Check for updates

Received: 31 March 2018 | Revised: 10 July 2019 | Accepted: 12 December 2019
DOI: 10.1002/cpe.5668

WILEY

SPECIAL ISSUE PAPER

The state-of-the-art in container technologies: Application, orchestration and security

Emiliano Casalicchio[1,2] | Stefano Iannucci[3]

[1]Department of Computer Science, Blekinge Institute of Technology, Karlskrona, Sweden
[2]Department of Computer Science, Sapienza University of Rome, Rome, Italy
[3]Department of Computer Science and Engineering, Mississippi State University, Starkville, MS

Correspondence
Emiliano Casalicchio, Department of Computer Science, Sapienza University of Rome, Rome, Italy; or Department of Computer Science, Blekinge Institute of Technology, Karlskrona, Sweden.
Email: emiliano.casalicchio@uniroma1.it

Stefano Iannucci, Department of Computer Science and Engineering, Mississippi State University, Starkville, MS.
Email: stefano@cse.msstate.edu

Funding information
Scalable resource-efficient systems for big data analytics, Grant/Award Number: 20140032; PRIvacy-preserving, Security, and MAchine-learning techniques for healthcare applications (PRISMA), Grant/Award Number: RG11816436C00DBA; SmartDefense: Models, Algorithms and Mechanisms for Reducing Cyber Risks in Smart Industry, Grant/Award Number: RG11916B88C838E8

Summary

Containerization is a lightweight virtualization technology enabling the deployment and execution of distributed applications on cloud, edge/fog, and Internet-of-Things platforms. Container technologies are evolving at the speed of light, and there are many open research challenges. In this paper, an extensive literature review is presented that identifies the challenges related to the adoption of container technologies in High Performance Computing, Big Data analytics, and geo-distributed (Edge, Fog, Internet-of-Things) applications. From our study, it emerges that performance, orchestration, and cyber-security are the main issues. For each challenge, the state-of-the-art solutions are then analyzed. Performance is related to the assessment of the performance footprint of containers and comparison with the footprint of virtual machines and bare metal deployments, the monitoring, the performance prediction, the I/O throughput improvement. Orchestration is related to the selection, the deployment, and the dynamic control of the configuration of multi-container packaged applications on distributed platforms. The focus of this work is on run-time adaptation. Cyber-security is about container isolation, confidentiality of containerized data, and network security. From the analysis of 97 papers, it came out that the state-of-the-art is more mature in the area of performance evaluation and run-time adaptation rather than in security solutions. However, the main unsolved challenges are I/O throughput optimization, performance prediction, multilayer monitoring, isolation, and data confidentiality (at rest and in transit).

KEYWORDS

Big Data, cloud computing, cyber-security, docker, orchestration, run-time adaptation

1 | INTRODUCTION

Nowadays, distributed applications and infrastructures are moving from being Virtual Machine (VM) centric to container centric. The technology of containers is strongly supported by PaaS providers,[1,2] IaaS providers,[3] and Internet Service Providers.[4] Moreover, container technologies are used to deploy large scale applications in challenging fields such as big data analytics,[5,6] scientific computing,[7,8] edge computing,[9,10] and Internet-of-Things (IoT).[11,12]

Container's keys to success are many: an easier management of the life-cycle of distributed applcations,[13] a negligible overhead for when they run either on bare-metal or virtual servers,[14-17] and their time to start, restart, and stop, which is reduced up to an order of magnitude with respect to VMs. Last but not least, containers enable application portability, which is where hypervisor-based virtualization failed. Docker, Kubernetes, and Cloudify, the de facto standard container technologies, are compliant with the cloud portability framework TOSCA.[18,19] Moreover, portability is also supported by the Open Container Initiative (opencontainers.org), created to define a standard image format and a standard run-time environment for containers.

Because of these many advantages, the container technology landscape is developing and expanding at the speed of light, but there are still many challenges to be solved, for example, the reduction of networking and I/O overheads compared to hypervisors, the secure resource

 © 2020 John Wiley & Sons, Ltd.

sharing and isolation to enable multitenancy, methodologies and tools for multilayer monitoring, functional to sophisticated run-time adaptation algorithms, advanced run-time adaptation capabilities (eg, scale-out/in, scale-up/down and migration), and high availability support.

In such fast evolving and challenging scenario, a wide spectrum review of the literature which identifies mature and early stage research results is missing.

To fill that gap, we analyzed research work using the following methodology: firstly, we scrutinized the literature to identify the most demanding application fields adopting container technologies (ie, High Performance computing, Big data analytics, and geo-distributed applications) and their related challenges (ie, performance, run-time adaptation, and security). Then, we collected and analyzed additional literature to identify the solutions proposed to solve these challenges and to point out the open issues. As a result of the above methodology, this survey presents a classification and analysis of the literature along the following four categories.

- Applications. This category clusters research papers that use containers in High Performance Computing, Big Data analytics, and geo-distributed (Fog, Edge, IoT) applications.
- Performance comparison. This category groups research works that evaluate the performance of containers under different workloads and on different host environments. All these papers investigated the advantages and drawbacks of using containers as an alternative to VMs or stacked on top of VMs.
- Orchestration. This category includes studies that address the challenge of run-time adaptation of containerized applications* and container platforms to guarantee performance and high availability.
- Security. This category contains research results addressing cyber-security challenges. The focus of the literature is on containers isolation, containers image layers security, storage volumes security, and network security.

To the best of our knowledge and at the time of writing this paper, this is the first extensive literature review on the subject, and it largely extends the previous published survey on container orchestration.[20] A research work quite close to this paper is a survey on the microservice software architectures.[21] However, that study investigates architectural patterns and does not mention challenges related to the implementation and deployment of microservices with containers.

The remainder of this paper is organized as in what follows. Section 2 provides background on application and system containers, container managers, and container orchestrators. The literature's taxonomy, which delineate the above mentioned categories, is described in details in Section 3. Sections 4, 5, 6, and 7 analyze the state-of-the-art works for the categories introduced above. To conclude, Section 8 summarizes how the challenges in the application field are addressed by the literature and put in evidence the open issues.

2 | BACKGROUND

The idea of container dates back to the work of Pike et al.[22] At the base of container technology, there is the concept of *cgroup* and Linux *namespace*.[23] To understand the landscape of containers technology, four main concepts are important: application and system containers, container managers, and container orchestrators.

2.1 | Application and system containers

Application and system containers build on the same technologies and concepts. The only difference is in their use: a system container runs a full operating system, while an application container is intended for the deployment of an application or an application component. For example, the Docker development guidelines and best practices recommend to run a microservice, rather than a full application, in a container. An application container is built up from a series of image layers leveraging the Union File System service[24] and the Copy-on-Write technique, which allows a filesystem to appear as writable but without actually allowing writes to change the filesystem content. Usually, an application developer can start from a base image (eg, the operating system kernel and default libraries). Afterwards, the application can be copied in a new layer (or possibly in two, a bottom layer with the source code and libraries and a top layer with the executable). In the same way, a system container incorporates a base operating system image in the bottom layer, which could be customized with the addition of libraries, tools, and data in new layers. After the multilayer image is built, all the layers are read-only except the last one, which is readable and writable but not persistent. That means the content is lost when the container is deleted.

To persistently store data, a containerized application needs to mount a virtual disk (Docker volumes in the Docker jargon). Docker *volumes* can be mounted in multiple containers simultaneously, allowing concurrent reads and writes. Concurrency can be either managed at the application level or with a distributed filesystem such as GlusterFS[25] and Ceph.[26] Bind-mount is an alternative solution for persistent storage that allows to store data in the host filesystem. Its data is accessible both from the container and the host. Such a solution is not portable and introduces security issues,[27] but on the other end, it could improve the I/O performance.

* A *containerized* application refers to an application packed in a container. It could be a monolithic application or an application composed of independent components. In the latter case, each component could be packed in an independent container

TABLE 1 Container technologies considered in this work. The type acronyms are defined as follows: application container (AC), system container (SC), container manager (CM), orchestration framework (OF)

	Type				
Technology	**SC**	**AC**	**CM**	**OF**	**Url**
Linux Container (LXC)	Y	Y			https://linuxcontainers.org
OpenVZ	Y		Y		https://openvz.org
Windows Hyper-V Container (WHC)		Y			https://docs.microsoft.com/en-us/virtualization/windowscontainers
Docker		Y	Y		https://www.docker.com/
Windows Server Container (WSC)		Y			https://docs.microsoft.com/en-us/virtualization/windowscontainers
`rkt`			Y		https://coreos.com/rkt
LXD			Y		https://linuxcontainers.org
Oracle Solaris Container	Y	Y	Y	Y	http://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html
Amazon EC2 Container Service (ECS)			Y	Y	https://aws.amazon.com/ecs
Google Container Engine (GCE)			Y	Y	https://cloud.google.com/container-engine
Microsoft Azure Container Service (ACS)			Y	Y	https://azure.microsoft.com/en-us/services/container-service
Joyent Triton			Y	Y	https://www.joyent.com/
Kubernetes				Y	https://kubernetes.io
Swarm				Y	https://www.docker.com
Marathon				Y	https://mesosphere.github.io/marathon
Cloudify				Y	http://cloudify.co

Table 1 lists the container technologies considered in this paper. The most widely used application container technology is Docker, a multiplatform solution designed for Linux, OSX, and Windows. Docker extends LXC with a kernel and application-level API to facilitate container management and the management of containerized applications. LXC also can be used as application container. `rkt` is a solution designed to run application containers in a cloud native environment, ie, CoreOS (Container Linux). The Oracle Solaris container solution offers more isolation than Docker. Windows Server Container (WSC) is the Microsoft version of application containers. Concerning system containers, Linux Container and OpenVZ allow to share a linux kernel among containers built from the same base image. LXC uses a standard kernel, while OpenVZ uses a specialized kernel. Windows Hyper-V Container (WHC) is the Microsoft version of system containers; however, in order to guarantee a strong environment isolation, WHCs do not share the kernel among containers.

2.2 | Container manager

A container manager is a framework providing a set of API to easily manage all the life cycle of the container[13] (cf. Figure 1), which consists of the following phases: Acquire, Build, Deliver, Deploy, Run, and Maintain. *Acquire* is related to select and download a container image from a container repository (eg, Docker Hub) and to use it as the base layer on top of which the application will be containerized. In the *Build* phase, all the application components, libraries, and possibly data are packaged in the container image. Afterwards, the image is published in a public/private repository. *Delivery* concerns to bring the application in production and could also include a vulnerability analysis step. *Deploy* is about deploying the application and maintaining it up-to-date, for instance, by using a continuous delivery model. The *run* phase sets the management system and runtime environment (eg, scaling policy, health check, recovery policies). The final step is to *maintain* the application at run-time and off-line. The application behavior is monitored, and when failures are triggered, the system tries to manage them at run-time, eg, by restarting a container. Off-line, the application is debugged to find the root-cause of fault and fixed. Then, the process goes again through the Acquire or Build phases to create a new containerized version of the application that will be delivered, deployed, and maintained.

