A Qualitative Cross-Comparison of Emerging Technologies for Software-Defined Systems

Awais Aziz Shah, Giuseppe Piro, Luigi Alfredo Grieco, Gennaro Boggia
Dept. of Electrical and Information Engineering (DEI),
Politecnico di Bari, v. Orabona 4, 70125, Bari, Italy;
Email: {awais.shah, giuseppe.piro, alfredo.grieco, gennaro.boggia}@poliba.it
CNIT, Consorzio Nazionale Interuniversitario per le Telecomunicazioni

***Abstract*—Software-Defined Systems are offering great opportunities for deploying programmable networks, as well as large-scale services distributed across clouds. Starting from the baseline principles of both Software-Defined Networking and Network Function Virtualization, they embrace a number of novel enabling technologies (like containers, container orchestrators, and many other supporting tools) that significantly simplify the integration and the management of virtual components, while promising high level of flexibility, isolation, and performance. The major tech giants are drastically building their business on these technologies. However, many other companies are struggling in the selection of suitable platforms, tools, and any other software instruments allowing them to move forwards in this direction. Based on these premises, this paper provides a three-folded contribution. First, it explores the state of the art of container engines and container orchestrators. Second, it analyzes the main supporting tools that offer advanced and additional features to the resulting container networking. Third, it defines a set of qualitative Key Performance Indicators to carry out a preliminary comparison of the reviewed technologies. The proposed study aims at providing high-level guidelines and constructive comments to foster the widespread usage of Software-Defined Systems.**

***Index Terms*—Software-Defined Systems, Containers, Container Orchestrators, Supporting Tools, Cross-Comparison**

I. Introduction

During the last decade, Software-Defined Networking (SDN) and Network Function Virtualization (NFV) introduced a revolutionary way to deploy programmable network architectures, based on a strict separation between data plane and control plane and a native ability to dynamically define, install, and configure virtualized network facilities [1], [2]. At the same time, the potential of the cloud-computing supported the design of advanced services and applications, leveraging virtual components distributed at the large scale. Indeed, the joint integration of SDN, NFV, and cloud-computing principles recently paved the way towards the definition of Software-Defined Systems (SDS) [3].

In the SDS, vision, resources, services, and applications are treated as a combination of virtual boxes, interacting with each other through the underlying communication infrastructure. Their management demands high levels of flexibility, isolation, and performance [4]–[6]. Since virtualization techniques based on traditional Virtual Machines (VMs) cannot meet these requirements [7], other enabling technologies are gaining momentum in the current state of the art. They include, for example, lightweight containers, container orchestrators, and many other supporting tools [8]–[11]. Therefore, as a metter of fact, SDS systems are significantly grounding their roots into the container networking paradigm [12]–[14].

At the time of this writing, the major tech giants (like Google, IBM, Amazon, and so on) are drastically adopting these technologies to build and manage their large scale infrastructures [15]. On the other hand, however, many other organizations, especially small and medium enterprises, are reluctant to launch their services through the SDS concept. The reason is that they generally experience hard difficulties in the selection of containerization softwares and tools that satisfy their business needs and visions [16]. To make things worse, a variety of containerization tools and management instruments are continuously emerging, developers are constantly introducing new features in their latest updates. Accordingly, their integration within an operating framework still appears as an important barrier to face.

Starting from these premises, the work presented herein intends to shed some light on the key enabling technologies of SDS based on container networking. Specifically, the conducted study provides a three-folded contribution. First, it explores containerization technologies, including both container engines (i.e., LXC, Docker, LXD, Rkt, and Kata container) and orchestrators (i.e., kubernetes, docker compose, docker swarm, OpenStack, Nomad, Apache Mesos, Bistro, ECS, Cloud Foundry, and OpenShift). Second, it analyzes the main supporting tools that offer advanced and additional features to SDS systems, such as load balancing, service discovery, user interfaces, and communication protocols. Third, it defines a set of qualitative Key Performance Indicators (KPIs) through which carrying out a preliminary comparison of the reviewed key enabling technologies. The resulting analysis clearly shows pros and cons arising from an integration of containers, container orchestrators and supporting tools and provides high-level guidelines and constructive comments to foster the widespread usage of SDS in the near future.

The rest of this paper is organized as in what follows. Section II presents containerization engines and orchestrators. Section III reviews the related supporting tools. Section IV discusses the proposed qualitative cross-comparison. Finally, Section V concludes the work and presents future research directions.

II. Containerization Technologies

As anticipated in the previous Section, the main rationale behind upcoming SDS is based on container networking. As depicted in Figure 1, resources, services, and applications are implemented as containers and distributed across network elements and clouds. Differently from traditional VMs, containers installed over the same physical machine share the same operating system. But, at the same time, they may still experience a good level of isolation. Also, local operations, like loading and turn-off procedures, could register limited latencies. On the other hand, a logically centralized orchestrator automates the management of containers lifecycle. This Section reviews containerization technologies available in the current state of the art.

