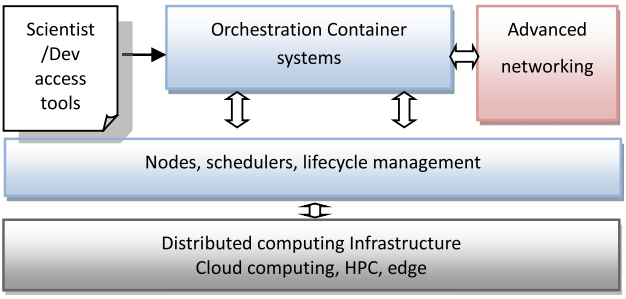

Fig. 2 Architecture-based container in computing system

This structure makes them easier to update, and simpler to manage. An application decomposes into multiple services, which are known as micro-services, delivered to users through the web technology, using an HTTP protocol mechanism. They communicate with each other using REST APIs [55].

According to [56], containerization can be considered as the ideal solution for micro-services-based applications, with the benefit of decomposing an application into smaller components, each component performs a specific task. This decomposition enables the parallelization of the processing using computational resources, which makes the application extensible and easily maintainable [57].

In Fig. 2, we present an overview of a container-based architecture model of computing systems. This architecture illustrates how to deploy container technologies for intensive application processing. It introduces fundamental layers that provide an architecture based on container environment, and their management systems that illustrate parallelization functionalities, to enable high-speed distributed processing of intensive scientific applications at a large scale.

3.2 Container lifecycle

Container lifecycle is referring to explore the states that are possible for the container. The container manager provides a framework offering a set of API. This framework allows easy management of the container's lifecycle, considering container creation, building, running, and maintaining. First, the developer has to create his new image as a template. Afterward, all operations are going to operate with this image created. As shown in Fig. 2, the entire lifecycle of the container, after the creation of the basic image, moves into the running state. Moreover, the container can have different states such as paused, killed, and stopped.

Container application implementation is primarily about consistency, flexibility, and scalability. Thus, to manage easily the container applications lifecycle [58], which means that is responsible to ensure that the system resources are efficiently utilized without downtime. As shown in Fig. 3, the container application lifecycle begins with a scientist or a developer that wants to implement his container image. Developers create the container-based image and hold everything inside the image,

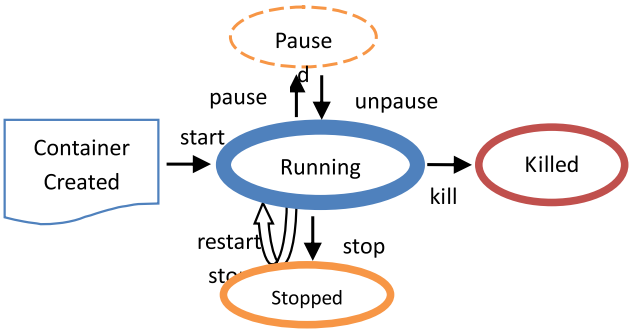

Fig. 3 Container operations lifecycle

including the software, the libraries with all dependencies. Hence, using different libraries than used by the host OS to reinforce the container security, and solving application conflicts between other environments. Then, they validate and push the container images into the registry, and finally, easily shared with IT Operations and throw them up in the production environment. The majors' benefits are to simplify the build, test, and deployment of the container workflow pipelines for DevOps environment, where developers can leverage third parties such as Jenkins to automate their Docker application releases.

3.3 Container orchestration

The container orchestration is a layer that interfaces with applications. It maintains service-level agreements, by scheduling containers in a cluster, choosing the optimal hosts and keeps the container running in the desired state. Nevertheless, the orchestration layer is required when the number of containers and devices increases regularly in the cluster. Containers orchestration is a platform-based micro-service that orchestrates computing, networking, and storage resources to support user workloads. Indeed, it is used to automate and manage procurement and deployment, allocation of resources, scaling, scheduling, load balancing, and securing interactions between containers, see Fig. 4. Moreover, container orchestration platforms allow organizations to rationalize application development through deploying the same application without needing to redesign it, configuring them according to the container which they will run, and, accelerating the process of delivering them, across clusters, or in cloud infrastructures [59]. In this case, the application runtime environment takes the form of jobs, where a job may define as a single containerized task. Furthermore, containerization became increasingly popular because of the following features provided by container technologies, including performance [21], isolation, scalability [36], portability [60], dependency, fault-tolerance, and load balancing [36]. Those advantages involve infrastructure services, ensuring distribution of the load between container instances and making applications run efficiently.

In this section, we discuss some container orchestration systems, including Docker Swarm, Mesos, Kubernetes, and Nextflow. Moreover, there are many other orchestration platforms such as Cloudify, Rancher, and Red hat Openshift, which use Kubernetes as a fundamental platform.

3.3.1 Docker swarm

Docker Swarm [61] is an open-source orchestration platform, originally is the native Docker orchestrator. Docker Swarm supports Docker container for the beginning on Linux, and after supported on Windows in 2017. Docker Swarm service is defined using a configuration file, written in YAML, to brin gup a pool of Docker instances across cluster nodes. Docker Swarm disposes of two categories of nodes, a manager, and a worker node. Indeed, the Swarm manager has the responsibility of managing

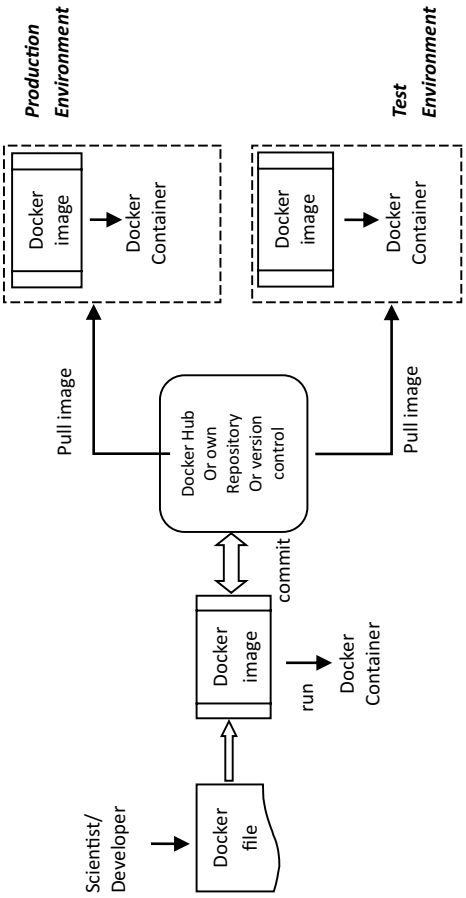

Fig. 4 Container application lifecycle [47]

related tasks to the delegation of process, membership, routing requests to worker nodes using the load balancing concept, and finally exposing services using IP address and port. Swarm nodes are composed of virtual or physical hosts managed by the Swarm manager, which is responsible to execute, control the deployment, manage the container lifecycle, and handle the replication of container applications. Therefore, the Docker engine supports the clustering functionalities, to build stacks of scalable production-grade applications. In this case, users may add Shiny-Proxy [62] layer to Docker Swarm. There are many features brought by Swarm, such as rolling updates, auto-recovery services, and applying security mechanisms much better than Docker-compose. This last, designed only for the development environment instead of the production environment.

Docker Swarm does not provide monitoring of resources utilization mechanism. Therefore, Bella et al. [63] analyzed the performance of the load balancing mechanism required to distribute and monitor the memory utilization in hosts.

3.3.2 Mesos

Mesos [15] is an orchestration and management platform developed to orchestrate applications using principles of the Linux kernel. It started with Linux cgroups containers and then supports Docker containers. Indeed, Mesos has three main components, namely 'Mesos Master' to manage resources negotiations amongst agents and frameworks, 'Mesos Agent' to execute tasks requested within available resources, and finally, 'Mesos Framework' to weave workloads with each other, pooling resources of all hosts to build up distributed computing systems efficiently. However, Mesos does not natively support service discovery functionality. Therefore, to address this lack, a third party such as Kubernetes or Swarm is involved. It takes a modular approach to group containers when dealing with other containers management. It allows the cluster-wide management of services running on its top. It maintains the execution of applications launched even if node failures, including Kubernetes, Swarm, Chromos, and Marathon [64]. Mesos-DNS integrated discovery service acts as a load balancer responsible to manage applications. Noting that due to Mesos features of redundancy and scalability, it is ideal to run applications that require large systems, such as Spark, Kafka, Hadoop, Elastic search, and Kubernetes. In addition, Mesos schedules the deployment of containers into clusters, by determining the best host for running containers. In [65], the authors adopted a scheduling policy based on two levels to schedule tasks into a node, and to allocate resources. Xiaolian et al. [43] verify the communication performance of Docker containers applications built on Mesos cluster. The authors show the resource allocations' influence on response time using the DRF (Dominant Resource Fairness) algorithm. As result, the authors prove that the system can ensure an adequate and feasible communication performance. Another work [66] shows system efficiency with higher performance when connecting a system based on workflow to container schedulers.