Container managers can be classified as either on-premise or managed solutions. On-premise solutions need to be installed, configured, and managed on private datacenters or on virtual machines running in the cloud or geo-distributed infrastructures. Managed solutions are instead offered by cloud providers as a service and need only to be partially configured. Among the technologies listed in Table 1. Docker, LXD, OpenVZ, and `rkt` are examples of on-premise solutions. Examples of managed solutions are Google Container Engine, Microsoft Azure Container Service, and Amazon ECS. Docker has been designed as a container management system and is becoming the de facto standard. For example, Windows Server container and Hyper-V container both can be managed with Docker; GCE, ACS, and ECS support Docker containers, and `rkt` offers APIs for easy application container management. Concerning system containers, LXD is the manager for LXC. OpenVZ also provides container management APIs.

2.3 | Container orchestration

Container orchestration allows cloud and application providers to define how to select, deploy, monitor, and dynamically control the configuration of multicontainer packaged applications in the cloud.[28] Container orchestration is concerned with the management at runtime to support the

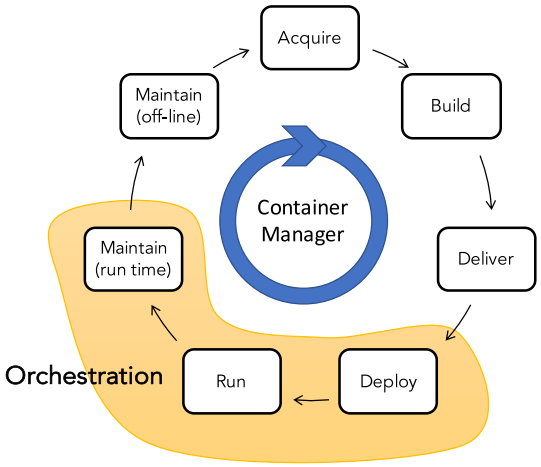

FIGURE 1 The container lifecycle. A container manager provides API to support at least from the acquire to the run phases, while the Orchestrator allows to automatize the deployment, run, and maintain (run-time) phases

TABLE 2 Features implemented by different orchestration frameworks

	Orchestration frameworks			
Orchestration features	**Swarm**	**Kubernetes**	**Cloudify**	**Marathon**
Resource limit control	CPU, Memory		N.A.	N.A.
Scheduling	Resource constrained, Node affinity, Custom			
Load balancing	Round-Robin, Custom		N.A.	Custom
Health check	Open port connections (UDP, TCP, SSL), HTTP req/res			
Fault tolerance	Replica control, High availability controller			N.A.
Auto-scaling	Custom	Custom, Threshold based (CPU)		Custom

Deploy, Run, and Maintain phases (cf. Figure 1). Container orchestrators usually offer the following main features (cf. Table 2): resource limit control, scheduling, load balancing, health check, fault tolerance, and auto-scaling.

Resource limit control allows to reserve a specific amount of CPU and memory for a container; these constraints can be used to make scheduling decisions and to limit the resource contention among containers. Resource limit control features leverage the equivalent mechanisms offered by the container manager. Indeed, while a container can use all the resource available in the underlying system, container managers provide APIs to limit the amount of memory and CPU used and the specific CPU used. However, more sophisticated mechanisms can be implemented. For example, Kubernetes offers the possibility to reserve a specific amount of a resource, eg, the CPU, with `spec.containers[].resources.requests.cpu` and/or to limit the use of a reserved resource, eg the CPU with `spec.containers[].resources.limits.cpu`. If we consider the example of the CPU, `request` allows to allocate a specific amount of CPU (eg, 0.5, 1, 2 cores), while the `limit` allows to specify the maximum amount of time the requested CPU portion is used over 100 milliseconds. The request quota is stringent: if a node is running two containers that specified a CPU request of 0.5 and 0.3, a new container can be allocated only if it has a CPU request less than or equal to 0.2. The limit quota is instead not stringent, that is, a container can exceed, for some time, the quota.

Scheduling defines the policy used to place the desired amount of containers on the desired nodes at a given time instant. Scheduling can be done either on the basis of resource constraints or node affinity or both of them. More sophisticated schedulers can usually be integrated as external components (the custom feature in Table 2).

The load balancer does the work of distributing the load among multiple container instances. Round-Robin is the default implemented policy. More complex policies can be provided by external load balancers (the custom feature in Table 2).

Health check is achieved controlling if a container is capable to answer requests. Usual implementations make use of TCP/UDP/SSH connection attempts to open ports and HTTP request/response checking.

Fault tolerance can be implemented as replica control and/or high availability controller. Replica control allows to specify and maintain a desired number of containers. Health check is used to determine when a faulty container should be destroyed and a new one launched to maintain the target number of replicas. The high availability controller allows to configure multiple orchestration managers to have always control on the application in case an orchestrator node fails or is overloaded. The same technique used to create a high availability controller can be used to implement a scalable controller.

Auto-scaling allows to automatically add and remove containers. The implemented policies are usually based on thresholds (eg, on CPU and memory utilization), but in some cases, it is possible to plug-in more sophisticated autoscalers or to define custom autoscaling policies (the custom feature in Table 2).

In the landscape of container orchestration frameworks, the choice is between on-premise solutions and managed solutions (as for the container managers). The main on-premise solutions are Docker Swarm, the native Docker orchestrator offering clustering functionality for Docker containers, which lets system administrator turn a group of Docker engines into a single virtual Docker engine. Kubernetes is an orchestration system for Docker containers capable to handle scheduling and to manage workloads based on user-defined parameters. Mesosphere Marathon

is a container orchestration framework for Apache Mesos. It offers key features for running applications in a clustered environment. Cloudify is a cloud orchestration framework that enables the modeling of applications and services and automates their entire life cycle. Cloudify is TOSCA compliant and could be used to deploy Docker containers, Docker Swarm clusters, or Kubernetes clusters. Container as a service solutions offer also orchestration services. Some examples are Google Container Engine, Amazon Elastic Container Service, and Microsoft Azure Container Service.

3 | STATE-OF-THE-ART TAXONOMY

The focus of this survey is schematized in Figure 2. The literature is classified using a taxonomy composed of the four categories described in Section 1 and several subcategories. *Application's* subcategories are High Performance Computing, Big Data analytics, and geo-distributed (Fog, Edge, and Internet-of-Things) applications. Although any application can be deployed using containers, those mentioned above are the most challenging domains. The *Performance comparison* category does not need to be further detailed by sub categories because all the studies compare the performance of container solutions w.r.t. system virtualization and native environments. *Orchestration* subcategories are: Performance Monitoring, Characterization and Prediction, Auto-scaling, High Availability, and Scheduling. Energy efficiency is another important aspect, and it is orthogonal to the Performance Evaluation and Orchestration categories. *Security* subcategories collect research results about container's isolation, image layers and volumes encryption, and network security.

We decided to organize the literature as above because the analysis of application fields is valuable to understand challenges and open issues. Moreover, the study of the other categories, orthogonal to the first one, present the solution proposed to address the challenges and put in evidence what areas and results are mature, and what need more effort.

This survey presents the result of an extensive although not omni-comprehensive literature review that analyzes and summarizes the contribution of 97 verified sources selected among about 110 retrieved from the main research databases like ACM digital library, IEEE Explorer, Scopus, and arXiv. The following are the main keywords used in the search: docker, container, application virtualization, performance, monitoring, orchestration, self-adaptation, scheduling, energy efficiency, auto-scaling, security, encryption, availability, and self-healing. The paper selection process took into consideration also the quantity of papers in each category. A reasonable balance between the applications category (about 1/4 of the papers) and the other categories (about 3/4 of the papers) has been maintained, as Figure 3 shows.

The distribution of the papers among the four main categories highlights that the majorities of research work is focused on orchestration (38%), while the 23% address security issues and 13% performance comparison. While it is understandable that there are few works on performance comparison because the outcomes obtained until now converge to similar results (small footprint w.r.t. VMs, and limitation in network I/O performance), the number of papers addressing security issues is inadequate, considering also the importance of cyber security today. A more

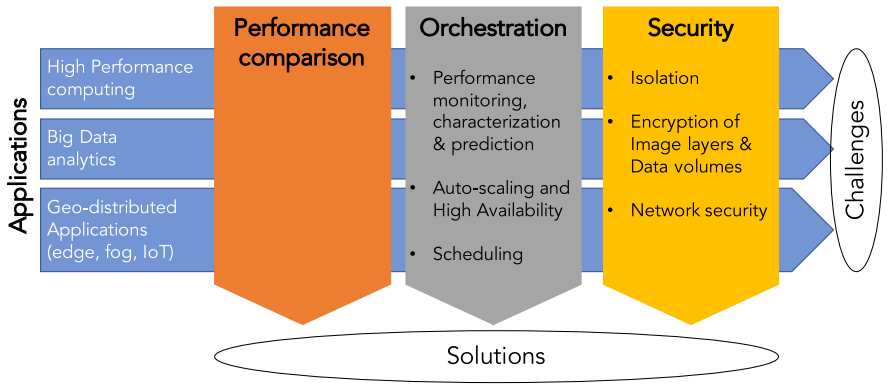

FIGURE 2 The proposed literature taxonomy

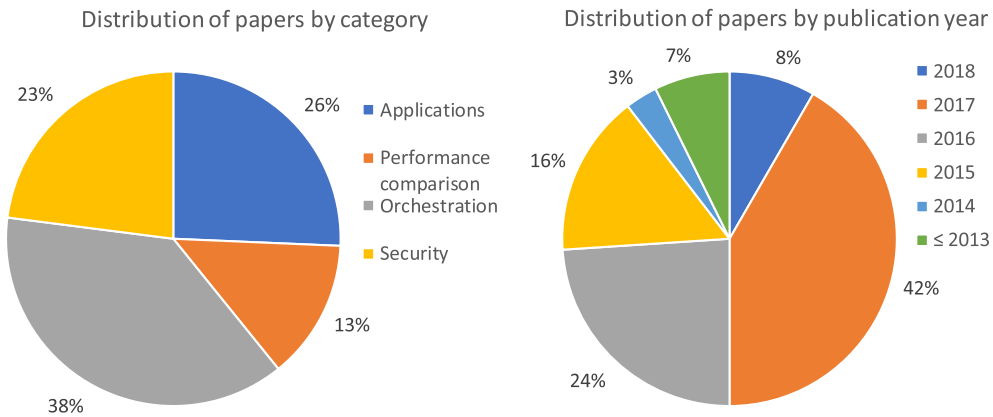

FIGURE 3 Distribution of the analyzed papers: by publication year (right) and by taxonomy categories (left)

detailed analysis (c.f. Table 6) highlights that researchers' effort is mainly devoted to enhance isolation properties of containers, while there are few research works on image/volume encryption and network security.