A. Container Engines

Popular container engines are: LXC, LXD, Docker, Rkt, and Kata Containers.

1) LXC(Linux Containers): It is an open-source software supported by Canonical Limited and in active development since 2008. Its first stable Version 1.0 was released on February 20, 2014. Containers in LXC rely on the Linux kernel containment features [8]. This functionality allows the user to run multiple images on a single host. The isolation between the containers is provided through kernel namespaces. Resource management, instead, is done through cgroup [16]. It supports multiple applications in a container and provides partial portability across Ubuntu distributions only [16]. These features make LXC more performant than standard VMs [17].

2) Docker: It is one of the leading containerization tools in the industry. The initial version of Docker was released on March 13, 2013. It uses the features of Linux kernel namespaces and cgroups to completely isolate applications and the underlying operating system. It also allows to run a single process in a container [8], [16], [18]. Docker extends the Linux container technology by creating portable, easy to use, and flexible images [18]–[20]. It creates self-elastic clusters with a size managed according to the workload and provides enhanced elasticity because of the fast loading speed of container [11].

3) LXD: Built on top of LXC, it provides new and better user experience. It is a free and open-source software, built under the Apache 2.0 license[1]. The initial Version 0.1 of LXD was released on February 13, 2015. Differently from LXC, it offers new features like a new single command line tool to enhance user experience for containers management. Moreover, multiple applications can run in a single container [21]. LXC containers can be easily used through REST API and a CLI clients.

4) Rkt: Its first version was released on November 27, 2014 by CoreOS (acquired by RedHat). It allows to run multiple isolated images sharing a common kernel namespace. Rkt container runtime is interoperable, secure, and open-source. It also provides security in various aspects. For example, it validates the authenticity of the image after downloading by cross checking publisher's image signature [16]. The work presented in [16] demonstrates that Rkt emerges as a suitable choice in computational and data intensive high performance application environment.

5) Kata Containers: It is an open source project available under the Apache 2.0 license. The first Version of Kata containers was released on May 22, 2018. It is a novel implementation of lightweight VMs, that seamlessly integrates within the container ecosystem. Therefore, benefits of both container and VMs are met, including high performance, workload isolation, and security. Kata Containers support the Open Containers Initiative (OCI)[2] [22].

B. Container orchestrators

Popular container orchestrators are: Kubernetes, Docker Compose, Docker Swarm, OpenStack, Nomad, Apache Mesos, Bistro, ECS, Cloud Foundry, and OpenShift.

1) Kubernetes: It is an open-source container orchestrator for automating application deployment, scaling, and management across clusters of hosts (physical or VMs) [22]. It is freely available under the Apache License 2.0. Its first available Version was released on June 7, 2014. It was developed by Google under their experience of building container management solutions [22]. It is now maintained by the Cloud Native Computing Foundation. It works with a range of technologies and provides an orchestration layer for managing Docker containers on different physical entities [10]. Kubernetes uses a master-slave architecture. The slave hosts, namely nodes, are intended for running the containers assigned by the master. It also adds an abstraction layer on top of containers, namely Pod. Specifically, it represents a group of up to five containers that share storage, networking resources, IP address. Pods form an atomic unit of scheduling and are created or destroyed automatically. They can communicate with each other without any Network Address Translation (NAT). This ensures a very easy management of a multi-host cluster [23].

2) Docker Compose: It is a solution developed by Docker for creating and running applications, including multiple Docker Containers [19]. The first production ready Version 1.0 of Docker Compose was released on October 16, 2014. It helps to configure applications and containers by using a YAML file with a single command [18].

3) Docker Swarm: It is the Docker's local clustering solution developed by Google. While the standard Docker API launchs the containers, Swarm takes care of selecting the appropriate hosts for running containers [23], [18]. It can group together several hosts, allowing the user to manage them as a unified cluster by using the Swarm CLI utility [19]. It combines multiple Docker engines into a single virtual engine. For managing and configuring services in the containers, a specific discovery service can be used with Swarm. The advantage of Swarm is that it natively incorporates Docker API

[1]https://linuxcontainers.org/lxd/

[2]OCI is a project under governance of Linux Foundation. It was started in June 2015 by Docker, CoreOS, and other leaders of containerization industry to standardize the container formats and runtime.