3.3.3 Kubernetes

Kubernetes (k8s) [16] built firstly by Google, then open-sourced in 2014 [67]. Kubernetes is a powerful orchestration platform, aims to automate the deployment of application services. These applications are set over multiple containers, scheduled, and managed at a large scale efficiently. The adoption of Google Kubernetes [14] has gained popularity to manage scientific workloads, due to the great features of deployment, flexibility, portability, and reproducibility [68]. Furthermore, Kubernetes provides frameworks to deploy and manage workloads inside clusters automatically. It is built out of one or multiple containers gathered in PODs. The POD is a fundamental building block in the Kubernetes cluster. It encapsulates one instance of running applications across clusters. The Kubernetes cluster is composed of Master and Worker nodes. Hence, the master node is the core component of the orchestration system, and the worker nodes perform well-defined services to run PODs. Indeed, Kubernetes takes control of running containers by automating their lifecycle management and securing workloads. Kubernetes can assists workload for load balancing by using basic monitoring, logging, and health checking. In addition, enables moving applications without redesigning them.

The Scheduling in Kubernetes uses containers as their primary unit of execution, and isolation. It decouples users from all details of the underlying computing infrastructure. It places containers on worker nodes automatically and recovering them from failure. Therefore, Kubernetes can apply a user remapping feature, which can rely upon defined users in a container to system-defined non-privileged user IDs. Hence, the controllers can manage a particular aspect of a cluster state, to regulate the state of a system. For example, they control the execution of containers in a cluster, or at a large scale. Nevertheless, it is necessary to make the current execution state come closer to that desired state.

Kubernetes become more prevalent, and his performance features become more crucial. It requires configuring the system, to match the application requirements, to the best concern. Thus, by choosing optimized images, configuring resources related to nodes, PODs. Also defining all Kubernetes features, by setting up the resource limits, ie., memory, CPU usage, Disk I/O, and network I/O. All these are about to give an efficient performance adjustment.

For instance, the main benefit of using Kubernetes is that it is proficient to automate the container's deployment and establish communications between them. To enhance an efficient integration between all micro-services, we have to monitor the performance of containers, micro-services interactions, trace user requests, and solve identified issues.

3.3.4 Nextflow

Nextflow [69] is a workflow management system "WfMS", designed for data-driven computational pipelines, developed by the comparative bioinformatics group at the Barcelona CRG (Centre for Genomic Regulation). Nextflow is designed to provide a portable, flexible, and reproducible workflow management system executed on multiple platforms, like HPC and cloud platforms. It supports several container

Table 2 Comparison of orchestration platforms

System	Container technology	Resource management		Infrastructure	System objectives	Isolation	
		Workload	Job composition			Access control	Container-centric
Kubernetes	Docker, rkt, CRI API implementations, OCI-compliant runtimes	All	Multiple co-located tasks	Virtualized, non-virtualized Elastic, manual, auto-scaling	Scalability	Restricted API access, Authentication, RBAC-based Authorization	Restricted execution, Hypervisor isolation
Swarm	Docker images	Long-running jobs	Co-located tasks	Virtualized, non-virtualized, Elastic, manual scaling	Scalability, availability, utilization, throughput	Authentication	Restricted Execution
Mesos	Mesos containers, Docker Singularity	All	Single task	Virtualized, Non-virtualized, Elastic, manual scaling	Scalability, Availability	Restricted API access, Authentication, RBAC-based Authorization	Restricted execution
Nextflow	Docker image Singularity	All	Multiple co-located tasks	Virtualized, non-virtualized, elastic, manual scaling	Scalability, Reproducibility	Restricted API access, Authentication, RBAC-based Authorization	Restricted execution

runtimes, such as Docker, Singularity, and Common Workflow Language (CWL) support, along with the integration of GitHub that allows reproducing any former of them, where they showed performances close to the local computing environment [70]. Moreover, it is more suitable to scale up or scale out of scientific workflows transparently. Google Pipelines APIs are used to manage computing services and allow the execution of containerized workloads. It provides parallelization features based on DSL (domain-specific language), to simplify writing computational pipelines to deploy workflows and process the huge amount of data that may alter into components independently.

3.3.5 Comparison

Table 2 presents a comparison of various features of different orchestration platforms, including Kubernetes, Swarm, Mesos, and Nextflow. This comparison allows administrators and researchers to choose the most appropriate tool for container orchestration. Docker Swarm is quite popular amongst developers. It offers fast deployments and simple features to get starting. However, it is not extensible with limited fault tolerance. Kubernetes is a powerful and highly extensible platform. Furthermore, Kubernetes, Mesos, and Nextflow platforms support services built up with more complex approaches than Swarm, used for production environments. Moreover, Mesos takes a distributed approach to combine multiple clusters to manage computing systems and cloud resources compared to Docker Swarm and Kubernetes.

4 Containerization taxonomy

In this section, literature taxonomies of containerization technologies are discussed. Then, a new one that covers and completes those proposed in the literature. There are some efforts in the literature to improve the understanding and classifying of container technology. Casalicchio and Lannucci [71] proposed a taxonomy of containers that structure the container landscape in subcategories, applications, orchestration, performance, and security. Many challenges are covered in each subcategory such as performance for scheduling, auto-scaling, availability, and security. The authors performed and evaluated containers performances using orchestration management systems to packaging containers for applications on different distributed computing infrastructures. Ernst et al. [72] proposed taxonomy based on the technological foundations of containers. They subdivided container technologies into two main building blocks: lifecycle management of containers and cluster management of containers across multiple hosts. The taxonomy provides a global overview of container components and it helps to compare, integrate, and analyze containerization and their respective orchestration system. Another taxonomy proposed in [73] describes a reference architecture that gives a global view of container ecosystems and their patterns that represents container units. The goal behind this taxonomy is to facilitate the work for those who are using container-based systems and perform security analysis of the whole system. Rodriguez and Buyya [74] present a

taxonomy that provides a classification of container orchestration systems, which can be applied for further research studies. This taxonomy identifies mechanisms that can use to meet challenges related to scalability, fault tolerance, and availability within efficient resource utilization. Maxime et al. [44] proposed a taxonomy related to container security that defense the infrastructure level. They analyzed some infrastructure approaches to improve the security level of containers. They classified frameworks required to set up and enforce security when data is transmitted between the container and the host kernel.

In these works, the authors introduced an architecture reference based on layers, application layer for submitting job users, cluster manager layer to managing clusters and orchestrating jobs, cluster of worker compute nodes layer and infrastructure layer for deploying compute nodes.

Due to the very rapid evolution of the container approach, most of the proposed taxonomies potentially became outdated in a short time. In this work, we introduce a taxonomy that completes the existing one by covering characteristics of containerization, discussed in the previews taxonomies [71–73], and [75]. Moreover, it illustrates container technologies challenges and puts in evidence concepts required to illustrate parallelism functionalities to scale scientific applications using container-based systems. Adding parallel functionality to the taxonomy will be beneficial for scientists for different application domains who are interested to easily deploy the required software efficiently across distributed computing infrastructure to scale their applications.

We proposed containerization taxonomy as shown in Fig. 5. It provides an adaptive sampling for container orchestration platforms designed for scientific purposes, which follows the container-based architecture model depicted in Fig. 2. This taxonomy defines a classification for all layers depicted to enable features needed to accomplish the processing of containers based on orchestration systems, designed for high-performance computing infrastructures. Then, itemize each layer with the appropriate classification that points to job submission, scheduling, security, lifecycle management, monitoring, and performance.

The proposed taxonomy aims to provide a classification for the whole container-based orchestration system for computing infrastructures. It divides container-based orchestration systems into three levels namely, application, orchestration system, and the computing infrastructures.