Finally, it is also interesting to observe the novelty of research results: the 90% of the studies have been published between 2015 and 2018, as illustrated by the diagram in Figure 3, and the 50% of the papers analyzed have been published in the last two years, from 2017 to 2018.

4 | APPLICATIONS

4.1 | High performance computing

Today, the users of super computing platforms are demanding for solutions to easily deploy their own defined software stack or virtual HPC clusters and for leveraging the flexibility offered by containers in their real-life computational and data intensive jobs. To satisfy such demand, many challenges should be solved, for instance:

Support for MPI and CUDA libraries; support for execution of containerized parallel applications on GPU and FPGA; scheduling of containerized HPC jobs; mechanisms to guarantee, in the new container enabled HPC platforms, the same performance of native environments; solutions to guarantee the proper level of security (isolation of users' environment, data security, network security). Table 3 (first column) summarizes the challenges and the related work raising and/or proposing solutions to the challenges.

In the work of Priedhorsky and Randles,[8] the authors address the problem of deploying User Defined Software Stacks (UDSS) in supercomputing centers. The challenge is to provide a usable environment while minimizing the risks: security, support burden, missing functionality, and performance. The paper proposes Charliecloud, a framework based on Linux user and mount namespaces to run UDSS in Docker containers with no privileged operations or daemons. The adopted solution does not use network, host, and domain name namespaces to gain in networking performance (critical for HPC). User namespace is used to limit the access to system calls, while security is managed using the Linux kernel capabilities (eg, to enforce access control). Charliecloud has been validated against security attacks and performance. Socker[29] aims to leverage the flexibility offered by containers in HPC real-life computational and data intensive jobs. Socker is a solution to securely run Docker within cluster jobs and HPC job scripts (it is essentially a secure wrapper) and to limit the resource usage of a Docker job to the borders defined by the HPC queuing system. Socker is based on the Slurm queuing systems and does not require to modify the underlying Docker engine or to replace it. Fe2vCl2[35] is a framework for deploying HPC applications on common cloud infrastructures. Specifically, Fe2vCl2 allows the deployment of user defined virtual clusters for running HPC applications. The performance (FLOPS and network throughput) of the proposed solution has been assessed against a deployment based on VMs.

In the work of Wrede and von Hof,[34] the authors propose a framework to efficiently deploy parallel C++ applications in a cloud environment by using Docker containers and Docker Swarm (for automatic node scaling). The proposed solution leverages OpenMPI (open-mpi.org) and CUDA (developer.nvidia.com/cuda-zone) container images to enable code execution on both CPU and GPU.

In the work of Bayser and Cerqueira,[33] the authors extend MPICH (www.mpich.org/) to support containers. The proposed solution allows to deploy MPI-based parallel applications with Dockers.

In the work of Julian et al,[40] the authors propose a solution to deploy scientific applications and workflows on HPC cluster using Docker containers and the Adaptive Computing's Moab scheduler. The performance of the proposed solution has been evaluated in term of FLOPS, network throughput and filesystem throughput, and compared with the performance of a RedHat cluster. To increase container's network performance, a bridge network solution and the Linux's macvlan feature are used instead of Docker's NAT native mechanisms. The filesystem performance in case of NFS has been enhanced using a Docker's NFS plugin.

4.2 | Big Data analytics applications

Containers offer an easy way to deploy Big data analytics applications based on the map-reduce or stream processing paradigms. Moreover, as discussed in the previous section, solutions are emerging to support the execution of containers on GPU and FPGA and to speedup specific analytics tasks. As summarized in Table 3 (second column), the deployment of Big Data analytics applications with containers poses new challenges: scalability and performance should be guaranteed for stream processing and batch processing applications, as well as for training

TABLE 3 Challenges in the HPC, Big Data analytics, and geo-distributed (Fog, Edge, IoT) application fields

HPC	Big Data analytics	Fog, Edge, IoT applications
Security[8,29]	Security[30,31]	Security[32]
Deployment of parallel applications on CPU[29,33-35] and on GPU[34,35]	Scalable stream processing[36,37] and batch processing[5,38] platforms	Container migration[10,39]
HPC job Scheduling[29,40]	Scalable testing[5,37] and training[31] platforms	Resource and energy efficient service provisioning[39,41,42]
Optimization of performance[8,29,35,40]	Large data set management[30,43,44]	Deployment on etherogeneus nodes[32,42,45]

and testing environments. For the latter, there is also the automation challenge. Finally, the large volume of data needs to be securely managed on-the-fly (ephemeral filesystems) and at rest (Docker volumes).

Data stream processing is a computing model that finds application in many fields such as big data analytics, security, IoT, performance monitoring, and distributed event processing. In the work of Wu et al,[36] the authors implement a Stream Processing Platform Service, deployed using Docker and Kubernetes, to test the data flow prediction algorithms they proposed. In the work of Kyong and Jeon,[37] the authors propose a Docker-based architecture and a partitioning method to reduce the scalability problem of Apache Spark-based scale-up servers caused by garbage collection and remote memory access overheads when the servers are equipped with significant number of cores and Non-Uniform Memory Access (NUMA). The paper shows that, using Docker container-based architecture, the problem could be minimized effectively by partitioning the original scale-up server into small logical servers.

In the work of Chen et al,[38] the authors describe the development of Virtual Hadoop, a Docker-based framework that overcomes the scalability issues of HadoopCL, a variant of Hadoop that takes advantage of heterogeneous computing by offloading the computation kernels of a Hadoop application to the accelerators composed by multicore CPUs, GPUs, and/or even FPGA boards. HadoopCL does not support on-demand resource scaling. Docker is used to wrap-up the execution environment of Hadoop rather than the user application. Virtual Hadoop includes an extension of the methods of resource inference and allocation, and an auto-scaling mechanism to dynamically allocate resources on-demand when new workloads are submitted to the Hadoop cluster. The autoscaling is based on a performance model that predicts the needed resources for the new workload and estimates if the available resources (Hadoop virtual nodes) are enough to run the new workload or new nodes are needed. In the work of Nguyen et al,[5] Docker is used to deploy an Hadoop virtual cluster on a single server node to run a distributed version of the Vp-Tree algorithm that leverages the MapReduce framework. The performance on the Docker cluster is compared with the performance of a traditional virtual machine based cluster (on a single server node).

In the work of Schipp et al,[31] the authors present the Isolated, Scalable, and Lightweight Environment for Training (ISELT), targeting Massive Open Online Courses requiring interactive GNU/Linux commandline environment. ISELT has been deployed using docker containers and claims to overcome many of the distribution, scaling, and security challenges for the specific training subject. While the authors provide a performance evaluation of ISELT, they do not provide any detail about how security and isolation are managed.

SciServer[30] is a big-data infrastructure project that is developing a modular and scalable infrastructure for the storage, access, query, and processing of large, petabyte scale, scientific datasets. The SciServer Compute component leverages Docker containers to provide security and isolation and to allow flexible configuration of computational contexts through domain-specific images and mounting of domain specific data sets. Docker volumes are used to encapsulate dataset, which makes it easy to attach them to application containers as external storage, bringing data close to compute resources running analytic applications.

The storage of large datasets is also addressed in the work of Ma et al,[44] which tackles the two main limitations of data deduplication based on Map-Reduce, namely the inability to find duplicate data in adjacent blocks and the possibility to miss duplicate date due to a fixed sliding window. Their approach has been validated using Docker containers to run three Hadoop instances on a physical node to execute the Map-Reduce jobs. Furthermore, given the highly distributed nature of container-based applications and the need to process large datasets, the role of efficient caching becomes crucial to reduce the network utilization, which might become a bottleneck and thus to reduce the processing and waiting time. Semantic cache allows the cache to be used even though the requested data is only partially present in the cache. Several solutions[46,47] have been proposed that implement semantic-caching; however, some challenges remain open such as the reduction of the query processing time and the cache hit ratio. To this end, in the work of Ma et al,[43] the authors propose a novel semantic caching algorithm that outperforms regular semantic cache and decisional semantic cache.[46]

4.3 | Edge, fog and IoT applications

Since containers are portable and have a small performance footprint, they are suitable to deploy applications on heterogeneous, resource-limited, and power-limited computational units. Deploying containers on edge or fog computing nodes or deploying containers on IoT boards like Rasperry PI is extremely advantageous, although it rises challenges in migration, service, and resource provisioning and security, as outlined in Table 3 (third column).

In the work of Ma et al,[10] the authors address the problem of mobile clients that need to offload services to an edge computing platform. The authors present a novel service handoff system based on container migration. In the paper, an important performance issue in Docker container migration has been identified and solved. The proposed solution leverages the layered storage system to reduce filesystem synchronization overhead without dependence on the distributed filesystem.

The feasibility of using docker in Mobile Edge Computing scenarios and specifically for video streaming and multiplayer gaming applications is investigated in the work of Avino et al.[48] The authors discover that, in case of video streaming, the overhead is independent from the number of clients and servers while, in case of multiplayer gaming, the overhead increases with the number of servers (running containers). A study about how to improve service provisioning flexibility in edge community cloud is proposed in the work of Baig et al.[41] The authors present the evolution of Guifi.net, integrating Docker and Cloudify to deploy software defined networks. Another example of how container could facilitate edge computing is provided in the work of Holdgraf et al.[39] In that paper, the authors describe how Docker is used: to deploy a learning environment on the Jetstream cloud infrastructure and to enable portability of the students' learning environment from the cloud to their edge devices (eg, laptop or tablet).

A container-based fog-oriented framework for Internet-of-Things applications based on Kura gateway and Docker is studied in the work of Bellavista and Zanni.[45] The paper demonstrates that Kura gateway can be containerized for an easy deployment on top of existing, off-the-shelf, and limited-cost gateway nodes. Specifically, it has been possible to implement a highly manageable and interoperable way to create fog nodes on-the-fly via the adoption of application containerization and container orchestration.