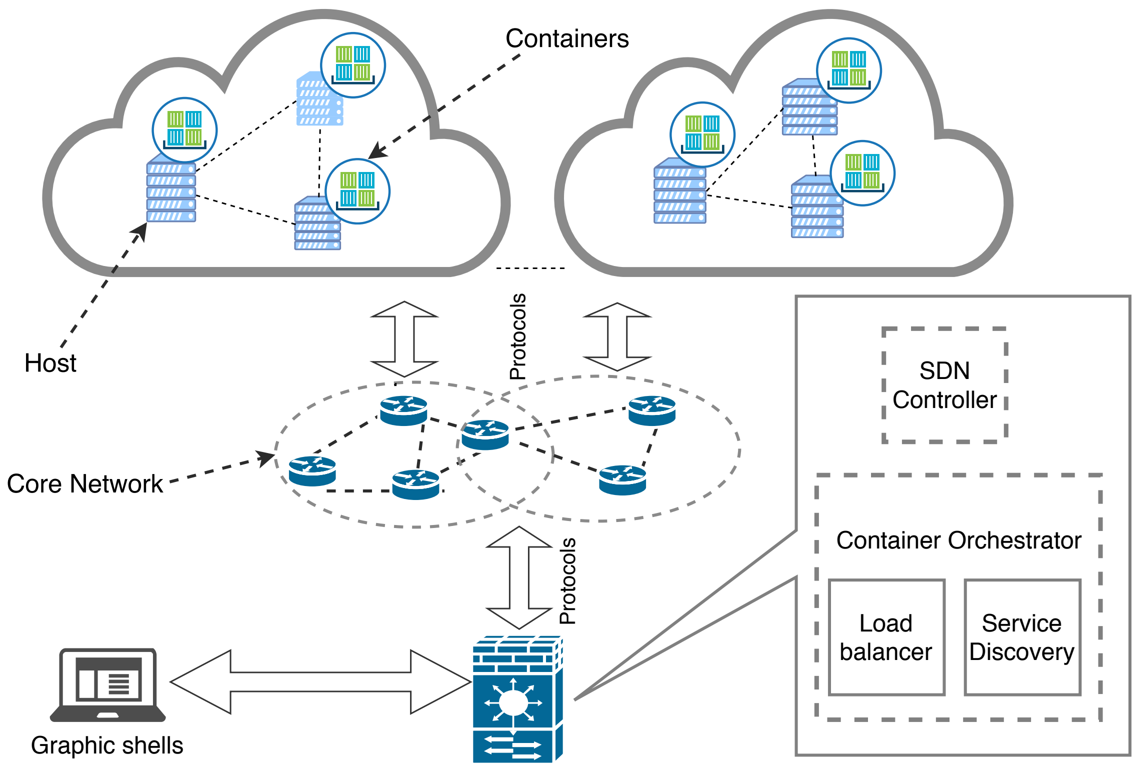

Fig. 1. Big picture of Software Defined Systems based on container networking.

calls. This consequently makes easy to move large workloads and applications to different clusters [23].

4) OpenStack: It is an open-source operating system developed to manage and control large pool of computations in the cloud. It was developed in Python by NASA and Rackspace to handle massive infrastructure in the cloud [24] [25]. Its initial version was released on October 21, 2010. It has already been used by several companies worldwide for managing Petabytes of distributed architectures, scaling to over 60 million VMs [25]. OpenStack provides many other services like multi-tenant security, monitoring, storage, and more [25]. Google Sponsors the OpenStack foundation [26]. It provides support for managing Bare-metal, VMs, and container based hosts [27] [28].

5) Nomad: Hashi group's Nomad was developed in 2015. It is an open-source scheduler that uses a declarative job file for scheduling containerized applications. It follows an agent-based architecture using single binary that is responsible for rolling upgrades and draining nodes for rebalancing [29].

6) Apache Mesos: It is an open-source and low level clustering solution, that integrates with a high level framework to provide the complete orchestration [23]. Mesos combines the resources of the cluster (like CPU, RAM etc.) in a way that looks like single giant host to the developer [29]. It was originally developed by students of UC Berkeley RAD Lab in 2009. Apache announced its Version 1.0 on July 27, 2016. It comes with a distributed system kernel that provides applications with API's for resource management and scheduling in large-scale clustered environments. It allows developers to create their applications. A job scheduler tool can be used with it for scheduling and running tasks [26]. Since Mesos acts like a kernel of the distributed OS therefore, a framework i.e., Marathon is used with it for Container Orchestration.

7) Bistro: Facebook's Bistro is a closed-source scheduler that runs data-intensive batch jobs on distributed systems. The present public release of this technology is partial, including just the server components. It is designed to handle workload efficiently and to respond rapidly to the changing configurations. This is because Facebook stores a high amount of data in different formats and frequently runs batch jobs for transformation and transfer of data [30].