- The application layer encompasses tools to specify container configurations related to networking, availability, load balancing, and service discovery. These tools aim to provide transparent access to the distributed computing system, via protocols and resource binding. Thus, guaranteed access throw different trust services, such as Federated AAI, GUI access, Direct API access.
- The container orchestration systems layer provides orchestration functionalities, which are suitable for container cluster management. Also to specify the coherent units of containers such as PODs in Kubernetes, a single host in Docker-compose, and a cluster of hosts in Docker Swarm. They are non-trivial stack that provides diverse features which are the main system objectives such as scalability, fault tolerance, and flexibility. The security mechanism is important to

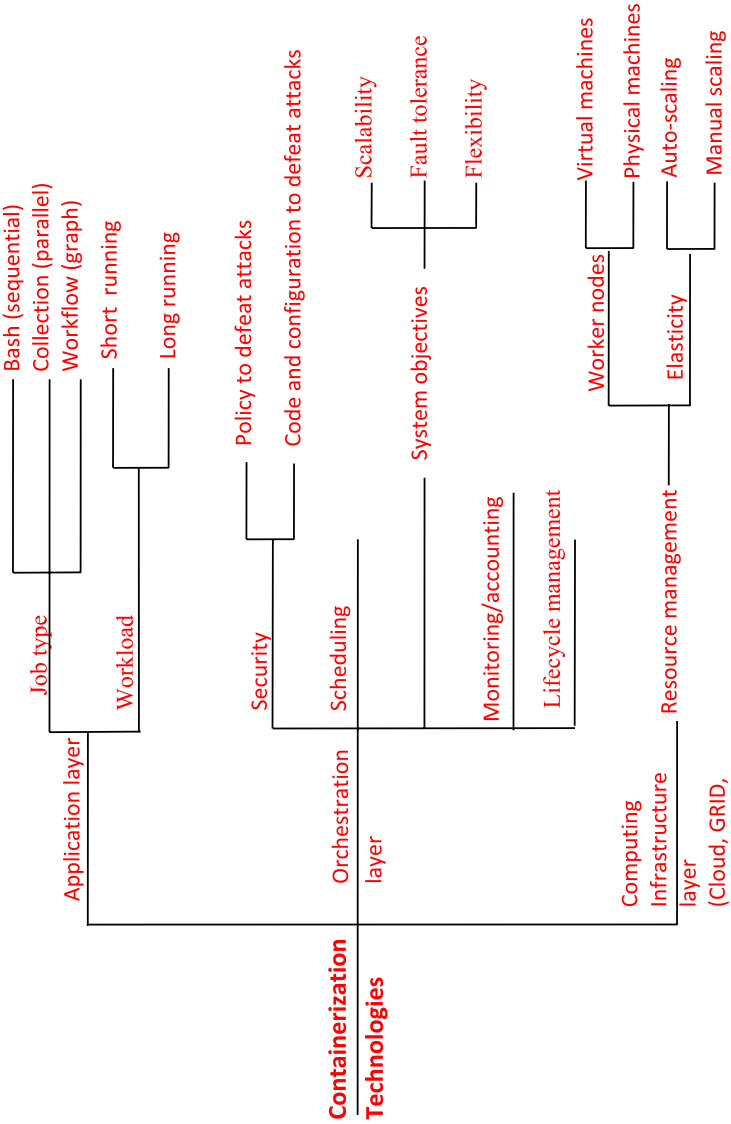

Fig. 5 Containerization taxonomy

improve performance by applying isolation mechanisms. Thus, mechanisms are suitable to defeats attacks from outside using some policies, and a code with a specific configuration. Additionally, advanced network tools are responsible for service discovery, monitoring, and accounting. The service discovery provides static access to different services, where they mapped dynamically to run containers, using a discovery agent to monitor the container's lifecycle, publish his state needed for observing and tracing a dynamic traffic routing. It monitors the containerized tasks deployed on nodes.

- Finally, the lifecycle management capability is important to distribute resources between computing nodes, scheduling of jobs, and execute containers. It considers the configuration of resources, including resource limits, resource granularity, and resource consumption. The scheduling of jobs consists of allocating necessary computing resources and controls the execution of jobs on those workloads. Usually, executing a job through a container requires some setup and cleanup for the lifecycle container processing. Also, it provides functionalities that enable scale-in and the scale-out of workloads. Moreover, it offers heterogeneities features of the underlying resources, to allocating computational resources in distributed environments, including locations, and resource types and sizes.
- The computing infrastructure is the physical level, composed of machines to manage computing processing and data management operations, considering the main features of resources management in the distributed computing platforms, including scalability, availability, flexibility, and fault tolerance. The distributed computing infrastructures can be physical machines and virtual machines on cloud (public or private). In addition, involved edge devices connected among them across the distributed infrastructure.

5 Applications of containerization

This part aims to extend previous containerization efforts made in different contexts that promote handling a large number of resources and providing fault tolerance to maintain computing systems. In this section, we provide some interesting container application domains, discussing features provided by relevant container technologies such as performance, isolation, scalability, portability, dependency, fault-tolerance, and load balancing. Besides, it illustrates the main properties in the container orchestration platform, which are orchestration, scheduling, and isolation.

5.1 Scientific computing

Scientific computing [76] is the power of computational processing, used by diverse scientific domains, especially life sciences [77], artificial intelligence, and machine learning [78]. Scientific computing handles a huge amount of intensive tasks as well as intensive data applications. Currently, containers are a hot research topic for scientific computing. An application built up from an image is containerized inside a container. However, containers encapsulate applications as well as all their

dependencies, isolated from the other processes running in the same host. The host can provide a view of all containerized resources, while containers share the same operating system kernel.

Scientific computing within the emergence of containerization offers to scientists a range of container solution to develop, manage, and applies isolation mechanism between application processes, and run them independently from the other running applications container. In addition, the orchestration platform like Kubernetes [16], enables automated deployment, scaling, and managing of containerized applications. As result, we can have an execution environment portable, secure, easy to scale, and simple to manage.

In the work [48], the authors introduced the INDIGO-Data Cloud platform that aims to facilitate access to e-Infrastructures, including distributed computing infrastructures (grid), cloud, and HPC clusters. It aims to enable porting scientific applications based on containers, using the uDocker tool. Moreover, the advent of "multicore" hardware processors and accelerators puts unprecedented pressure on the need for parallelism of applications to gain time processing by applying advanced implementations of parallelization (MPI, OpenMP, and Hybrid) and GPU support. Then, container technology is well suitable to deploy parallel scientific computing in distributed computing and shared storage systems. To take advantage of the fast network and leverage hardware acceleration, Lucas Benedicic et al. [60] deployed a detection system tool in a host of MPI libraries, which provides parallel implementations inside a container, to ensure the portability of the application. In particular, the advantage is to achieve native performance by providing efficient and scalable message parsing communications. Wang et al. [21] studied how to improve container performance aspects for aerial systems by executing containerized applications in native and computing environments.

On the other hand, modern scientific workflows [66, 79] represent another trend of scientific computing-intensive applications, which require sharing a large amount of data between tasks. A container-based virtualization mechanism is emerged within the scientific workflow to provide a portable, flexible, and reproducible workflow management system across HPC platforms.

5.2 Big data processing

Big data processing is the handling of data-intensive applications that require frameworks and sophisticated programming models to handle a mass of data in a reasonable amount of time. Humans and machines generate such data sets from sensors and social networks. However, it is tough to store, analyze, process, and visualize those datasets using traditional database approaches. Nevertheless, big data application processing faces challenges in dealing with those massive data sets, to do their processing above distributed computing nodes that require a high speed of the network, scalable storage systems, and large bandwidth of the memory. Besides, distributed computing systems must change the way hardware resources are managed to improve their performance.

Over the last years, containers are considered an exciting topic to manage intensive-data applications, which have gained more performance from distributed storage environments. Various scientific domains such as big data analytics [77], distributed processing, monitoring devices, security, and the Internet of Things (IoT) are producing a mass of data that carried out using frameworks capable to manage those data sets at a large scale. However, containerized parallel applications enabled for big data processing systems require MPI libraries and specialized hardware like graphical processing units (GPUs) and field-programmable gate array (FPGA). It provides scheduling of containerized jobs to speed up the analytics tasks. To guarantee that the security performances of HPC platforms based on containers for big data processing are near to the native computing environment.

Zhan et al. [6] considered the efficiency of hardware resources such as CPU and memory, used in scheduling big data applications to achieve better performance and higher utilization. The authors compared the execution of Spark Jobs between a container and virtual machine environments to improve performance in terms of makespans of running workloads, scaling, and convenient execution environment. The results mentioned that containers offer easy deployment and scalable environments of big data workloads compared to virtual machines.

Liu et al. [80] presented a study of a big data platform based on cloud computing container technology. They deployed the Kubernetes cluster to manage computing resources instead of YARN, where YARN manages cluster resources, including storage and execute data jobs efficiently at a large scale. This study poses challenges when running big data workloads that require scalability, performance, and promise fault-tolerance features. Thus, prove that those features guaranteed an easy deployment of scalable stream processing and batch processing for applications at a large scale efficiently.

A containerized distributed platform for high-performance computing and scaling big data applications used for execution and management on HPC nodes, illustrated in the work of Aldinucci et al. in [81]. The authors presented the SmartData@POLITO Big Data HPC centre that belongs to the Italian research network, which aims to offer a big data workloads, using Apache Mesos and Marathon for scheduling of containers, designed to be horizontally scalable to enhance performance, and do not affect users execution workspace, even during the execution phase.

Another work is done by Sergeev et al. in [78] studied distributed deep learning and machine learning used by TensorFlow instead of MLIB library afforded by Apache Spark machine learning. TensorFlow is a typical toolkit specialized to support machine and deep learning stacks in high-performance computing, it enables the build-up of singularity images for scientific computing.