In the work of Rufino et al,[32] the authors address the heterogeneity problem in industrial IoT by proposing a Docker-based architecture that allows the deployment of applications despite the specific device/protocol. Moreover, high availability and scalability, as well as fault tolerance, are guaranteed by the Docker engine and Docker Swarm.

Another interesting use of container is Cloudrone[42]: a microcloud deployed on Raspberry PIs mobile nodes transported by drones. The solution uses Docker to deploy services with a very small resource demand footprint and Swarm to manage clusters of Docker nodes.

5 | PERFORMANCE COMPARISON

One of the first goals of the research community has been to assess, under different workloads: how the performance of a deployment based on containers compare with the performance of virtual machines and/or bare-metal deployments and how containers compares with VMs in term of energy consumption. The outcome of the analysis is the following: the performance footprint of containers is negligible, read and write performance heavily depends on the filesystem type selected and on its configuration, and network I/O is poor compared to VMs and native environments.

Table 4 summarizes the comparison scenarios considered in literature.

The first seminal work on container performance evaluation[14] provides an extensive comparison among a native Linux environment, Docker, and KVM. In that work, the three environments are compared in presence of CPU-intensive, I/O-intensive, network-intensive, and NoSQL/SQL workloads. The main intention of the work is to assess the performance improvement of running workloads in containers rather than in VMs. The comparison is based on the performance metrics collected by the benchmarking tools rather than on the workload footprint. In the work of Morabito et al,[15] the authors present a similar study, aimed at comparing the performance of containers with hypervisors. The performance of Cassandra while running on bare-metal native cluster of nodes, VMware VMs, and Docker containers is investigated in the work of Shirinbab et al.[49] The authors focus their investigation on the throughput and latency of Cassandra when different replication factors and consistency schemas are used, with the final goal of providing recommendations for Cassandra virtual cluster configuration and deployment.

Different studies investigate the performance of containers running on cloud infrastructures, like AWS EC2. A performance comparison of Docker versus Flockport (LXC) is presented in the work of Kozhirbayev and Sinnot[16] using the same benchmarks as in the work of Felter et al.[14] In this work, containers are deployed on top of the NeCTAR cloud. The comparison is intended to explore the performance of CPU, memory, network and disk. Docker versus Joyent's Triton is studied in the work of Balasubramanian Sekar et al.[50] The authors show that on top of AWS EC2 virtual machines, Joyent's Triton performs better or almost the same than Docker and that it is a valid alternative considering the added security features (cf. Section 7).

While standard benchmarks are used in the previous studies, the scientific workload is considered in the work of Adufu et al.[51] The authors show that Docker memory configuration can be tuned to make container performance be slightly better than VMs.

The performance of containers running on Internet-of-Things devices are investigated in the work of Morabito.[55] Docker runs on a Single Board Computer device such as Raspberry Pi 21. Workload used are system benchmarks to independently stress CPU, memory, network I/O, disk I/O, and application benchmarks reproducing MySQL and Apache workloads. The reference for comparison are the performance of the system without any virtualization.

How filesystem selection and configuration could impact the performance of Docker is investigated in the works of Tarasov et al[52] and Dua et al.[53] In the work of Tarasov et al,[52] the authors perform an extensive study to unravel the multifaceted nature of Docker storage, for a wide range of filesystems, and to demonstrate its impact on system and workload performance. An assessment of the performance of Docker when the Union filesystem and CoW are used to build image layers is provided in the work of Dua et al.[53] The impact of high speed SSDs storage on Docker performance is analyzed in the work of Xu et al.[54] The authors evaluate the performance of storing Docker images and data w.r.t. different filesystems type and storage drivers.

In the work of Morabito,[56] the authors compare the energy consumed by Docker and LXC with the energy consumed by KVM and Xen. In the comparison, the cases for memory-, CPU-, and network-intensive workload are considered, as well as for the idle state. Results show that

TABLE 4 A summary of the performance comparison studies

Performance comparison
Docker vs KVM and Linux native[14,15]
Docker vs VMware and Linux native[49]
Docker vs LXC[16] and Joyent's Triton[50]
Docker configuration tuning: memory,[51] filesystems[52-54]
Docker vs native deployment on Raspberry Pi[55]
Energy consumption Docker/LXC vs KVM/Xen[56]

TABLE 5 A summary of the solutions proposed by the research community to address the container orchestration challenges: performance monitoring, characterization and prediction, auto-scaling and high-availability, and scheduling

Performance monitoring, characterization, and prediction	Auto-scaling and high-availability	Scheduling
Measurement methods and techniques[17,57,58]	Horizontal scaling[6,57,59-62]	Scheduling of containers on VMs,[63-65] on PMs,[66] and in geo-distributed environments[67-69]
Multilayer monitoring solutions[57,5861]	Vertical scaling[59,70]	CPU and memory-based opimization,[64,66] I/O-based optimization[63]
Characterization of resource contention[71,72]	Geo-distributed scaling[59,73]	Multiobjective optimization[67]
Enhancement of Docker monitoring:[57,72]	Application specific scaling: Time critical,[60] Hadoop,[38] Data stream processing[36]	Energy-based optimization[11]
Performance prediction[58,61,71,74] Characterization of energy consumption[77]	High-availability solutions[75,76]	

container technologies consume less energy than hypervisors-based virtualization only for network intensive workload. In all the other cases, virtualization technologies behave the same.

6 | ORCHESTRATION

In the container life cycle, orchestration supports automation and run-time adaption (c.f. Figure 1). In this survey, we focus on the run-time adaptation challenges, which are to design new methods and tools for performance monitoring, characterization, and prediction, to design resource and energy efficient auto-scaling and scheduling algorithms, and to design new solutions for high-availability. The solutions proposed by the research community to address the above challenges are summarized in Table 5. An analysis of the literature is presented in what follows.

6.1 | Performance monitoring, characterization, and prediction

Performance monitoring is a topic of increasing interest for the containers' research community. In the work of Casalicchio and Perciballi,[17] the authors assess the different measurement techniques used to collect performance counters for CPU and disk I/O-intensive Docker workload. The study demonstrates that container performance counters introduce an under estimation ranging between the 5 and 10 percent w.r.t. system performance counters. Such a difference could hurt the behavior of scheduling and autoscaling algorithms, as proved in the work of Casalicchio and Perciballi,[61] where the impact of relative and absolute metrics on the Kubernetes autoscaling algorithm is investigated.

The importance of planning deployment and adaptation decisions on the basis of performance counter values collected at all the virtualization layers is addressed in the works of Khazaei et al[57] and Taherizadeh and Stankovski.[58] In the work of work of Khazaei et al,[57] the authors propose Elascale, a cloud service that automatically instruments the containerized application to collect performance counters at container and VMs layers, and then, it uses the performance measure for autoscaling. The same approach is used in the work of Taherizadeh and Stankovski,[58] where multilayer monitored data are used to drive the run-time adaptation. In the work of McDaniel et al,[72] the authors propose an enhancement of the Docker's kernel to collect the maximum I/O bandwidth for the machine it is running on. That made available one more performance measure in Docker.

In the work of Ye and Ji,[71] the authors characterize the impact of Docker configuration and resource contention on the performance and scalability of big data workloads (four typical Apache Spark's applications). The resource contention among multiple I/O-intensive applications sharing the same resources and running in Docker containers is characterized in the work of McDaniel et al.[72] A preliminary characterization of the energy consumed by Docker containers is provided in the work of Tadesse et al.[77] The authors focus on correlating the energy consumed with the CPU and network pressure. The results show that CPU is the main source of energy consumption. Intensive network load that stresses the OS results in a not negligible consumption of energy as well. Such results are important for designing energy efficient resource scheduling.

Performance prediction is key for capacity planning, optimal deployment, and run-time adaptation. A performance prediction model based on the Support Vector Regression is proposed in the work of Ye and Ji.[71] The paper considers data stream processing Apache Spark applications, running in Docker containers, with different configurations and resource contention settings. In the work of Taherizadeh and Stankovski,[58] the authors propose a Learning Classifier System that, on the basis of multilayer monitored data, incrementally learns rules representing the behavior of containerized applications. The extracted behavior is used to predict the QoS level offered by the application. A Layered Queuing Network-based performance prediction model for multitier containerized applications is proposed in the work of Barna et al.[74] The model allows to predict resource demand in development phase. In the work of Casalicchio and Perciballi,[61] the author study the correlation between the container performance counters (counters from `/cgroup` filesystem) and the system performance counters (from `/proc` filesystem). Such correlation model is useful for workload prediction and for the design of scheduling and autoscaling strategies.

6.2 | Auto-scaling and high-availability

The adoption of container technologies call for new run-time adaptation solutions.[28,78]

Recently, many research works have proposed new solutions for horizontal and vertical scaling of Docker containers. The approaches used are generally inspired by the research results obtained in the last two decades in autonomic computing.[79-82]

Among the auto-scaling solutions, the most promising mix vertical and horizontal scaling and/or migration. Usually, the autoscaling algorithms were designed for locally distributed adaptation, but solutions that consider the geo-distributed case suitable for IoT applications have been proposed as well. In high-availability, the most promising approach is to make the detection of faulty nodes more sophisticated, eg, based on abnormal behavior rather than only on network level or application level probes.

An early study on container management[83] shows that Elastic Application Container-based resource management outperforms the VM-based approach.

In the work of Nardelli et al,[6] the authors provide an Integer Linear Programming (ILP) formulation of the elastic provisioning of virtual machines for container deployment. The general ILP problem formulation takes explicitly into consideration the heterogeneity of container requirements and virtual machine resources. Only QoS and cost are considered in the problem formulation. The same approach is used in the work of Nardelli et al[59] where the authors propose Adaptive Container Deployment (ACD), a general model of the deployment and adaptation of containerized applications. Besides acquiring and releasing geo-distributed computing resources, ACD can optimize multiple run-time deployment goals, by exploiting horizontal and vertical elasticity of containers. The ACD model is used as a benchmark to evaluate the behavior of several greedy heuristics for determining the container deployment.

An adaptive multi-instance container-based architecture targeting time-critical applications is proposed in the work of Stankovski et al.[60] The solution has been implemented with Docker and Kubernetes.