8) Amazon's Elastic Container Service (ECS): It is a closed-source solution used in Amazon Web Services (AWS) for running, scaling, and securing container applications[3]. ECS can be used with any third-party hosted Docker image repository or accessible private registry, such as Docker Hub.

9) Cloud Foundry: It is an open-source, multi-cloud application platform as a service. The software was originally developed as a container-based architecture by VMware in 2011. Then, it was transferred to Pivotal Software, which is a joint venture by EMC, VMware, and General Electric. Cloud Foundry supports Docker images by connecting to the Docker registry, giving those enterprises that are already running Docker take advantage of all the platform capabilities that Cloud Foundry provides. It also supports the OCI initiative.

[3]https://aws.amazon.com/ecs/features/

10) OpenShift: Developed by the RedHat under the Apache license 2.0 on May 4, 2011. It is a combination of open-source technologies for orchestration, including RedHat Enterprise Linux, OCI-standard containers, and Kubernetes for orchestration and management. OpenShift extends to give users their choice of frameworks, databases, and runtimes. OpenShift is a part of the CNCF Certified Kubernetes program, ensuring portability and interoperability for container workloads.

III. Supporting Tools for Containerization

There are many tools supporting specific facets in container networking, including load balancing, service discovery, protocols, and user interfaces. They facilitate the orchestration and management of containers, depending on the business requirements.

A. Load balancing tools

Load balancing tools aims at spreading the inbound requests (i.e., the load) across the containers. In this way, they minimize the response time and increase the throughput [29]. The list of some popular load balancing tools for containerization are reported below.

1) Envoy: It was built at Lyft in C++. It is a distributed proxy designed for single applications and services. It may also serve as a communication bus and data plane for microservice architectures [29].

2) HAProxy and Bamboo: It is a very popular, fast, and reliable solution for load balancing, which ensure a high availability for TCP and HTTP-based applications. For years, it has become the standard load balancing tool. Thus, it is now shipped with most mainstream linux distributions. It can be integrated with almost all the existing technologies [29]. On the other hand, Bamboo is a tool that automatically configures HAProxy for web services deployed on the top of Apache Mesos and Marathon [29].

3) Kube-Proxy: It runs on every node of the kubernetes cluster. It works for the cluster's internal load balancing and service discovery [29].

4) MetalLB: It is a tool for load balancing in bare-metal Kubernetes clusters. This tool exists because Kubernetes does not offer default load balancing mechanisms for bare metal clusters [29].

5) NGINX: It is a load balancing tool for HTTP-based traffic. It offers further mechanisms for configuration and monitoring[4] [29].

6) Traefik: It is an open source tool that is gaining much popularity nowadays like HAProxy [29]. It is supported by every major cluster technology. The popular features of Traefik are the auto discovery and tracing of clusters (including Kubernetes, Mesos, Docker Swarm, Marathon, and Rancher).

7) Vamp-router: It is a service routing, load balancing, and filtering application. It updates the configurations through REST API or Zookeeper [5] [29].

[4]https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/

[5]https://github.com/magneticio/vamp-router

8) Vulcand: It is a HTTP API management and micro services tool that uses Etcd as a configuration backend. With Vulcand, changes to configuration take effect immediately without restarting the service.

B. Service discovery tools

In container networking, it is not possible to manually assign applications to high number of containers. Instead, such a task is managed by a specific software, generally referred to as scheduler, that controls the lifecycle of containers. In this context, service discovery is used to determine where the container ended up with being scheduled.

1) ZooKeeper: It is a software by Apache Software Foundation, originally developed at Yahoo. It is a key-value store for maintaining configuration information in distributed systems. ZooKeeper organizes the data in a file system (to this end, it basically uses tables, called znodes) [29]. Even if it emerges as a mature tool, its installation procedure appears complex [29]. This tool is sponsored by giants like Google, Microsoft, AWS, and Facebook.

2) Etcd: It is another key-value store developed by the CoreOS team in the GO language. The Etcd security feature allows TLS/SSL authentication between client and cluster Etcd nodes [29].

3) Consul: It is a key-value store developed by the HashiGroup in the Go language. Consul offers a multi-data center support for service registration, discovery, and health monitoring [29].

4) Mesos-DNS: It is a DNS-based customized solution for service discovery, working in Apache Mesos. Mesos-DNS is written in the Go Language. Moreover, it polls the active running processes on the Mesos architecture and exposes the running tasks through DNS and HTTP APIs [29].

5) SkyDNS: It is a DNS-based service discovery tool used with Etcd. It stores the service records in Etcd and updates their DNS records accordingly [29].

6) WeaveDNS: It is another DNS-based service discovery solution that allows containers to find other containers through their IP addresses.