Rao et al. [82] explored several stages of the big data system, data sources, data management, computing frameworks, and data analysis, categorized into various types, in-memory computing, and in-memory data models, set respectively by Apache Spark, Map-Reduce frameworks. Therefore, they provide an overview of advanced processing tools necessary for the analysis of massive datasets, using the stream, batch, distributed/cloud environment, container orchestration platforms, interactive querying, and data-ingestion. They compared those systems using machine learning (ML) tools, utilizing streaming processing, computing processing,

interactive analytical processing and graph processing at a large scale, by handling massive data sets that belong to real case studies. As result, they make an efficient observation that helps to use suitable tools properly for big data application processing in respect of operations of the delivery mechanism, replication, scalability, fault tolerance, storage, memory management, and other parameters, like throughput and latency measurements.

Recently, Blamey et al. in [83] introduce a study on containers that enhance using Apache Spark for stream processing instead of using Map-Reduce. Furthermore, Apache Spark provides a high throughput processing of streams, such as (Spark SQL as a high-level computationally tasks tool).

5.3 High-performance computing

High-performance computing (HPC) or "computing capacity" addresses the problems that require specific capacity. Usually, it allows highly optimized and tightly coupled centralized installations of processors. Moreover, interconnected with specialized hardware, using Infiniband, accelerators, and GPGPU, which is offering functionalities used to accelerate application processing. In the work of Aldinucci et al. [81], the authors give an overview of HPC4AI project that involves various high-performance computing centers that belongs to the Italian research Infrastructure. The HPC-center hosted HPC clusters provide a solution of adaptive container-based architecture, targeting distributed nodes interconnected by Infiniband. Those clusters shared the same Luster storage for HPC applications. They allow creating a customized execution environment using Singularity containers images. The container images are managed through a private Singularity HPC registry. Where users can upload images and download them.

Azab in [19] introduced a secure wrapper named "**Socker**" to deploy Docker containers on high-performance computing platforms. He added functionalities inside the wrapper for running Docker containers to solve issues related to enforcing membership of running Docker containers (for security) and limitations of resource consumption (CPU, memory) on queuing systems in production. Socker is suitable for running many tasks computing using MPI jobs. Wang et al. [21] studied how to improve container performance aspects for Aerial Systems by executing native and containerized applications in computing environments. Results show that container outperforms virtual machine in most of the performance aspects, due to the benefit of isolation of most hardware resource.

Piras et al. [84] proposed a method that brings a solution to deploy worker nodes in Kubernetes clusters. They used the Grid engine to deploy Bach jobs dynamically on the HPC system.

In [60], Lucas Benedicic discussed how computing systems could achieve scalability by leveraging modular architecture. He presented a container engine called "Sarus" deployed to launch container instances for HPC environments. The Sarus environment offers security within resources management and application deployment. Results show that performance and scalability are better than the native execution.

Zheng et al. [66] presented a model built to process container applications on HPC cluster with the Mesos orchestration platform. They connect Makeflow to Mesos platform to claim resources on demand. They adopt two levels of resource scheduling model, namely static and dynamic, which allow more flexibility.

We consider the work of Zhou et al. in [85]. The authors introduced an architecture based on Torque-Operator to bridge HPC workload managers and container orchestrator platforms. This architecture composed of an HPC cluster with Torque, and a Kubernetes cluster with WLM-Operator. Torque-Operator extends WLM-Operator with Torque. WLM-Operator allows the submission of Slurm jobs to Kubernetes cluster. This platform enables to schedule the execution of containerized and non-containerized jobs from HPC cluster to the cloud platform.

However, considering HPC as a specific environment is not opportunistic since various architectures and categories of distributed computing systems exist. Therefore, we discuss some of the high-performance computing categories, including grid, cloud, and cluster in the next paragraphs.

5.4 Grid computing

Grid computing [59] introduced by Ian Foster, as a federation of distributed heterogonous resources, cooperates among multi-institutional research communities, called virtual organizations. Virtual organization (VO) [86] is a community group, logically grouped, working on the same, or different topics of research to authorize them using the computing systems. The Grid computing infrastructure comprises compute, storage, and network resources, designed to bring the processing power and many of the technology capabilities, to solve computational problems that belong to a specific scientific domain. It provides toolkits to accredit users through authorizations and authentications. It allows data management, accounting, and resource monitoring, and designed to run massive and long-running computations over days and weeks. Moreover, it establishes application processing using job schedulers such as Slurm, LSF, and HTCondor. Grid computing is now known as the Distributed Computing Infrastructure (DCI).

Containers' applicability towards Distributed Computing Infrastructure recently considered by using orchestration platforms, to handles scheduling of containers across clusters. The European Grid Infrastructure Foundation project (EGI)[81]provides a computing and data infrastructure to carry out innovation and research in Europe. EGI funded to provide scientists access to 1,000,000 computing core and 740 PB of disk storage resources [87]. Indeed, it is suitable to run containers for intensive applications in batch systems on their HPC, and HPC systems [88]. Also, Open Science GRID (OSG) [89] is providing support for containers, including Docker and Singularity throw HTCondor [90] workload scheduler.

The Worldwide LHC Computing Grid (WLCG) involved solving major challenges of Large Hadrons Collider (LHC) [91] particle physics experiments at CERN [86]. This infrastructure allows access to a huge amount of distributed resource archives and can support powerful, complex, and tedious data analyses. Usually, data storage relies

on a shared file system such as NFS, DPM, Storm, and Luster, to leverage data information when scheduling computational tasks.

Nevertheless, deploying containers on distributed computing systems poses challenges due to the massive volume of high-energy physics (HEP) software stacks. Simone Mosciatti et al. [92], introduces container as a solution to run scientific applications on high-energy physics domain. They deployed Docker image used as a source image, converted after to Singularity image as a flat root file systems, and to Docker image as a compatible directory structure, and then published into CernVM-FS repository. Then, to evaluate their approach they deployed real uses-cases using LHC experiments in a production environment, integrating Singularity container. Results of this work show that the computing environment based on container technology is robust and near to the native performance, compared to that one based on virtual machines.

5.5 High throughput computing HTC

In particular, high throughput computing (HTC) [93] is another category of computing infrastructure that performs a lot of computing power to address problems that require a large amount of computation time. HTC is the efficient execution environment of a large number of loosely coupled tasks. It offers storage capacity and analysis support to process a huge amount of experimental data. For instance, recently, CERN [86] started to emerge Singularity containers among HTC (high throughput computing) computational infrastructure, which has played an important role in deploying complex software inside containers. Results of integration show that HPC resources based on container technology are robust and near to the native performance, compared to that one based on virtual machines [94].

5.6 Cloud computing

Cloud computing is defined by Ian Foster [59] as a "large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically scalable, managed computing power, storage, platforms, and services are delivered on-demand to external customers over the Internet". Cloud computing allows deploying and scale services on-demand, by selecting preconfigured virtual appliances, with complete control over computing resources. Cloud computing comprises three fundamental layers: (1) Infrastructure as a Service layer (IaaS), which provides virtual appliances, including, computing, networking, and storage. (2) Platform as a Service layer (PaaS), which provides platforms to manage, develop, and deploy applications. (3) Software as a Service layer (SaaS), which provides software to end-users. Maenhaut et al. [31] introduced the adoption of container technology in traditional cloud computing (private, public, and hybrid). According to them, containerization gives advantages that guarantee security, performance, and elasticity concepts, for High-performance computing during scheduling [2] of compute nodes and managing resources efficiently. Then, cloud computing has adopted the usage of container technologies in their infrastructures through the Containers as a Service (CaaS) solution [53].

European Grid Initiative Foundation infrastructure (EGI) [95], implements a federated cloud to access cloud provider's resources that belong to National Resources Network initiatives "NREN", based on cloud providers resources. Recently, it delivers Container as a Service "CaaS" [96]for advanced computing power and storage for data-intensive science research applications using EC3 (Elastic Cloud Computing Cluster) [97], which allows the creation of elastic virtual clusters as a service [98]. Moreover, EGI enables using the INDICO-Data cloud [99] platform as a PaaS orchestrator to build an application encapsulated as a container, and launch it in the Cloud, from a web interface (Future Gateway). INDICO-Data cloud offers services as service (SaaS) to deploy specific services relevant for scientific computing regarding authentication, workload, and data management. It provides too, another platform as a service PaaS for CMS experiment, called Dynamic on Demand Analysis Service (DODAS), to instantiate container-based clusters to execute software applications and distribute the load across cloud providers.

Several popular cloud providers have invested in Container technologies. For instance, Amazon Web Services (AWS) [100] provides Elastic Compute Cloud (EC2) as compute instances, using Elastic Container Service (ECS) to build Kubernetes (EKS) clusters since 2017. Google Cloud Platform (GCP) [101] provides Google Compute Engine (GCE) for users to create and run virtual machines on a large scale. Moreover, Google Cloud Storage is used to store large data objects, considering consistency, durability, high availability, and scalability features. Besides, to benefit from Google's network capabilities that offer high performance and automatic dealing of load balancing issues. Microsoft Azure [102] provides Hyper-V containers, where each container runs inside of a special virtual machine. Red Hat's OpenShift [17] is a leader and active builder of Kubernetes. It provides additional tools for automatically deploying, building, updating container infrastructure. It offers monitoring and security features for cluster management and resource provisioning.