The behavior of the Kubernetes Horizontal Pods auto-scaling algorithm is studied in the work of Casalicchio.[84] The author proposes a new solution to make a more appropriate allocation of resources to fulfill application response time constraints. ElasticDocker, an autonomic controller powering vertical elasticity of Docker containers, is presented in the work of Al-Dhuraibi et al.[70] ElasticDocker scales up and down both CPU and memory assigned to each container according to the application workload and live-migrates containers when there is not enough resources on the hosting machine. The experiments show that ElasticDocker makes better resource utilization for container providers and improves Quality of Experience for application end-users. An architecture of a SaaS autonomic application manager based on Docker and Kubernetes is proposed in the work of Truyen et al.[73] The high-level architecture is based on three autonomic managers that should be capable to adapt the multicloud infrastructure and the multitechnology data storage level with the goal of guaranteeing tenants' SLAs. While no details on adaptation algorithms are provided, the proposed architecture delineates a new research problem in the field. Elascale[57] scales microservices based applications deployed with Docker and Docker Swarm. The proposed solution applies a default threshold-based, reactive scaling algorithm for all the application's micro and macro services. In the work of Kim et al,[62] the authors propose a proactive autoscaling mechanism that scales in/out containers and distributes the load among instances on the base of the network traffic intensity.

Autoscaling algorithms targeting specific computing paradigms are discussed in the works of Chen et al[38] (Hadoop) and Wu et al[36] (Data stream processing). In the work of Chen et al,[38] the authors propose an auto-scaling algorithm for docker containers wrapping up the execution environments of Hadoop nodes. The autoscaling algorithm is integrated in a framework called Virtual Hadoop for deploying Hadoop clusters on heterogeneous nodes. The algorithm estimates the needed amount of containers to meet the time constraint of the Hadoop job and automatically allocates containers. In the work of Wu et al,[36] the authors propose SSPS, an autoscaling mechanisms for Data Stream Processing applications deployed using Docker. The proposed autoscaling algorithm takes decisions based on the predicted data arrival rate. SSPS is designed to replace the default CPU threshold-based scaling policy in the Horizontal Pod Autoscaler of Kubernetes.

Container orchestrators such as Docker Swarm and Kubernetes offer high-availability mechanisms based on monitoring the health state of a service and restarting the container in case the health check fails. Such basic mechanisms are enhanced in Serfnode,[75] a lightweight platform agnostic and easy to integrate with an existing system of Docker containers. The proposed solution includes a monitoring and self-healing mechanism based on Supervisor (supervisord.org) for increased availability. In the work of Naik,[76] the author proposes an intuitive approach based on Computational Intelligence (CI) for enhancing the availability of Docker Swarm. The proposed CI-based approach predicts the possible failure of the host of a manager node by observing its abnormal behavior. Thus, this indication can automatically trigger the process of creating a new manager node or promoting an existing node as a manager for enhancing the orchestrator's availability.

6.3 | Scheduling

Scheduling is another brick of the run-time adaptation. Both static and dynamic scheduling solutions have been proposed.

In the work of Zhao et al,[63] the authors investigate the problem of scheduling network I/O-intensive containerized applications on top of a cloud computing infrastructure (ie, VMs). The proposed solution maximizes the number of allocated applications (ie, containers) on a finite set of VMs with no outbreak of I/O bandwidth of physical nodes. At the same time, the optimal scheduling satisfies the application constraints on CPU, memory, and disk demand. The proposed scheduler has been implemented in Diego, a Docker container orchestrator for Cloud Foundry.

In the work of Kaewkasi and Chuenmuneewong,[64] the authors propose an Ant Colony Optimization (ACO) algorithm to schedule resource for Docker containers. The ACO algorithm has been implemented in Docker SwarmKit and the performance has been compared with the default

greedy scheduling algorithm. A framework for Application Oriented Docker container (AODC) resource allocation is presented in the work of Guan et al.[66] AODC minimizes the application deployment cost in datacenters and considers deployment of containers on PM. The deployment cost is related to available supporting libraries on PMs and required libraries of applications. AODC is compared against optimal VM placement algorithms.

C-Port[67] is the first example of orchestrator that makes it possible to deploy and manage containers across multiple clouds. The authors address the issues of container scheduling and placement and dynamic adaptation. In terms of orchestration policy, C-port uses a constraint-programming model for dynamic resource discovery, selection, and scheduling. Decisions can be taken on the basis of availability, cost, performance, security, or power consumption constraints. The C-Port orchestrator is also used in the work of Abdelbaky et al[65] to realize a distributed software defined environment that deploys applications with docker over the CometCloud federated cloud infrastructure.

Scheduling containers on geo-distributed platform (eg, federated clouds, edge, fog, and IoT infrastructures) is challenging for a latency efficient download and deployment. In CoMICon,[68] the authors propose a solution for reducing deployment time by cooperative management of Docker images. That enables their latency efficient scheduling algorithm to start container in a distributed or geo-distributed platform. In the work of Kangjin et al,[69] the authors propose FID, a system for fast docker image distribution. FID uses a P2P BitTorrent-based approach integrating a BitTorrent client in Docker registry and developing a FID agent that could work as a proxy for the Docker engine or as a REST API for developers (in that case the FID component uses Docker load).

In the work of Asnaghi et al,[11] the problem of properly scheduling resources to containers in a IoT scenario to reduce energy consumption is addressed. The authors propose DockerCap, a software-level power capping orchestrator for Docker containers that follows an Observe-Decide-Act loop structure, which allows to quickly react to changes that impact on the power consumption by managing resources of each container at run-time, to ensure the desired power cap. The paper shows that it is possible to obtain results comparable with the state-of-the-art power capping solution provided by Intel RAPL while still being able to tune the performance of the containers and even guarantee SLA constraints.

7 | SECURITY

Docker security issues and challenges are discussed in the work of Combe et al.[85] The authors give an overview of the vulnerabilities in terms of container isolation, host hardening, and network security. Moreover, there are relevant security issues at the image distribution and container control levels. The Docker Content Trust Infrastructure relies on TUF[86] to enable image signature; Docker Hub integrates Diplomat[87] to implement disambiguated trust delegations. However, other repositories could apply lower security mechanisms. Mechanisms and protocols to make trustable an image repository do not prevent the image to contain vulnerabilities that could be exploited. The authors of the Docker Image Vulnerability Analysis (DIVA) framework[88] discovered that both official and community images contain more than 180 vulnerabilities on average when considering all versions. That challenge can be solved by means of automatic security updates. Frameworks for updates recommendation such as RUDSEA[89] can be an approach to mitigates the problem.

A software engineering approach to enhance container security is considered in the work of Syed and Fernandez.[90] The authors propose a reference architecture for the container ecosystem, claiming that it could facilitate who aims to ensure compliance, privacy, safety, reliability, and/or governance. For example, security patterns can be used to build secure systems by describing ways to control specific threats, fix a vulnerability, or provide a security attribute.

Table 6 summarizes the solutions proposed by the research community to improve container isolation, image encryption, and network security. A detailed analysis of the literature is presented in what follows.

7.1 | Isolation

The majority of work in container security addresses the isolation challenges. Isolation in accessing the resources is fairly guaranteed by the Linux namespace and *cgroup*, but there is a flaw in how containers share the same network bridge. Moreover, container isolation can be lowered at launch time playing with specific settings. In terms of host hardening, Linux operating system security policies offer a good protection for the

TABLE 6 A summary of the solution proposed by the research community to address the security challenges: isolation, volumes and images encryption, network security

Isolation	Volumes and images encryption	Network security
SGX-based isolation[91-96]	SGX-based volumes encryption[91-93,96]	SGX-based communication channel encryption[92-94,97]
Kernel-based isolation in Linux[8,29] and in Solaris[50] (Joynet Triton)	SGX-based ephemeral filesystem encryption[92,93]	Host identity protocol based[98]
Linux hardening-techniques[99,100]	Selective image layers encryption at-rest[91,101] and on-the-fly[101]	Channel leakage mitigation techniques[102]
Application level isolation[103]		
Detection of escape attacks[104]		

host, but they do no protect a container from other containers. The most promising solutions are based on Intel SGX enclaves, but the drawback is the dependence on the Intel architecture and the implementation complexity.

In the work of Jian and Chen,[104] the authors focus on Docker escape attacks and exhaustively discuss the existing security mechanisms and security issues of Docker. The paper describes different methods to conduct Docker escape attacks and proposes a defense method based on status inspection of namespaces to detect anomalous processes. Covert channels could cause critical results like information leak between one container and another (or even the host). Covert channels attacks against Docker are extensively investigated in the work of Luo et al,[99] where the authors analyze how to identify different types of covert channels and how current isolation mechanisms can be used and configured to prevent the attacks. Furthermore, the authors conclude that adopting full-fledged SELinux or AppArmor security policy is a key condition to protect the security perimeters of containers. Another study about the use of Linux hardening techniques to improve container security is presented in the work of MP et al.[100] Joinet Triton are containers running on Oracle/Solaris and offer the built-in zone, an isolated and secure environment for running containerized applications.[105] Hence, Joynet Triton is a secure alternative to Docker when containers run on bare-metal. In the work of Balasubramanian Sekar et al,[50] the authors compare the performance of Docker running on AWS EC2 with Joyent's Triton container showing that the latter performs better or almost the same and that it is a valid alternative adding security to performance.

As mentioned before, Charliecloud[8] is a secure framework based on Linux user and mount namespaces to run HPC applications on Docker containers. Charliecloud provides isolation using Linux kernel to enforce access control and other aspects of security. Moreover, Charliecloud prevents shenanigans such as creating device files or setuid binaries by means of doing operations on center-owned resources as the invoking unprivileged user. Charliecloud has been proved secure against hacks tentatives such as `chroot` escape, bypass of file and directory permissions, bind to privileged ports on all host IP addresses, `seteuid` to an unmapped UID, and the like. As mentioned before, Socker[29] is a secure wrapper for running Docker containers on Slurm and similar HPC queuing systems. The execution of containers within Slurm jobs is done with the submitting user privileges instead of root privileges. Moreover, Socker enforces the inclusion of containers in the cgroups assigned by the queuing system to the parent jobs.