7) SmartStack: The Airbnb smartstack writes the service registration in Zookeeper and dynamically configures the HAProxy for lookup [29].

8) Eureka: It was developed and deployed for the AWS (i.e., where Netflix runs) [29]. Specifically, Netflix's Eureka is a Rest-based service discovery tool used for load balancing and checks the failover of intermediary servers. It comes with its own load balancer.

C. User interfaces

Graphic User Interfaces (GUIs) and User Interfaces (UIs) provide an abstract access to the remote resources for supporting the monitoring and the management of cloud-based processes. Popular solutions include:

1) Rancher: It is an open-source software for delivering Kubernetes-as-a-Service for multi-cloud computing.

2) Clocker: It is a self-hosted and open-source container-management platform, built on the top of Apache Brooklyn. Clocker is used to easily deploy production grade Docker swarms or Kubernetes clusters to a range of clouds (including AWS, Azure, Google Cloud, IBM Softlayer, and IBM Blue-Box)[6]. Moreover, it supports a wide range of cloud providers through the use of the jclouds toolkit [23]. It can be configured with cloud though deployment tokens and access keys. After this, it can automatically detect hosts and install a network with the support of service discovery tools like Weave or Project Calico [23].

3) Tutum: It was used to build, deploy, and manage containerized applications across any cloud infrastructure. Tutum decouples the orchestration layer from the underlying infrastructure on which the application runs. Tutum works on the top of any infrastructure provider and users can choose the provider that best satisfies their requirements. Started in 2013, later acquired and integrated by Docker in 2015.

4) Portainer: It is a lightweight management UI, which allows to easily manage different Docker environments (like Docker hosts or Swarm clusters). Portainer is able to provide support for Linux, Windows, and OSX [7].

5) Kitematic: It is a powerful UI for managing containers[8]. Kitematic gives a one-click installation ability for running Docker on Mac.

6) DockStation: It is a developer-centric tool for managing Docker projects. Instead of lots of CLI commands you can monitor, configure, and manage services/containers by using a GUI. DockStation supports MacOS, Linux, Ubuntu, and Windows.

7) Panamax: It is a browser rendered GUI for pulling together image compositions. It supports Docker.

8) Docker UI: It provides a simple interface into the currently running docker VM and allows the users to browse/check the state of installed containers.

9) Docker Compose UI: It is a UI for Docker containers. Differently from other dashboards, it appears like a browser-based interface to manage deployed container compositions.

10) Shipyard: It is another UI for managing Docker Containers. Within few seconds it is ready for the login and gives the snappy looking dashboard. Shipyard has a simple installation procedure. After pulling down a script, simply run it, and Shipyard pulls down a number of images and spins them up[9].

D. Protocols

The communication between the different components in container networking is performed with the help of protocols. Main solutions are: OpenFlow, NETCONF, and RESTCONF.

[6]http://www.clocker.io

[7]https://github.com/portainer/portainer

[8]https://kitematic.com/

[9]https://dzone.com/articles/managing-docker-containers-with-shipyard

1) OpenFlow: Born in 2008, it was considered as the first standard for SDN [31]. Therefore, it is a well-known communication protocol that gives access to the physical components (i.e., routers or switches) of the data plane [32], and adapt their functionalities to changing business requirements. Today, all the standard manufacturers of network devices provide support for OpenFlow in their devices.

2) NETCONF: The Network Configuration Protocol (NETCONF) provides a simple mechanism to manage, configure and upload configurations of a network device. It allows the device to expose a full and formal API, used by the applications to receive configuration data sets [33].

3) RESTCONF: NETCONF defines the configuration data stores and a set of CRUD operations (that include create, read, update and delete) useful to access to these data stores. RESTCONF uses HTTP methods to provide CRUD operations on a conceptual datastore containing YANG-defined data [34].

IV. Cross comparison of SDS enabling technologies

In order to remark the pros and cons of SDS enabling technologies and to shed some lights on their joint usage within a more complete framework, this Section defines a set of KPIs and proposes a qualitative cross comparison.

First of all, the qualitative KPIs defined for evaluating container engines include the development language, the developing company, the community support, the reference operating system, the licensing type, the OCI compliance, and the support for IPv6. The resulting cross comparison is reported in Table I. It is evident that LXC, LXD, Docker, Rkt, and Kata containers are open-source technologies, promoted by strong community, and working with Linux. All the container engines, except LXD, are compliant with the OCI guidelines. Moreover, excepting Kata containers, all the other engines support IPv6. Among the others, Docker emerges as a powerful technology that works on multiple platforms (i.e. Linux, Windows, and MacOS).