5.7 Internet of Things "IoT"

The Internet of Things (IoT) is a new paradigm. It is a recent trend of traditional cloud computing infrastructure that integrates not only traditional computers but also many kinds of things, lightweight devices, or a large number of technologies manage objects around us like mobile edge (smartphone) In the real world, IoT embedded many of network sensors, and intelligent devices, which generates a huge amount of data. These massive volumes of data generated [80] by IoT devices require storage capabilities, computational resources, and wireless communication of the distributed computing infrastructures. Where computing resources might be closer to IoT devices to reduces latency and provides high bandwidth of the network.

In another work, Chang et al. [103] depicted a virtualized system based on Open5GCore using virtualization techniques to simulate IoT devices. The authors analyze the performance of the 5G network throughput, response time, CPU utilization, and memory usage in this system. Their results show that the use of the Docker

container is the most appropriate solution and feasible because of its low virtualization overhead and rapid deployment at a large scale.

Aruna et al. [36] studied scalability in the Internet of Things (IoT) system using container technologies to manage resources, integrating many connected devices. For this aim, they introduced a new paradigm that enables IoT scalable architecture. Results show that IoT based on the model may extend the network; it expands the scalability and improves the efficiency of services related to CPU, memory, and I/O performance.

In the work [104], the authors tested the performance of Internet of Things (IoT) applications within computational processing and storage capabilities on top of Fog computing based on a cluster of SBC (Small Board Computers) devices. They propose a solution based on Linux containers and an orchestration platform. They tested wireless communication in Fog computing. The comparison shows that the Internet of Things and its applications based on container orchestration platforms (Docker swarm end Kubernetes) can support running more pods of containers than cores in SBC devices. However, Docker Swarm outperforms Kubernetes in terms of the use of resources and processing time. But Kubernetes outperforms Swarm with higher performance, due to the auto-scaling to perform the load balancing features offered by Kubernetes. As result, IoT applications showed an improved performance.

5.8 Fog and edge computing

Fog computing [104] is a paradigm, introduced by Cisco Systems Inc leveraging heterogeneous nodes such as devices, gateways, and computers. It provides faster reaction with low overhead of storage and networking services, between devices [105]. Moreover, Fog computing extends traditional cloud computing architecture to the edge of networks, to have better flexibility to scale servers. Both of them have brought a new era to design, deploy IoT applications, and distribute them through FoT nodes. While Fog nodes provide tools to manage storage and data processing into devices, considering, resource allocation, monitoring, and security [36].

Edge computing (EC) [106] tends to push computing applications, data processing, and services away from centralized cloud data center architectures to edges belongs to the underlying network, to save network bandwidth.

Rafael et al. [104] have introduced a study among Fog computing using SBC cluster (Small Board Computers) devices. The authors proposed a solution based on Linux containers and orchestration platforms such as Swarm and Kubernetes, to carry out the scheduling, the processing, and manage storage capabilities on top of two types of clusters (Homogenous and Heterogeneous). They perform a comparison between them using both communications wired and wireless environments. The authors collect experimental tests related to the monitoring of devices' status, measuring their memory and CPU availability. Results showed in general an improved performance in terms of execution time, latency, and throughput network.

For instance, in Kaur et al. [107], KEIDS's system model for Kubernetes deployment in an IoT ecosystem equipped with edge-cloud functionality is a competent controller. KEIDS aims to minimize the energy usage of edge-cloud in IoT devices

and solve the problem of application tasks scheduling on available nodes in less time, using linear programming based on multi-objective optimization. An evaluation of this system indicates that its superior performance in real-time.

In reference [108], the authors study the performance and scalability bottleneck of data movement (MPI, asynchronous) on HPC environment-based virtualization. They designed a model based on the range of two-stage methods to enhance the performance of the HPC system. Their results show that the performance of this model can improve the performance needed for running a real scientific application.

5.9 DevOps

Development and Systems Operations "DevOps" [109] is a community interested in application development and programming. In recent years, micro-services architecture is considered an exciting topic besides the community of DevOps. Developers became more productive within programming applications through micro-services format deployment. Then, the final application is obtained by the composition of micro-services components.

Furthermore, container orchestration engines, namely Kubernetes, Swarm, and Mesos are involved in DevOps infrastructure management to provide portability and dependency features. Containers can run everywhere, from development to production. Each developer should develop a component separately, defining the standard image format, and the run-time environment for containers, without interfering with the other application components. Usually, the developer starts to develop an application from a base image format, which contains the operating system and default libraries. Although, the image layered by adding customized libraries and data to the code source. Moreover, the multilayered application image built is read-only except for the last layer who is persistent.

6 Performance metrics of containerization

Many works [6, 32] have been compared to the performance of different virtualization concepts. In this section, we discuss performance parameters considered on container technologies over computational environments, as well as how to evaluate the best performance of using containerization along with virtualization.

6.1 Computing performance

Computing performance attracts the measurement of performance that encompasses testing **CPU**, **Memory** throughput, and **Disk I/O** resources. To do that we need to measure the number of operations, the completion time of tasks performed by the system, using benchmarks tools.

Many works [56] that have been done testing containers overhead by comparing them to virtual machines such KVM, as seen in [32], the authors measured the CPU overhead of containers, including Docker, LXC, and Singularity. Testing CPU performance provided by using the 'Sysbench' benchmark across the industry application domain. Although this work showed that the CPU performance of containers is better than virtualization and almost the same as native performance. Indeed, for the work of Wang et al. in [21], where authors studied container performance aspects for Aerial Systems by executing native and containerized applications in computing environments.

Recently in the work [110], they discussed applying Sysbench, Apache and Phoronix benchmarks tools, to evaluate performances of containerized applications in computing environments such as HPC cluster. Therefore, most of the results show that container performance is better than virtual machines and almost the same as the native. Because they use fewer resources, and while they share the same OS, their boot-up time is short and their creation time is within seconds.

Rafael Fayos-Jordan et al. in [104] compared container orchestration platforms with different communication topologies (wired/wireless) on SBC devices, in terms of use of resources, and processing time, show that the model deployed on the Internet of Things devices and its applications has the lowest power consumption and the lower throughput.

In paper [6], Zhan et al. provided container performance for Big data processing using different big data workloads, considering scheduling, packaging, and resource access problems. According to their results, container achieves low overhead with using CPU and memory, and better scalability than virtual machines.

In[36], Aruna et al. studied the ability to scale an Internet of Things (IoT) system using containers to manage resources, including many connected devices. As result, the performance of the extended network improves the scalability and efficiency of services related to CPU, memory, and I/O.

6.2 Isolation performance

Containers provide a virtualization mechanism that allows running an application and its dependencies in an isolated environment (user-space) instead of creating virtual machines. In resource isolation, the failure is due to the excessive resource utilization by the process of one container and cannot affect the others. As seen in [56], the authors evaluated the performance of an isolated execution environment, by executing containers with collocated micro-services and comparing them to a baseline container running one micro-service, to identify the interference between them with minimal degradation performance. Their results show that the best performance of micro-services is achieved within the deployment of micro-services in separate containers.

Container measurement as discussed in the work of Wang et al. [21] used to refer to one or more processes running in an isolated fashion, where the OS-kernel features enforce the isolation. There are many levels of security applied to measure isolation performance in, considering user-pace, data, and network security. However,

Linux containers provide an isolated execution environment in High-performance computing, and this by relying on Docker Container for the execution and the resources management on HPC nodes, illustrated in the work of Aldinucci et al. in [81].

In reference [36], the authors analyzed the performance of low-cost devices of the Internet of Things (IoT) ecosystem on a cloud computing environment, with different orchestration platforms, including Kubernetes, Docker Swarm and Mesos. In this work, the authors conclude that the use of Kubernetes is superior to all other orchestration platforms including deployments of complex IoT applications.

Vazhkudai et al. in [111] introduce containerization in the field of high-performance computing data-intensive applications using CORAL2 data science benchmark, to assess the performance of the CPU, GPU, memory, I/O, and deep learning workloads. Thus, to allow performing the provisioning of resources efficiently and provides low overhead performance isolation.

6.3 Network performance

About [32] network performance measurements in container systems are satisfying, using a network benchmark IPerf tool. On another hand, there are many other benchmarks used such as Linpak, Stream, Fio, and Netperf [56].

In terms of network performance throughput, in the field of Internet of Things 'IoT' devices, Chang et al. [103] assess an evaluation of the performance of the Open5GCore based on container virtualization techniques, show that the usage of Docker container is the most appropriate solution, and feasible in terms of response time, CPU utilization, and memory usage in this system. This is because of its low overhead, and it allows rapid deployment at a large scale. Also, network performance refers to the ability to perform and maintain the reliability of the network in the computing systems based on containers that require features such as scalability metrics as seen in [30].