In the work of Richter et al,[91] the authors present a generic concept to provide SGX security guarantees for operating system components, isolating kernel functionalities by means of Intel SGX enclaves. The solution can be useful to increase container security. SCONE[92] is a secure container mechanism for Docker that uses the Intel SGX to protect container processes from outside attacks. SCONE exposes a C standard library interface for transparently encrypting/decrypting application data on a per-file-descriptor basis. The proposed solution has a low overhead and is transparent to Docker because secure containers behave like regular containers. SCONE is also used by other solutions such as SERECA[93] and SecureCloud[94] to increase security of the stored data and of the network communications. For example, SecureCloud aims to provide a secure remote computation platform running microservices in secure SCONE containers. Moreover, in the work of Krieter et al,[95] the authors propose to use Intel SGX enclaves to isolate applications by means of encrypted memory. They assume to have applications which code need to be partitioned in single functionalities (step not needed if the application is designed using the microservice architecture). SecureStream[96] is a reactive middleware framework to deploy and process secure streams (eg, Map-Reduce applications) at scale. SecureStream combines the high-level reactive dataflow programming paradigm with Intel SGX enclaves in order to guarantee privacy and integrity of the processed data. All the containers run in enclaves and store encrypted data within enclaves, thus realizing high isolation.

DATS, a system to increase isolation at application level, is proposed in the work of Hunger et al.[103] The authors consider web applications that heavily access data in shared folders and that are deployed with containers. DATS introduces a templating language to compose data across data containers. Moreover, DATS introduces two primitives that act as robust declassifiers to enforce noninterference across containers, taking large applications out of the trusted computing base.

7.2 | Encryption of image layers and of data volume

Another issue with Docker is the lack of native support for the encryption of image layers to protect sensitive data from malicious privileged users (eg, registry administrator and cloud provider) and the encryption of the data volumes. Moreover, for the data security challenge, the most encouraging approach is the use of the SGX support. For container images, the selective layer encryption is another possibility.

In the work of Giannakopoulos et al,[101] the authors propose and demonstrate a mechanism for secure Docker image manipulation throughout its life cycle: mechanisms for encryption/decryption at-rest and on-the-fly are used to protect confidentiality all the time, during creation, storage, and usage of a Docker image. Distribution and migration of images has been enhanced with a mechanism that selectively encrypts only the layers of the filesystem that require strong confidentiality. SCONE[92] supports filesystem shields, preventing low-level attacks such as the OS kernel controlling pointers and buffer sizes passed to the services and ensuring the confidentiality and integrity of the application data passed through the OS. With the filesystem shield, it is possible to encrypt files or to authenticate the access to files. Moreover, SCONE provides a support for encryption and authentication of the ephemeral container's filesystem. SCONE is used in SERECA,[93] which runs MongoDB and MySQL into SCONE secure containers. In that way, the data is encrypted in memory and on the disk.

Another solution that leverages Intel SGX enclaves for disk encryption is presented in the work of Richter et al[91] and has proved to be robust to attacks to single-level and mult level authentication. This solution could be used for encrypting data volumes and images. Unfortunately, it introduces a significant overhead w.r.t. traditional disk encryption.

7.3 | Network security

Portability is one of the main advantages of containers; hence, the mechanism for interconnecting containers should be independent from the cloud service provider (or platform). Moreover, in fog computing or federated cloud scenarios the application components could communicate intercloud. The intercloud connectivity highlights the need for NAT and firewall traversal support in the containers and the need for secure end-to-end connectivity.

The network security challenges have been addressed in different ways, depending on the type of attack considered. The solutions independent from the specific attack and that seem to offer the stronger protection are based on SGX enclaves.

In the work of Ranjbar et al,[98] the authors propose a solution, named SynAPTIC, to provide secure and persistent connectivity for containers. The proposed solution is based on the Host Identity Protocol (HIP), which enables strong authentication based on public key encryption. SynAPTIC integrates HIP in Docker containers. It does not require any changes in networking infrastructure and supports the use of Software Defined Networking. The Docker daemon is usually controlled through a Unix socket, but this can be changed to a TCP socket to grant attacker remote control to run any container in privileged mode. The only remedy is to enable by default TLS connections. SCONE,[92] before mentioned, wraps all socket operations and redirects them to a network shield. The network shield, upon establishing a new connection, performs a TLS handshake and encrypts/decrypts any data transmitted through the socket. This approach is transparent to the client and server. The private key and certificate are read from the container's filesystem and can be protected by the filesystem shield (above mentioned).

SERECA[93] addresses the network security challenge by creating a secure event bus and encryption service leveraging SGX enclaves and lets them communicate with the event bus. In SecureCloud,[94] the secure communication among microservices (deployed into containers) is supported by Intel SGX enclaves. The proposed solution protects the exchanged messages by encrypting them inside of enclaves via an event bus. Encryption and decryption are automatically performed within enclaves. Decryption keys are stored securely within enclaves only. In the work of Riella et al,[97] SecureCloud is used to deploy a secure metering system.

The problem of mitigating network side channel leakage in stream processing applications is addressed in the work of Bilal et al.[102] The authors proposed a multicast and an any-cast technique to make the communication obvious and to mitigate leakage at interstage communication. That solution can be used to increase the security of any type of application and specifically of container-based data-stream processing applications.

8 | DISCUSSION AND FINAL REMARKS

Since 2014, a considerable research effort has been devoted to improve container technologies. Many application areas found attractive the advantages offered by containers with respect to consolidated virtualization technologies, ie, virtual machines. This survey reviewed the use of containers in High Performance Computing, Big Data analytics, and geo-distributed (Fog, Edge, IoT) applications. These are different fields but with common challenges: performance, scalability, high availability, isolation, data, and network security. Moreover, there are also specific challenges, for example, in High Performance Computing, there is the need to integrate parallel programming libraries with containers such as Docker. Scheduling of containerized I/O-intensive jobs is also a challenge. Furthermore, I/O performance limitation of containers should be mitigated or eliminated at all. Concerning Big Data applications, data stream and batch processing applications have performance as the key requirement and there is a need of run-time adaption algorithms to optimally deploy the required computing elements and to scale the ones with higher demand in case of surge of workload. Edge and fog computing applications are deployed on geo-distributed infrastructures; hence, they demand for deployment strategies and run-time adaptation mechanisms that take into account latency and time to download images from repositories. In that case, there is also the problem of guaranteeing trusted geo-distributed image repositories. Internet of Things applications mainly demand for energy efficient deployment and run-time adaptation strategies.

The literature works on *performance comparison* give an important contribution in demonstrating the small performance footprint of containers and the advantages with respect to virtual machines. Moreover, limitations in terms of I/O overhead are found and remedies are proposed. Other performance aspects investigated in literature classify as *Performance Monitoring*, *Characterization*, *and Prediction*. They focus on performance prediction models, multilayer monitoring solutions, characterization of the performance contention for multitenant-intensive I/O workloads, and impact of different filesystems on the container performances. All these research works provide results and suggestions to fully exploit the advantages of containers.

Run-time adaptation challenges are investigated by research work on *auto-scaling*, *high-availability*, and *scheduling*. The auto-scaling and high-availability solutions are mainly developed for big data and analytics applications running on cloud, edge, and fog infrastructures (and more in general for fog and IoT applications). Among the auto-scaling solutions, the most promising mix vertical and horizontal scaling and/or migration. Usually, the auto-scaling algorithms are designed for locally-distributed adaptation, but solutions that consider the geo-distributed case, suitable for IoT applications, are proposed as well. Different approaches aim to enhance the solutions implemented in tools like Docker Swarm and Kubernetes, like threshold-based algorithms, and algorithms based on mathematical programming or Ant Colony Optimization. The effort on high-availability focuses on enhancing the self-healing capabilities for container orchestrators like Docker Swarm and Kubernetes. The most promising approach is to make the detection of faulty nodes more sophisticated rather than based only on network level or application level probes.

The scheduling solutions are mainly intended for high performance computing applications and Internet-of-things applications. Scheduling challenges are addressed using different techniques to produce optimal schedules that take into account resource usage and energy consumption. Because of the poor network I/O performance of container, scheduling algorithms try to optimize the schedule of network I/O intensive workloads. Scheduling solutions try also to be energy efficient in the case of IoT applications. Solutions capping the energy at application level rather than at hardware level are promising.

Finally, *security*-related research results largely focus on how to improve container isolation by means of hardening the configuration of existing mechanisms or proposing new solutions based on Intel SGX. Resilience to different type of attacks (like covert channels and root escape) is also evaluated, and solutions to prevent and mitigate such attacks are proposed. Another security issue is to preserve the confidentiality of data storage in image layers. The proposed solutions are mainly techniques to encrypt/decrypt at rest and at run-time the image layers. Finally, the secure communication among container is evaluated and solutions or protocols to secure the communication are proposed. Results obtained leveraging the Intel-SGX enclaves are encouraging also because they allow to address all the challenges: isolation, data encryption, and network security. Hardening techniques are easier to implement but weaker than SGX-based solutions.

To conclude, from the analysis of the literature presented in this paper, considering also how research works are distributed among topics (cf. Figure 3), it emerges that more effort should be devoted to solve I/O performance, multilayer adaptation, energy efficiency, and security challenges. Probably, security is one of the most urgent needs, and security mechanisms should be integrated by design in the most popular container managers and orchestrators like Docker and Kubernets. While security solutions should be application and platform agnostic, it makes sense that the energy efficiency, run-time adaptation, and high availability solutions are tailored to specific deployments, eg, geo-distributed versus locally-distributed, and workloads, eg, batch versus stream processing.

ACKNOWLEDGMENTS

The work of E. Casalicchio is funded by the following research grants: *Scalable resource-efficient systems for big data analytics* under grant 20140032, Knowledge Foundation, Sweden, *PRIvacy-preserving, Security, and MAchine-learning techniques for healthcare applications (PRISMA)*, (RG11816436C00DBA) Sapienza University of Rome, Italy, and *SmartDefense: Models, Algorithms and Mechanisms for Reducing Cyber Risks in Smart Industry* (RG11916B88C838E8) Sapienza University of Rome, Italy.