The set of qualitative KPIs used to evaluate containers orchestrators include the development language, the developing company, the community support, the licensing type, the builtin scheduler, the native load balancing feature, the native service discovery, the reference operating system, the cloud support, the bare metal support, and the possibility to handle clustering. The resulting cross comparison is shown in Table II. The conducted study demonstrates that Kubernetes and OpenStack are the open-source orchestration technologies that come up with built-in scheduler, load balancer, service discovery native functionalities. They support multiple operating systems, cloud, bare metal, and clustering features. They also have a strong community support, backed by Google and Cloud Native Computing Foundation (CNCF). On the other hand, instead, other technologies guarantee limited features (they are used in the industry only for specific purposes). Among all the investigated solutions, Kubernetes allows a huge amount of flexibility for containerization networking. It

TABLE I
CROSS COMPARISON AMONG CONTAINERIZATION ENGINES.

Engine	**Language**	**Developers**	**Community support**	**OS Support**	**Licencing type**	**OCI**	**IPv6**
LXC	C, Python, Lua	Canonical Ltd	Canonical Ltd and Ubuntu* (Forum & Github)	Linux	Opensource, GNU LGPL v.2.1	✓	✓
LXD	Go	Canonical Ltd	Canonical Ltd and Ubuntu** (Forum & Github)	Linux	Opensource, Apache 2.0	✗	✓
Docker	Go	Docker, Inc.	Docker, Inc. (Community, Forum, Blog & Github)	Linux, Win, Mac	Opensource, Apache 2.0	✓	✓
Rkt	Go	CoreOS (REDHAT)	Red Hat Inc., Github	Linux	Opensource, Apache 2.0	✓	✓
Kata Containers	Go	The OpenStack Foundation	OpenStack foundation, Blog, Github,	Linux	Opensource, Apache 2.0	✓	✗

*LXC 1.0 will be supported until June 1st 2019 and LXC 2.0 until June 1st 2021.
**The current LTS of LXD if 3.0, which will be supported until June 2023

can be used with almost all the container engines to provide agility.

The joint usage of container engines and supporting tools is evaluated below.

Table III, for instance, focuses on load balancing tools. It clearly emerges that NGINX can be natively used with all the reviewed container engines. HA-Proxy, instead, can be used with LXC, LXD, and Docker. The rest of load balancing tools, excluding Bamboo, could be used with all the container engines thanks to the additional plugins implemented in both Kubernetes orchestrator and Etcd tool. Among the containers, Docker can work with all the considered load balancing tools. The Bamboo tool, instead, provides a very scarce support for the container engines.

Table IV illustrates the possible joint usage of service discovery tools and container engines. Also in this case, many container engines are natively or indirectly (i.e., with the usage of integration instruments provided by some container orchestrators) compatible with the reviewed service discovery tools. Nevertheless, a very scarce integration is observed for both SmartStack and Eureka.

Table III concludes the analysis by showing the usage of user interfaces with the considered container engines. In this case, only the Rancher graphical interface is supported by all the container engines. The rest of the tools, instead, have been specifically conceived for Docker.

V. CONCLUSION

Software Defined Systems allow to define, deploy, and use virtual resources, services, and applications across the network elements and clouds. To fulfill strict requirements (including, for instance, flexibility, isolation, and high performance), their enabling technologies include container engines, container orchestrators, and many other supporting tools. The paper deeply reviewed the state of the art on enabling technologies and provided a qualitative cross comparison, based on a specific set of Key Performance Indicators. The conducted study clearly demonstrated that Docker is emerging as the leading container engine. In fact, it offers a strong set of features and allows the usage of a large number of supporting tools. At the same time, it is remarked that Kuberentes appears as a very promising container orchestrator because it comes up with its own scheduler, load balancer, and service discovery features. Future research activity will complete the proposed qualitative investigation with a quantitative performance evaluation of container engines, container orchestrators, and their supporting tools.

VI. ACKNOWLEDGMENT

This work was partially funded by Italian MIUR PON projects Pico&Pro (ARS01_01061), AGREED (ARS01_00254), FURTHER (ARS01_01283), RAFAEL (ARS01_00305) and by Apulia Region (Italy) Research Project E-SHELF (OSW3NO1) and INTENTO (36A49H6).

REFERENCES

[1] D. Kreutz, F. M. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[2] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[3] Y. Jararweh, M. Al-Ayyoub, E. Benkhelifa, M. Vouk, A. Rindos *et al.*, "Sdiot: a software defined based internet of things framework," *Journal of Ambient Intelligence and Humanized Computing*, vol. 6, no. 4, pp. 453–461, 2015.

[4] Y. Alahmad, A. Agarwal, and T. Daradkeh, "High availability management for applications services in the cloud container-based platform," in *Proc. of IEEE IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, 2018, pp. 1–8.