We can also find interesting contributions in references [36, 60] to network performance and platforms. The authors introduced a solution to handle computational resources using network interface from devices involved in Fog and Edge environments, they conclude that this variability is null, and cannot affects the communication between devices.

In [108], Hang et Lu tried to improve the performance and scalability of parallel and asynchronous data movement in HPC system based on virtualization. To enhance the performance degradation and scalability bottleneck of I/O virtualization stacks, they designed a BADM middleware based on v-switches to reduce the bandwidth of the virtual network and collective data movements. Their results improve the performance of executing a real scientific application.

Also, Arnaldo Pereira et al. [112] evaluates the containers performance, running on various Kubernetes platforms across different mainstream Cloud environments including Amazon Elastic Container Service for Kubernetes (EKS), Azure Kubernetes Service (AKS), and Google Kubernetes Engine (GKE), using CEEM (Cloud Evaluation Experiment Methodology) to support traceable and reproducible

experimental evaluation results. Their results show that for CPU and the network-intensive containerized applications provided the best performance.

7 Conclusion and future work

Both containers and virtual machines technologies enable users to define and build their software environments and then run them on top of various resources in a portable, reproducible way. This paper presents a thorough investigation of containerization technologies that are widely used. We have identified the main features of container technology. Then, we have discussed already existing taxonomies and proposed a taxonomy that covers most of the existing ones. We described containers architecture for computing systems by illustrating and discussing some details for application domains. The work has shown that is important to understand the capabilities and techniques available for a given containers-based solution as well as the characteristics of workloads to optimize systems. At this moment, the container approach is at the heart of the modern computing infrastructure as it avoids several of the challenges related to intricate execution environment dependencies that are often in conflict with other components of the application workflows. Containers have been adopted by several initiatives and approach a default technology like Cloud Native, Dev/Ops. With containers, it is possible to construct scalable architecture composed of a large number of services (micro-services), IT companies like Google, Microsoft, Netflix etc., are nowadays relying on container technology in their production environment.

While the adoption of container technologies is growing, the community still has many questions about such technologies. To address the shortcomings of existing container-based platforms and solutions, further research must be done regarding best practices, foundational functionalities, standards and tools. In the following, we highlight some interesting future research directions on the containerization technologies field. Future research in this field is highly related to the promising environments in the field. Thus, we discuss such new technologies and related challenges.

In another vein, effort should be made to track the development of frameworks. However, the implementation of new solutions in computing infrastructures at a large scale leads to a set of questions that need to be addressed in future research: Containers are known for their lightweight footprint and their easy deployment, but there is always a better way to do things. The research community will not stop at container technology. New emerging, technologies are already on the horizon; unikernels, which look like'improved containers' but with better security and performance, are merging very quickly. Unikernels offer better isolation and run on a minimalistic operating system, which is tailored, to the application [113]. Another recent research initiative around container technology is the combination with the serverless approach tools like AWS Fargate, KNative combine the serverless approach and containers to reduce the cost to manage the container-based system.

Acknowledgements This work was partially supported by the General Directorate of Scientific Research and Technological Development (DGRSDT, Algeria), under the PRFU project (ref: C00L07UN060120200003).

References

1. Bermejo B, Juiz C, Guerrero C (2019) Virtualization and consolidation: a systematic review of the past 10 years of research on energy and performance. J Supercomput 75(2):808–836
2. Menouer T, Darmon P (2019) Containers scheduling consolidation approach for cloud computing. In: Esposito C, Hong J, Choo KK. (eds) Pervasive Systems, Algorithms and Networks. I-SPAN. Communications in Computer and Information Science, vol. 1080
3. Zheng L, et al. (2017) Performance overhead comparison between hypervisor and container based virtualization, IEEE 31st International Conference on Advanced Information Networking and Applications (AINA) pp. 955–962
4. Chae M, Lee H, Lee K (2019) A performance comparison of linux containers and virtual machines using Docker and KVM. Cluster Comput 22:1765–1775. https://doi.org/10.1007/s10586-017-1511-2
5. Yu B, Tian J, Ma S, Yi S, Yu D (2011) Gird or cloud? Survey on scientific computing infrastructure, IEEE International Conference on Cloud Computing and Intelligence Systems, Beijing, pp. 244–249
6. Zhang Q, Liu L, Pu C, Dou Q, Wu L, Zhou W (2018) A comparative study of containers and virtual machines in big data environment. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). San Francisco, CA, pp 178–185. https://doi.org/10.1109/CLOUD.2018.00030
7. Linux containers LXC, [Online] available March 2020: https://linuxcontainers.org/
8. Eric Chiang Containers from Scratch. [Online], available March 2020: https://ericchiang.github.io/post/containers-from-scratch/#container-file-system
9. Campeanu G (2018) A mapping study on microservice architectures of Internet of Things and cloud computing solutions, 2018 7th Mediterranean Conference on Embedded Computing (MECO), Budva, pp. 1–4, doi: https://doi.org/10.1109/MECO.2018.8406008
10. OpenVZlinu containers, [online] available April 2020: http://openvz.org/
11. Docker, [Online] available April 2020: https://docs.docker.com
12. Singularity, [Online], available April 2020: https://www.sylabs.io/docs/
13. uDocker. [Online], available April 2020: https://github.com/indigo-dc/udocker
14. Beltre AM, Saha P, Govindaraju M, Younge A, Grant RE (2019) Enabling HPC workloads on cloud infrastructure using kubernetes container orchestration mechanisms. Paper presented at 2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms forIsolated Environments in HPC(CANOPIE-HPC), Denver, CO, USA, 2019, pp. 11-20. doi: https://doi.org/10.1109/CANOPIE-HPC49598.2019.00007
15. The apache software foundation. Mesos, apache. [Online], available April 2020: http://mesos.apache.org/
16. Kubernetes, [online], available July 2020 : https://kubernetes.io/
17. RedHat Openshift.[Online], available April 2020: https://www.redhat.com/en/technologies/cloud-computing/openshift
18. Pahl C, Brogi A, Soldani J, Jamshidi P (2019) Cloud container technologies: a state-of-the-art review," in IEEE Transactions on Cloud Computing, 1 July–Sept. 2019, vol. 7, no. 3, pp. 677–692
19. Azab A (2017) Enabling docker containers for high-performance and many-task computing," IEEE International Conference on Cloud Engineering (IC2E), Vancouver, BC , pp. 279–285, doi: https://doi.org/10.1109/IC2E.2017.52
20. Lingayat A Badre RR, A. K. Gupta AK (2018) Integration of linux containers in openstack: an introspection. Indones J Electr Eng Comput Sci. Vol. 12, no. 3
21. Wang B, Xie J, Li S, Wan Y, Fu S, Lu K (2018) Enabling high-performance onboard computing with virtualization for unmanned aerial systems", 2018 International Conference on Unmanned