ORCID

Emiliano Casalicchio https://orcid.org/0000-0002-3118-5058
Stefano Iannucci https://orcid.org/0000-0001-7485-9772

REFERENCES

1. Dua R, Raja AR, Kakadia D. Virtualization vs containerization to support PaaS. In: Proceedings of 2014 IEEE International Conference on Cloud Engineering; 2014; Boston, MA.
2. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J. Borg, omega, and kubernetes. *ACM Queue*. 2016;14:70-93.
3. Vogels W. Under the hood of Amazon EC2 container service. http://www.allthingsdistributed.com/2015/07/under-the-hood-of-the-amazon-ec2-container-service.html. 2015.
4. Natarajan S, Ghanwani A, Krishnaswamy D, Krishnan R, Willis P, Chaudhary A. *An Analysis of Container-Based Platforms for NFV*. Technical Report. Vancouver, Canada: IETF; 2016.
5. Nguyen D-T, Yong CH, Pham X-Q, Nguyen H-Q, Loan TTK, Huh E-N. An index scheme for similarity search on cloud computing using mapreduce over docker container. In: Proceedings of the 10th ACM International Conference on Ubiquitous Information Management and Communication; 2016; New York, NY.
6. Nardelli M, Hochreiner C, Schulte S. Elastic provisioning of virtual machines for container deployment. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion; 2017; L'Aquila, Italy.
7. Gerlach W, Tang W, Keegan K, et al. Skyport: container-based execution environment management for multi-cloud scientific workflows. In: Proceedings of the 5th IEEE International Workshop on Data-Intensive Computing in the Clouds; 2014; New Orleans, LA.
8. Priedhorsky R, Randles T. Charliecloud: unprivileged containers for user-defined software stacks in HPC. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis; 2017; Denver, CO.
9. Ismail BI, Goortani EM, Ab Karim MB, et al. Evaluation of docker as edge computing platform. Paper presented at: 2015 IEEE Conference on Open Systems (ICOS); 2015; Bandar Melaka, Malaysia.
10. Ma L, Yi S, Li Q. Efficient service handoff across edge servers via docker container migration. In: Proceedings of the Second ACM/IEEE Symposium on Edge Computing; 2017; San Jose, CA.
11. Asnaghi A, Ferroni M, Santambrogio MD. Dockercap: a software-level power capping orchestrator for docker containers. Paper presented at: 2016 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC) and 15th International Symposium on Distributed Computing and Applications for Business Engineering (DCABES); 2016; Paris, France.
12. Celesti A, Fazio M, Giacobbe M, Puliafito A, Villari M. Characterizing cloud federation in IoT. Paper presented at: 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA); 2016; Crans-Montana, Switzerland.

13. McGee J. The 6 steps of the container lifecycle. https://www.ibm.com/blogs/cloud-computing/2016/02/the-6-steps-of-the-container-lifecycle/. 2016.
14. Felter W, Ferreira A, Rajamony R, Rubio J. *An Updated Performance Comparison of Virtual Machines and Linux Containers*. Technical Report No. RC25482(AUS1407-001). Austin, TX: IBM Research Division, Austin Research Laboratory; 2014.
15. Morabito R, Kjällman J., Komu M. Hypervisors vs. lightweight virtualization: a performance comparison. Paper presented at: 2015 IEEE International Conference on Cloud Engineering; 2015; Tempe, AZ.
16. Kozhirbayev Z, Sinnott RO. A performance comparison of container-based technologies for the cloud. *Future Gener Comput Syst*. 2017;68:175-182.
17. Casalicchio E, Perciballi V. Measuring docker performance: what a mess!!! In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion; 2017; L'Aquila, Italy.
18. OASIS. Topology and orchestration specification for cloud applications. OASIS Standard. Version 1.0. 2013.
19. Di Martino B, Cretella G, Esposito A. Advances in applications portability and services interoperability among multiple clouds. *IEEE Cloud Comput*. 2015;2(2):22-28.
20. Casalicchio E. Container orchestration: a survey. In: Puliafito A, Trivedi K, eds. *Systems Modeling: Methodologies and Tools*. Cham, Switzerland: Springer, Cham; 2019. *EAI/Springer Innovations in Communication and Computing*.
21. Cerny T, Donahoo MJ, Trnka M. Contextual understanding of microservice architecture: current and future directions. *SIGAPP Appl Comput Rev*. 2018;17(4):29-45.
22. Pike R, Presotto D, Thompson K, Trickey H, Winterbottom P. The use of name spaces in plan 9. *SIGOPS Oper Syst Rev*. 1993;27(2):72-76.
23. Biederman EW. Multiple instances of the global Linux namespaces. Paper presented at: 2006 Ottawa Linux Symposium; 2006; Ottawa, Canada.
24. Zadok E, Iyer R, Joukov N, Sivathanu G, Wright CP. On incremental file system development. *ACM Trans Storage (TOS)*. 2006;2(2):161-196.
25. Davies A, Orsaria A. Scale out with glusterFS. *Linux Journal*. 2013;2013(235):1.
26. Maltzahn C, Molina-Estolano E, Khurana A, Nelson AJ, Brandt SA, Weil S. Ceph as a scalable alternative to the hadoop distributed file system. *Login USENIX Mag*. 2010;35:38-49.
27. Laurén S, Memarian MR, Conti M, Leppänen V. Analysis of security in modern container platforms. *Research Advances in Cloud Computing*. Singapore: Springer; 2017:351-369.
28. Casalicchio E. Autonomic orchestration of containers: problem definition and research challenges. Paper presented at: 10th EAI International Conference on Performance Evaluation Methodologies and Tools; 2016; Taormina, Italy.
29. Azab A. Enabling docker containers for high-performance and many-task computing. Paper presented at: 2017 IEEE International Conference on Cloud Engineering (IC2E); 2017; Vancouver, Canada.
30. Medvedev D, Lemson G, Rippin M. Sciserver compute: bringing analysis close to the data. In: Proceedings of the 28th International Conference on Scientific and Statistical Database Management; 2016; Budapest, Hungary.
31. Schipp J, Dopheide J, Slagell A. Islet: an isolated, scalable, & lightweight environment for training. In: Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure; 2015; St. Louis, MO.
32. Rufino J, Alam M, Ferreira J, Rehman A, Tsang KF. Orchestration of containerized microservices for IIoT using docker. Paper presented at: 2017 IEEE International Conference on Industrial Technology (ICIT); 2017; Toronto, Canada.
33. Bayser M, Cerqueira R. Integrating MPI with docker for HPC. Paper presented at: 2017 IEEE International Conference on Cloud Engineering (IC2E); 2017; Vancouver, Canada.
34. Wrede F, von Hof V. Enabling efficient use of algorithmic skeletons in cloud environments: container-based virtualization for hybrid CPU-GPU execution of data-parallel skeletons. In: Proceedings of the ACM Symposium on Applied Computing; 2017; Marrakech, Morocco.
35. Ditter A, Graf G, Fey D. Fe2vcl2: from bare metal to high performance computing on virtual clusters and cloud infrastructure. In: Proceedings of the 4th ACM Workshop on CrossCloud Infrastructures & Platforms; 2017; Belgrade, Serbia.
36. Wu Y, Rao R, Hong P, Ma J. Fas: a flow aware scaling mechanism for stream processing platform service based on LMS. In:Proceedings of the 2017 ACM International Conference on Management Engineering, Software Engineering and Service Sciences; 2017; Wuhan, China.
37. Kyong J, Jeon J, Lim S-S. Improving scalability of apache spark-based scale-up server through docker container-based partitioning. In: Proceedings of the 6th ACM International Conference on Software and Computer Applications; 2017; Bangkok, Thailand.
38. Chen Y-W, Hung S-H, Tu C-H, Yeh CW. Virtual hadoop: Mapreduce over docker containers with an auto-scaling mechanism for heterogeneous environments. In: Proceedings of the International Conference on Research in Adaptive and Convergent Systems; 2016; Odense, Denmark.
39. Holdgraf C, Culich A, Rokem A, Deniz F, Alegro M, Ushizima D. Portable learning environments for hands-on computational instruction: using container- and cloud-based technology to teach data science. In: Proceedings of the ACM Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact; 2017; New Orleans, LA.
40. Julian S, Shuey M, Cook S. Containers in research: initial experiences with lightweight infrastructure. In: Proceedings of the ACM Conference on Diversity, Big Data, and Science at Scale (XSEDE16); 2016; Miami, FL.
41. Baig R, Freitag F, Navarro L. Fostering collaborative edge service provision in community clouds with docker. In: 2016 IEEE International Conference on Computer and Information Technology (CIT); 2016; Nadi, Fiji.
42. Sathiaseelan A, Lertsinsrubtavee A, Jagan A, Baskaran P, Crowcroft J. Cloudrone: micro clouds in the sky. In: Proceedings of the 2nd ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use; 2016; Singapore.
43. Ma K, Yang B, Yang Z, Yu Z. Segment access-aware dynamic semantic cache in cloud computing environment. *J Parallel Distrib Comput*. 2017;110:42-51.
44. Ma K, Dong F, Yang B. Large-scale schema-free data deduplication approach with adaptive sliding window using mapreduce. *Comput J*. 2015;58(11):3187-3201.
45. Bellavista P, Zanni A. Feasibility of fog computing deployment based on docker containerization over raspberrypi. In: Proceedings of the 18th ACM International Conference on Distributed Computing and Networking; 2017; Hyderabad, India.
46. Perrin M, Mullen J, Helff F, Gruenwald L, d'Orazio L. Time-, energy-, and monetary cost-aware cache design for a mobile-cloud database system. In: *Biomedical Data Management and Graph Online Querying*. Cham, Switzerland: Springer; 2015:71-85.
47. Ren Q, Dunham MH, Kumar V. Semantic caching and query processing. *IEEE Trans Knowl Data Eng*. 2003;15(1):192-210.