[5] J. Matias, J. Garay, N. Toledo, J. Unzilla, and E. Jacob, "Toward an sdn-enabled nfv architecture," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 187–193, 2015.

TABLE II
CROSS COMPARISON AMONG CONTAINERS ORCHESTRATORS.

Name	Language	Developers	Community support	Licensing	S	LB	SD	Platform	Cloud	BM	Cluster
Kubernetes	Go	Google	Cloud Native Computing Foundation	OpenSource (Apache License 2.0)	✓	✓	✓	Linux, Mac, Windows	✓	✓	✓
Docker Compose	Python	Docker, Inc	Docker, Inc. (GitHub)	OpenSource (Apache License 2.0)	✓	✗	✗	Linux, Mac, Windows	✓	✓	✗
Docker Swarm	Python	Docker, Inc	Docker, Inc. (GitHub)	OpenSource (Apache License 2.0)	✓	✗	✗	Linux, Mac, Windows	✓	✗	✓
OpenStack	Python	NASA/ Rackspace	OpenStack Foundation/ Google	OpenSource (Apache License 2.0)	✓	✓	✓	Ubuntu	✓	✓	✓
Nomad	Go	Hashicorp	Hashicorp (GitHub)	Mozilla Public License 2.0	✓	✓	✗	Linux, Mac, Windows	✓	✓	✓
Apache Mesos	C++	UC Berkeley RAD Lab Students	Apache Software Foundation(Blog, GitHub)	OpenSource (Apache License 2.0)	✓	✗	✗	Linux, Mac, Windows	✓	✓	✓
Bistro	C++	Facebook, Inc.	Facebook, Inc.(Bistro Group, GitHub)	Closed source (BSD License)	✓	✗	✗	Ubuntu	✓	✗	✓
ECS	JS etc	Amazon.com, Inc.	Amazon.com, Inc.	Paid	✓	✓	✓	Linux, Windows	+(only AWS)	✗	✓
Cloud Foundry	Go, Ruby, Java	VMWare	Cloud Foundry Foundation Community and GitHub	OpenSource (Apache License 2.0)	✓	*	*	Linux, Mac, Windows	✓	✓	*
OpenShift	Go, AngularJS	RedHat Software	Red Hat (OpenShift personal Blog and Community)	OpenSource (Apache License 2.0)	+	+	+	Linux, Mac, Windows	✓	✗	✓

S = Scheduler; LB = Load Balancer; SD = Service Discovery; * through BOSH or Kubernetes; + through Kubernetes

TABLE III
JOINT USAGE OF LOAD BALANCING TOOLS AND CONTAINER ENGINES.

Tool	LXC	LXD	Docker	Rkt	Kata Containers
Bamboo	✗	✗	✓	✗	✗
Envoy	*	*	✓	*	*
HAProxy	✓	✓	✓	*	*
Kube-Proxy	*	*	*	*	*
MetalLB	*	*	*	*	*
NGINX	✓	✓	✓	✓	✓
Traefik	*	*	✓	*	*
Vamp-router	*	*	✓	✓	*
Vulcand	**	**	✓	**	**

* through Kubernetes; ** through Etcd

TABLE IV
JOINT USAGE OF SERVICE DISCOVERY TOOLS AND CONTAINER ENGINES.

Tool	LXC	LXD	Docker	Rkt	Kata Containers
ZooKeeper	Yes	Yes	Yes	*	**
Etcd	Yes	**	Yes	Yes	**
Consul	**	**	Yes	**	**
Mesos-DNS	***	***	***	***	✗
SkyDNS	****	****	****	****	*****
WeaveDNS	**	**	Yes	**	**
SmartStack	✗	✗	Yes	✗	✗
Eureka	✗	✗	Yes	✗	✗

through *Zetcd and Etcd3; **Kubernetes; ***Apache Mesos; ****Etcd; *****combination of Etcd and Kubernetes

TABLE V
JOINT USAGE OF USER INTERFACES AND CONTAINER ENGINES.

Tool	LXC	LXD	Docker	Rkt	Kata Containers
Rancher	✓	✓	✓	*	*
Clocker	✗	✗	✓	✗	✗
Tutum	✗	✗	**	✗	✗
Portainer	✗	✗	✓	✗	✗
Kitematic	✗	✗	✓	✗	✗
DockStation	✗	✗	✓	✗	✗
Panamax	✗	✗	✓	✗	✗
Docker UI	✗	✗	✓	✗	✗
Docker Compose-UI	✗	✗	✓	✗	✗
Shipyard	✗	✗	✓	✗	✗

*through Kubernetes; ** Docker acquired and integrated it

[6] V. Kaushik, A. Sharma, and R. Tomar, "Virtualizing network functions in software-defined networks," in *Innovations in Software-Defined Networking and Network Functions Virtualization*. IGI Global, 2018, pp. 26–51.