Aircraft Systems (ICUAS), Dallas, TX, 2018, pp. 202–211.doi: https://doi.org/10.1109/ICUAS.2018.8453368
22. VMware_paravirtualization. [Online], available April 2020: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf
23. Bazm M, Lacoste M, Südholt M et al (2019) Isolation in cloud computing infrastructures: new security challenges. Ann Telecommun 74:197–209
24. VMWare. [Online], available April 2020: http://www.vmware.com
25. Xen. [Online], available April 2020: https://xenproject.org/
26. KVM "Kernel based Virtual Machines". [Online] Available April 2020: https://www.redhat.com/fr/topics/virtualization/what-is-KVM
27. Wei M, Lin Y, Lee C (2019) Performance optimization for InfiniBand virtualization on QEMU/KVM," 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Sydney, Australia, 2019, pp. 19–26
28. Masdari M, Zangakani M (2019) Green cloud computing using proactive virtual machine placement: challenges and issues. J Grid Computing. https://doi.org/10.1007/s10723-019-09489-9
29. Sultan S, Ahmad I, Dimitriou T (2019) Container security: issues, challenges, and the road ahead. IEEE Access 7:52976–52996. https://doi.org/10.1109/ACCESS.2019.2911732
30. Stephen S et al Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. SIGOPS Oper. Syst. Rev., pp 275–287. ISSN 0163–5980. https://doi.org/10.1145/1272998.1273025
31. Maenhaut P, Volckaert B, Ongenae V et al (2020) Resource management in a containerized cloud: status and challenges. J NetwSyst Manage 28:197–246
32. Á. Kovács 2017 "Comparison of different Linux containers," 2017 40th International Conference on Telecommunications and Signal Processing (TSP), Barcelona, pp. 47–51, doi: https://doi.org/10.1109/TSP.2017.8075934
33. Marcel (2018) Performance evaluation of mikroTik-based virtual machine for small-scale network virtualization on VMware Platform. In: 2018 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC), 2018, pp 154-158. https://doi.org/10.1109/ICCEREC.2018.8712000
34. Lingayat A, Badre RR, Kumar Gupta A (2018) Performance evaluation for deploying docker containers on baremetal and virtual machine. In: 2018 3rd International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, «pp. 1019–1023, doi: https://doi.org/10.1109/CESYS.2018.8723998
35. Openstack, Cloud operating system [Online], available June 2020: https://openstack.org
36. Aruna K, Pradeep G (2020) Performance and scalability improvement using IoT-based edge computing container technologies. SN COMPUT SCI 1:91. https://doi.org/10.1007/s42979-020-0106-9
37. Barika M, Garg S, Zomaya AY, van Lizhe Wang A, Moorsel, Rajiv R (2019) Orchestrating big data analysis workflows in the cloud: research challenges, survey, and future directions. ACM Comput Surv 52:1–37. https://doi.org/10.1145/3332301
38. Socker: A wrapper for secure running of docker containers on slurm, A. Azab, [online] Available June 2020: https://github.com/unioslo/socker
39. Raicu I, Foster IT, Zhao Y (2008) Many-task computing for grids and supercomputers", Many-Task Computing on Grids and Supercomputers 2008. MTAGS 2008. Workshop on, pp. 1–11
40. Dominic L, Sukhpal SG, Peter G (2019) PRISM: an experiment framework for straggler analytics in containerized clusters. In Proceedings of the 5th International Workshop on Container Technologies and Container Clouds (WOC '19).2019, pp. Association for Computing Machinery, New York, NY, USA, 13–18
41. Chen J et al. (2018) Build and execution environment (BEE): an encapsulated environment enabling HPC applications running everywhere," 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 1737–1746, doi: https://doi.org/10.1109/BigData.2018
42. Smith JE, Nair R (2005) Virtual machines: versatile platforms for systems and processes. The Morgan Kaufmann Series in Computer Architecture and Design Series. Morgan Kaufmann Publishers; 2005
43. Li X, Jiang Y, Ding Y, Wei D, Ma X, Li W (2010) Application research of docker based on mesos application container cluster," 2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL), 2020, 476–479, Doi:https://doi.org/10.1109/CVIDL51233.2020.00-47
44. Sergei A, Bohdan T, Franz G, Thomas K, Andre M, Christian P, Joshua L, Divya M, Dan O'Keeffe, Mark L. Stillwell, David G, David E, Rüdiger K, Peter P, Christof F (2016) SCONE: secure Linux

containers with Intel SGX. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16). USENIX Association, USA, 689–703
45. Docker Hub. [online], available 20 April 2020: https://hub.docker.com/
46. Hu G, Zhang Y, Chen W (2019) Exploring the performance of singularity for high performance computing scenarios. In: 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 2019, pp. 2587–2593. https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00362
47. Zhang J, Lu X, Panda DK (2017) Is singularity-based container technology ready for running MPI applications on HPC clouds", Proceedings of the l0th International Conference on Utility and Cloud Computing,
48. Salomoni D et al (2018) INDIGO-DataCloud: aplatform to facilitate seamless access to e-infrastructures. J Grid Comput. 163:381–408
49. Gomes J, Bagnaschi E, Campos I, David M, Alves L, Martins J, Pina J, López-García A, Orviz P (2018) Enabling rootless Linux Containers in multi-user environments: the udocker tool. Comput Phys Commun 232:84–97. https://doi.org/10.1016/j.cpc.2018.05.021
50. Kurtzer GM, Sochat V, Bauer MW (2017) Singularity: scientific containers for mobility of compute. PLoS ONE 12(5):e0177459. https://doi.org/10.1371/journal.pone.0177459
51. Silva V, Kirikova M, Alksnis G (2018) Containers for virtualization: an overview. Appl Comput Syst. 23(1):21–27
52. De Lauretis L (2019) From monolithic architecture to micro-services architecture. 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Berlin, Germany, pp. 93–96, doi: https://doi.org/10.1109/ISSREW.2019.00050
53. Yang M, Huang M (2019) An micro-services-based openstack monitoring tool. In: 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, pp. 706–709, doi: https://doi.org/10.1109/ICSESS47205.2019.9040740..
54. Wilhelm H (2016) Micro-services for Scalability: Keynote Talk Abstract. In Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (ICPE '16). Association for Computing Machinery, New York, NY, USA, 133–134
55. Li L, Tang T, Chou W (2015) A REST service framework for fine-grained resource management in container-based cloud," 2015 IEEE 8th International Conference on Cloud Computing, New York, NY, pp. 645–652, Doi: https://doi.org/10.1109/CLOUD.2015.91
56. Jha DN, Garg S, Jayaraman PP, Buyya R, Li Z, Ranjan R (2018) A holistic evaluation of docker containers for interfering micro-services. In: 2018 IEEE International Conference on Services Computing (SCC), San Francisco, CA, pp 33–40. https://doi.org/10.1109/SCC.2018.00012
57. Sampaio AR, et al. (2017) Supporting microservice evolution. 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, pp. 539–543, doi: https://doi.org/10.1109/ICSME.2017.63
58. Cesar de la Torre C (2016) Containerized docker application lifecycle with microsoft platform and tools
59. Foster I, Zhao Y, Raicu I, Lu S (2008) Cloud computing and grid computing 360-degree compared," 2008 Grid Computing Environments Workshop, Austin, TX, 2008, pp. 1-10, doi: https://doi.org/10.1109/GCE.2008.4738445
60. Benedicic L, Cruz FA, Madonna A, Mariotti K Sarus(2019) Highly scalable docker containers for HPC systems. Benedicic L, Cruz FA, Madonna A., Mariotti K , ISC high performance 2019. Lecture notes in computer science, vol 11887. Springer, Cham, https://doi.org/10.1007/978-3-030-34356-9_5
61. Menouer T, Darmon P (2019) Containers scheduling consolidation approach for cloud computing. In: Esposito C, Hong J, Choo KK (eds) Pervasive systems, algorithms and networks. I-SPAN. Communications in computer and information science, vol 1080, Springer, Cham. https://doi.org/10.1007/978-3-030-30143-9_15
62. Perampalam P, Dick FA (2020) BEAVR: a browser-based tool for the exploration and visualization of RNA-seq data. BMC Bioinformatics 21:221
63. Bella MRM, Data M, Yahya W (2018) Web server load balancing based on memory utilization using docker swarm. In" 2018 International Conference on Sustainable Information Engineering and Technology (SIET), 2018, pp. 220-223, doi: https://doi.org/10.1109/SIET.2018.8693212
64. Marathon, orchestration tool for Mesos. [Online], availableJune 2020: https://mesosphere.github.io/marathon

65. Saha P, Beltre A, Govindaraju M (2018) Exploring the fairness and resource distribution in an apache mesos environment. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, 2018, pp. 434–441, doi: https://doi.org/10.1109/CLOUD.2018.000
66. Zheng C, Tovar B, Thain D (2017) Deploying high throughput scientific workflows on container schedulers with makeflow and mesos," 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 2017, pp. 130-139, doi: https://doi.org/10.1109/CCGRID.2017.9
67. Bisong E. (2019) Containers and Google Kubernetes Engine. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform. Apress, Berkeley, CA
68. Medel V, Tolosana-Calasanz R, ÁngelBañares J, Arronategui U, Rana OF (2018) Characterising resource management performance in Kubernetes. Comput Electr Eng 68:286-297. https://doi.org/10.1016/j.compeleceng.2018.03.041
69. Nexflow [Online], available Jully 2020: https://www.nextflow.io/
70. Larsonneur E, Mercier J, Wiart N, Floch EL, Delhomme O, MeyerV (2018) Evaluating workflow management systems: a bioinformatics use case. In: 2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Madrid, Spain, 2018, pp. 2773–277, https://doi.org/10.1109/BIBM.2018.8621141
71. Casalicchio E, Iannucci S. The state-of-the-art in container technologies: application, orchestration and security. Concurrency Computat Pract Exper. 2020; e5668. https://doi.org/10.1002/cpe.5668
72. Ernst D, Bermbach D, Tai S (2016) Understanding the container ecosystem: a taxonomy of building blocks for container lifecycle and cluster management. Retrieved from the the: Proceedings of WoC. IEEE
73. Madiha HS, Eduardo BF (2018). A reference architecture for the container ecosystem. In Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES 2018). Association for Computing Machinery, New York
74. Rodriguez MA, Buyya R (2019). Container-based cluster orchestration systems: ataxonomy and future directions. Softw Pract Exp, 49(5), 698–719
75. Bélair M, Laniepce S, Menaud J-M (2019) Leveraging kernel security mechanisms to improve container security: a survey. In Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES '19). Association for Computing Machinery, New York, NY, USA, Article 76, 1–6
76. Jenkins J, Shipman G, Mohd-Yusof J, Barros K, Carns P, Ross R (2017) A case study in computational caching micro-services for HPC. In: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lake Buena Vista, FL, 2017, pp
77. Becker M. et al. (2020) Scaling genomics data processing with memory-driven computing to accelerate computational biology. In: Sadayappan P, Chamberlain B, Juckeland G, Ltaief H (eds) High performance computing. ISC High Performance 2020. Lecture Note
78. Alexander S, Del Balso M (2018) "Horovod: fast and easy distributed deep learning in TensorFlow." arXiv preprint. https://arxiv.org/abs/1802.05799
79. Vahi K et al (2019) Custom execution environments with containers in pegasus-enabled scientific workflows. In: 2019 15th International Conference on eScience (eScience), pp 281–290. https://doi.org/10.1109/eScience.2019.00039
80. Liu W, Fan W, Li P, Li L (2018) Survey of big data platform based on cloud computing container technology. In: Barolli L, Terzo O (eds) Complex, intelligent, and software intensive systems. CISIS. Advances in intelligent systems and computing, vol 611. Springer, Cham. https://doi.org/10.1007/978-3-319-61566-0_90
81. Aldinucci M et al (2018) HPC4AI: an AI-on-demand federated platform endeavour. In: Proceedings of the 15th ACM International Conference on Computing Frontiers (CF '18). Association for Computing Machinery, New York, NY, USA, pp 279–286. https://doi.org/10.1145/3203217.3205340
82. Rao TR, Mitra P, Bhatt R et al (2019) The big data system, components, tools, and technologies: a survey. Knowl Inf Syst 60:1165–1245. https://doi.org/10.1007/s10115-018-1248-0
83. Blamey B, Hellander A, Toor S (2019) Apache spark streaming, Kafka and HarmonicIO: A performance benchmark and architecture comparison for enterprise and scientific computing. In: Gao W., Zhan J., Fox G., Lu X., Stanzione D. (eds) Benchmarking, Measuri. «and optimizing. Bench 2019. Lecture notes in computer science, vol 12093. Springer, Cham