48. Avino G, Malinverno M, Malandrino F, Casetti C, Chiasserini CF. Characterizing docker overhead in mobile edge computing scenarios. In: Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems; 2017; Los Angeles, CA.
49. Shirinbab S, Lundberg L, Casalicchio E. Performance evaluation of container and virtual machine running cassandra workload. Paper presented at: The 3rd IEEE International Conference on Cloud Computing Technologies and Applications (IEEE CloudTech17); 2017; Rabat, Marocco.
50. Balasubramanian SV, Patil V, Giusti M, Bhide A, Gupta A. AWS EC2 vs. joyent's triton: a comparison of docker container-hosting platforms. In: Proceedings of the 8th ACM Workshop on Scientific Cloud Computing; 2017; Washington, DC.
51. Adufu T, Choi J, Kim Y. Is container-based technology a winner for high performance scientific applications? Paper presented at: 17th IEEE Asia-Pacific Network Operations and Management Symposium, APNOMS; 2015; Busan, South Korea.
52. Tarasov V, Rupprecht L, Skourtis D, et al. In search of the ideal storage configuration for docker containers. Paper presented at: 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W); 2017; Tucson, AZ.
53. Dua R, Kohli V, Patil S, Patil S. Performance analysis of union and cow file systems with docker. Paper presented at: 2016 International Conference on Computing, Analytics and Security Trends (CAST); 2016; Pune, India.
54. Xu Q, Awasthi M, Malladi KT, Bhimani J, Yang J, Annavaram M. Docker characterization on high performance ssds. Paper presented a: 2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS); 2017; Santa Rosa, CA.
55. Morabito R. A performance evaluation of container technologies on internet of things devices. Paper presented at: 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS); 2016; San Francisco, CA.
56. Morabito R. Power consumption of virtualization technologies: an empirical investigation. Paper presented at: 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC); 2015; Limassol, Cyprus.
57. Khazaei H, Ravichandiran R, Park B, Bannazadeh H, Tizghadam A, Leon-Garcia A. Elascale: autoscaling and monitoring as a service. In: Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering (CASCON17)2017; Markham, Canada.
58. Taherizadeh S, Stankovski V. Incremental learning from multi-level monitoring data and its application to component based software engineering. Paper presented at: 41st IEEE Annual Computer Software and Applications Conference (COMPSAC); 2017; Turin, Italy.
59. Nardelli M, Cardellini V, Casalicchio E. Multi-level elastic deployment of containerized applications in geo-distributed environments. Paper presented at: 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud); 2018; Barcelona, Spain.
60. Stankovski V, Trnkoczy J, Taherizadeh S, Cigale M. Implementing time-critical functionalities with a distributed adaptive container architecture. In: Proceedings of the 18th ACM International Conference on Information Integration and Web-based Applications and Services; 2016; Singapore.
61. Casalicchio E, Perciballi V. Auto-scaling of containers: the impact of relative and absolute metrics. Paper presented at: 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W); 2017; Tucson, AZ.
62. Kim W-Y, Lee J-S, Huh E-N. Study on proactive auto scaling for instance through the prediction of network traffic on the container environment. In: Proceedings of the 11th ACM International Conference on Ubiquitous Information Management and Communication; 2017; Beppu, Japan.
63. Zhao D, Mohamed M, Ludwig H. Locality-aware scheduling for containers in cloud computing. *IEEE Trans Cloud Comput*. 2018. Early Access.
64. Kaewkasi C, Chuenmuneewong K. Improvement of container scheduling for docker using ant colony optimization. Paper presented at: 2017 9th International Conference on Knowledge and Smart Technology (KST); 2017; Chonburi, Thailand.
65. Abdelbaky M, Diaz-Montes J, Parashar M. Towards distributed software-defined environments. In: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing; 2017; Madrid, Spain.
66. Guan X, Wan X, Choi BY, Song S, Zhu J. Application oriented dynamic resource allocation for data centers using docker containers. *IEEE Commun Lett*. 2017;21(3):504-507.
67. Abdelbaky M, Diaz-Montes J, Parashar M, Unuvar M, Steinder M. Docker containers across multiple clouds and data centers. Paper presented at: 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC); 2015; Limassol, Cyprus.
68. Nathan S, Ghosh R, Mukherjee T, Narayanan K. Comicon: a co-operative management system for docker container images. Paper presented at: 2017 IEEE International Conference on Cloud Engineering (IC2E); 2017; Vancouver, Canada.
69. Kangjin W, Yong Y, Ying L, Hanmei L, Lin M. Fid: a faster image distribution system for docker platform. Paper presented at: 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W); 2017; Tucson, AZ.
70. Al-Dhuraibi Y, Paraiso F, Djarallah N, Merle P. Autonomic vertical elasticity of docker containers with ELASTICDOCKER. Paper presented at: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD); 2017; Honolulu, CA.
71. Ye K, Ji Y. Performance tuning and modeling for big data applications in docker containers. Paper presented at: 2017 International Conference on Networking, Architecture, and Storage (NAS); 2017; Shenzhen, China.
72. McDaniel S, Herbein S, Taufer M. A two-tiered approach to I/O quality of service in docker containers. Paper presented at: 2015 IEEE International Conference on Cluster Computing; 2015; Chicago, IL.
73. Truyen E, Van Landuyt D, Reniers V, Rafique A, Lagaisse B, Joosen W. Towards a container-based architecture for multi-tenant SaaS applications. In: Proceedings of the 15th ACM International Workshop on Adaptive and Reflective Middleware; 2016; Trento, Italy. http://doi.acm.org.miman.bib.bth.se/10.1145/3008167.3008173
74. Barna C, Khazaei H, Fokaefs M, Litoiu M. Delivering elastic containerized cloud applications to enable devOps. Paper presented at: 12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS); 2017; Buenos Aires, Argentina.
75. Stubbs J, Moreira W, Dooley R. Distributed systems of microservices using docker and serfnode. Paper presented at: 2015 7th International Workshop on Science Gateways; 2015; Budapest, Hungary.
76. Naik N. Applying computational intelligence for enhancing the dependability of multi-cloud systems using docker swarm. Paper presented at: 2016 IEEE Symposium Series on Computational Intelligence (SSCI); 2016; Athens, Greece.
77. Tadesse SS, Malandrino F, Chiasserini CF. Energy consumption measurements in docker. Paper presented at: IEEE 41st Annual Computer Software and Applications Conference (COMPSAC); 2017; Turin, Italy.
78. Pahl C. Containerization and the PaaS cloud. *IEEE Cloud Comput*. 2015;2(3):24-31.
79. Huebscher MC, McCann JA. A survey of autonomic computing - degrees, models, and applications. *ACM Comput Surv*. 2008;40(3):7:1-7:28. http://doi.acm.org/10.1145/1380584.1380585

80. Lemos AL, Daniel F, Benatallah B. Web service composition: a survey of techniques and tools. *ACM Comput Surv*. 2015;48(3):33:1-33:41. http://doi.acm.org/10.1145/2831270
81. Faniyi F, Bahsoon R. A systematic review of service level management in the cloud. *ACM Comput Surv*. 2015;48(3):43:1-43:27. http://doi.acm.org/10.1145/2843890
82. Singh S, Chana I. Qos-aware autonomic resource management in cloud computing: a systematic review. *ACM Comput Surv*. 2015;48(3):42:1-42:46.
83. He S, Guo L, Guo Y, Wu C, Ghanem M, Han R. Elastic application container: a lightweight approach for cloud resource provisioning. Paper presented at: 2012 IEEE 26th International Conference on Advanced Information Networking and Applications; 2012; Fukuoka, Japan.
84. Casalicchio E. A study on performance measures for auto-scaling CPU-intensive containerized applications. *Cluster Computing*. 2019.
85. Combe T, Martin A, Pietro RD. To docker or not to docker: a security perspective. *IEEE Cloud Computing*. 2016;3(5):54-62.
86. Samuel J, Mathewson N, Cappos J, Dingledine R. Survivable key compromise in software update systems. In: Proceedings of the 17th ACM Conference on Computer and Communications Security; 2010; Chicago, IL.
87. Kuppusamy TK, Torres-Arias S, Diaz V, Cappos J. Diplomat: using delegations to protect community repositories. Paper presented at: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16); 2016; Santa Clara, CA.
88. Shu R, Gu X, Enck W. A study of security vulnerabilities on docker hub. In: Proceedings of the 7th ACM on Conference on Data and Application Security and Privacy; 2017; Scottsdale, AZ.
89. Hassan F, Rodriguez R, Wang X. Rudsea: recommending updates of dockerfiles via software environment analysis. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering; 2018; Montpellier, France.
90. Syed MH, Fernandez EB. A reference architecture for the container ecosystem. In: Proceedings of the 13th ACM International Conference on Availability, Reliability and Security; 2018; Hamburg, Germany.
91. Richter L, Götzfried J, Müller T. Isolating operating system components with intel SGX. In: Proceedings of the 1st ACM Workshop on System Software for Trusted Execution; 2016; Trento, Italy.
92. Arnautov S, Trach B, Gregor F, et al. Scone: secure Linux containers with intel SGX. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation; 2016; Savannah, GA.
93. Fetzer C, Mazzeo G, Oliver J, Romano L, Verburg M. Integrating reactive cloud applications in sereca. In: Proceedings of the 12th ACM International Conference on Availability, Reliability and Security; 2017; Reggio Calabria, Italy.
94. Kelbert F, Gregor F, Pires R, Köpsell S. Securecloud: secure big data processing in untrusted clouds. In: Proceedings of the Conference on Design, Automation & Test in Europe, European Design and Automation Association; 2017; Lausanne, Switzerland.
95. Krieter S, Krüger J, Weichbrodt N, Sartakov VA, Kapitza R, Leich T. Towards secure dynamic product lines in the cloud. In: Proceedings of the 40th ACM International Conference on Software Engineering: New Ideas and Emerging Results; 2018; Gothenburg, Sweden.
96. Havet A, Pires R, Felber P, Pasin M, Rouvoy R, Schiavoni V. Securestreams: a reactive middleware framework for secure data stream processing. In: Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (DEBS '17); 2017; Barcelona, Spain.
97. Riella RJ, Iantorno LM, Luciana R Jr, et al. Securing smart metering applications in untrusted clouds with the securecloud platform. In: Proceedings of the 1st ACM Workshop on Privacy by Design in Distributed Systems; 2018; Porto, Portugal.
98. Ranjbar A, Komu M, Salmela P, Aura T. Synaptic: secure and persistent connectivity for containers. In: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing; 2017; Madrid, Spain.
99. Luo Y, Luo W, Sun X, Shen Q, Ruan A, Wu Z. Whispers between the containers: high-capacity covert channel attacks in docker. Paper presented at: 2016 IEEE Trustcom/BigDataSE/ISPA; 2016; Tianjin, China.
100. MP AR, Kumar A, Pai SJ, Gopal A. Enhancing security of docker using Linux hardening techniques. Paper presented at: 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT); 2016; Bangalore, India.
101. Giannakopoulos I, Papazafeiropoulos K, Doka K, Koziris N. Isolation in docker through layer encryption. Paper presented at: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS); 2017; Atlanta, GA.
102. Bilal M, Alsibyani H, Canini M. Mitigating network side channel leakage for stream processing systems in trusted execution environments. In: Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems; 2018; Hamilton, New Zealand.
103. Hunger C, Vilanova L, Papamanthou C, Etsion Y, Tiwari M. DATS - data containers for web applications. *ACM SIGPLAN Not*. 2018;53(2):722-736.
104. Jian Z, Chen L. A defense method against docker escape attack. In: Proceedings of the 2017 International Conference on Cryptography, Security and Privacy; 2017; Wuhan, China.
105. Oracle. System administration guide: oracle solaris containers-resource management and oracle solaris zones. Oracle Corporation 500 Oracle Parkway Redwood City, CA 94065 U.S.A.

How to cite this article: Casalicchio E, Iannucci S. The state-of-the-art in container technologies: Application, orchestration and security. *Concurrency Computat Pract Exper*. 2020;32:e5668. https://doi.org/10.1002/cpe.5668