[7] P. Sharma, L. Chaufournier, P. Shenoy, and Y. Tay, "Containers and virtual machines at scale: A comparative study," in *Proc. of ACM International Middleware Conference*, 2016, p. 1.

[8] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, no. 3, pp. 81–84, 2014.

[9] X. Tang, F. Zhang, X. Li, S. U. Khan, and Z. Li, "Quantifying cloud elasticity with container-based autoscaling," *Future Generation Computer Systems*, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18307842

[10] S. V. Gogouvitis, H. Mueller, S. Premnadh, A. Seitz, and B. Bruegge, "Seamless computing in industrial systems using container orchestration," *Future Generation Computer Systems*, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17330236

[11] C. de Alfonso, A. Calatrava, and G. Moltó, "Container-based virtual elastic clusters," *Journal of Systems and Software*, vol. 127, pp. 1–11, 2017.

[12] J. Claassen, R. Koning, and P. Grosso, "Linux containers networking: Performance and scalability of kernel modules," in *Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2016, pp. 713–717.

[13] V. Marmol, R. Jnagal, and T. Hockin, "Networking in containers and container clusters," *Proceedings of netdev 0.1, February*, 2015.

[14] R. Cziva, S. Jouet, K. J. White, and D. P. Pezaros, "Container-based network function virtualization for software-defined networks," in *Proc. of IEEE Symposium on computers and communication (ISCC)*, 2015, pp. 415–420.

[15] Ł. Makowski and P. Grosso, "Evaluation of virtualization and traffic filtering methods for container networks," *Future Generation Computer Systems*, vol. 93, pp. 345–357, 2019.

[16] J. P. Martin, A. Kandasamy, and K. Chandrasekaran, "Exploring the support for high performance applications in the container runtime environment," *Human-centric Computing and Information Sciences*, vol. 8, no. 1, p. 1, 2018.

[17] T. Adufu, J. Choi, and Y. Kim, "Is container-based technology a winner for high performance scientific applications?" in *Proc. of IEEE Network Operations and Management Symposium (APNOMS),*, 2015, pp. 507–510.

[18] P. S. Kocher, *Microservices and Containers*. Addison-Wesley Professional, 2018.

[19] F. Paraiso, S. Challita, Y. Al-Dhuraibi, and P. Merle, "Model-driven management of docker containers," in *Proc. of IEEE International Conference on Cloud Computing (CLOUD)*, 2016, pp. 718–725.

[20] C. Kniep, "Containerization of high performance compute workloads using docker," *doc.qnib.org*, 2014.

[21] J.-S. Ma, D.-J. Kang, and H.-Y. Kim, "The lxc-lxd virtualization in arm64bit x-gene2 server," in *International Conference on Green and Human Information Technology*. Springer, 2018, pp. 168–175.

[22] C. Abdelmassih, "Container orchestration in security demanding environments at the swedish police authority," 2018.

[23] A. Mouat, *Orchestrating, clustering, and managing containers*. O'Reilly Media, Inc., 2016.

[24] A. Corradi, M. Fanelli, and L. Foschini, "Vm consolidation: A real case based on openstack cloud," *Future Generation Computer Systems*, vol. 32, pp. 118–127, 2014.

[25] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[26] K. Cacciatore, P. Czarkowski, S. Dake, J. Garbutt, B. Hemphill, J. Jainschigg, A. Moruga, A. Otto, C. Peters, and B. E. Whitaker, "Exploring opportunities: Containers and openstack," *OpenStack White Paper*, vol. 19, 2015.

[27] C. G. Kominos, N. Seyvet, and K. Vandikas, "Bare-metal, virtual machines and containers in openstack," in *Proc. of IEEE Conference on Innovations in Clouds, Internet and Networks (ICIN)*, 2017, pp. 36–43.

[28] H. Kang, M. Le, and S. Tao, "Container and microservice driven design for cloud infrastructure devops," in *Proc. of IEEE International Conference on Cloud Engineering (IC2E)*, 2016, pp. 202–211.

[29] M. Hausenblas, *Container Networking*. O'Reilly Media, Inc., 2018.

[30] A. Goder, A. Spiridonov, and Y. Wang, "Bistro: Scheduling data-parallel jobs against live production systems." in *USENIX Annual Technical Conference*, 2015, pp. 459–471.

[31] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: A survey," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 493–512, 2014.

[32] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[33] R. Enns, "Netconf configuration protocol," Tech. Rep., 2006.

[34] A. Bierman, M. Bjorklund, and K. Watsen, "Restconf protocol," Tech. Rep., 2017.