84. Piras ME, Pireddu L, Moro M, Zanetti G (2019) Container orchestration on HPC clusters. In: Weiland M., Juckeland G., Alam S., Jagode H. (eds) High performance computing. ISC high performance 2019. Lecture notes in computer science, vol 11887. Springer,Cham
85. Zhou N, Georgiou Y, Zhong L, Zhou H, Pospieszny M (2020) Container orchestration on HPC systems," 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), , pp. 34–36, doi: https://doi.org/10.1109/CLOUD49709.2020.00017
86. Ciubăncan M, Dulea M (2017) Implementing advanced data flow and storage management solutions within a multi-VO grid site," 2017 16th RoEduNet Conference: Networking in Education and Research (RoEduNet), TarguMures, pp. 1-4, doi: https://doi.org/10.1109/ROEDUNET.20
87. Pablo Orviz F, Joao P, Álvaro López G et al (2018) umd-verification: automation of software validation for the EGI Federated e-infrastructure. J Grid Comput 16(4):683–696
88. Alfonso DE, Carlos C, Amanda M, Germán. et al (2017) Container-based virtual elastic clusters. J Syst Softw 127:1–11
89. OSG, Open Science GRID. [Online], available Jully 2020: https://opensciencegrid.org/
90. HTcondor resource management. [Online], available Jully 2020: https://htcondor.readthedocs.io/en/latest/overview/index.html
91. The Large Hadron Collider (LHC) - CERN. [Online], available Jully 2020: http://lhc.web.cern.ch
92. Simone M, et al. (2020) CernVM-FS container image integration. J Phys Conf Ser. Vol. 1525. No. 1. IOP Publishing
93. High throughput computing"HTC". [Online], available Jully 2020: https://htcondor.readthedocs.io/en/latest/overview/high-throughput-computing-requirements.html
94. Singularity on HTC. [Online], available Jully 2020: https://indico.cern.ch/event/578972/contributions/2652740/attachments/1491278/2318170/ATLAS_Singularity_Status_1.pdf.
95. Fernández-Del-Castillo E, Scardaci D, López García Á (2015) The EGI Federated Cloud e-Infrastructure. Proc Comput Sci, vol. 68, 2015. doi:https://doi.org/10.1016/j.procs.2015.09.235
96. EGI: Advanced Computing for Research, presentation of Webinar. [Online], available Jully 2020: https://indico.egi.eu/event/5090/attachments/12961/15418/egi-containers-webinar-20200610.pdf
97. EC3 (Elastic Cloud Computing Cluster). [Online], available July 2020 : https://egi-federated-cloud.readthedocs.io/en/latest/aod.html#ec3
98. Moltó G, Caballer M, Pérez A, De Alfonso C, Blanquer I (2017) Coherent application delivery on hybrid distributed computing infrastructures of virtual machines and docker containers. In: 2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), p. 486-490. https://doi.org/10.1109/PDP.2017.29
99. EGI, European GRID Infrastructure Foundation. [Online], available Jully 2020: https://www.egi.eu/
100. AWS, Amazon Web Services.[Online], available August 2020: https://aws.amazon.com/
101. Bisong E (2019) An overview of google cloud platform services. In: building machine learning and deep learning models on google cloud platform. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-4470-8
102. Microsoft Azure. [Online], available August 2020: https://azure.microsoft.com/en-us/
103. Chang H, et al. (2018) Performance evaluation of Open5GCore over KVM and Docker by using Open5GMTC," NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, pp. 1–6, doi: https://doi.org/10.1109/NOMS.2018.8406141
104. Fayos-Jordan R, Felici-Castell S, Segura-Garcia J, Lopez-Ballester J et al (2020) Maximo cobos, performance comparison of container orchestration platforms with low cost devices in the fog, assisting Internet of Things applications. J Netw Comput Appl 169:102788. https://doi.org/10.1016/j.jnca.2020.102788
105. Mouradian C, Naboulsi D, Yangui S, Glitho RH, Morrow MJ, Polakos PA (2018) A comprehensive survey on fog computing: state-of-the-art and research challenges. IEEE Commun Surv Tutorials, vol.20, no.1, pp.416–464, Firstquarter2018. , https://doi.org/10.1109/COMST.2017.2771153
106. Svorobej S, Bendechache M, Griesinger F, Domaschka J. (2020) Orchestration from the Cloud to the Edge. In: Lynn T, Mooney J, Lee B, Endo P (eds) The Cloud-to-thing continuum. Palgrave studies in digital business & enabling technologies., Palgrave Macmillan, Cham
107. Kaur K, Garg S, Kaddoum G, Ahmed SH, Atiquzzaman M (2020) KEIDS: kubernetes-based energy and interference driven scheduler for industrial IoT in edge-cloud ecosystem. IEEE Internet Things J 7(5):4228–4237. https://doi.org/10.1109/JIOT.2019.2939534
108. Huang D, Lu Y (2020) Improving the efficiency of HPC data movement on container-based virtual cluster. CCF Trans HPC 2:67–80. https://doi.org/10.1007/s42514-020-00025-w

109. Riti (2018) Introduction to DevOps. In: Pro DevOps with Google Cloud Platform. A press, Berkeley, CA. https://doi.org/10.1001/978-1-48-42-3897-4_3
110. Potdar AM, Narayan DG, Kengond S, Mulla MM (2020) Performance evaluation of docker container and virtual machine. Procedia Computer Science, vol 171, Pp 1419–1428, ISSN 1877–0509. https://doi.org/10.1016/j.procs.2020.04.152
111. Vazhkudai SS, de Supinski BR, Bland AS, Geist A, Sexton J, Kahle J, Zimmer CJ, Atchley S, Oral S, Maxwell DE, et al. (2018) The design, deployment, and evaluation of the coral pre-exascalesystems.In: , Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis. IEEE Press, p. 52
112. Pereira Ferreira A, Sinnott R (2019) A performance evaluation of containers running on managed kubernetes services. In: 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), , pp. 199–208, doi: https://doi.org/10.1109/CloudCom.2019.00038
113. Bratterud A, Happe A, Duncan RAK (2017) Enhancing cloud security and privacy: the Unikernel solution. In: Eighth International Conference on Cloud Computing, GRIDs, and Virtualization, Athens, Greece. Curran Associates

Authors and Affiliations

Ouafa Bentaleb[1,2,3] · Adam S. Z. Belloum[3] · Abderrazak Sebaa[4,5] · Aouaouche El-Maouhab[1]

Adam S. Z. Belloum
a.s.z.belloum@uva.nl

Abderrazak Sebaa
sebaa@estin.dz

Aouaouche El-Maouhab
elmaouhab@arn.dz

[1] Network Division, CERIST, Algiers, Algeria

[2] Department of Computer Science, Faculty of Exact Sciences, University of Bejaia, Bejaia, Algeria

[3] Institute for Informatics, University of Amsterdam, Amsterdam, Netherlands

[4] LIMED Laboratory, Faculty of Exact Sciences, University of Bejaia, 06000 Bejaia, Algeria

[5] Higher School of Computing and Digital Sciences and Technologies (ESTIN), 06008 Amizour, Bejaia, Algeria