DE GRUYTER OPEN
RIGA TECHNICAL UNIVERSITY
Applied Computer Systems
ISSN 2255-8691 (online)
ISSN 2255-8683 (print)
May 2018, vol. 23, no. 1, pp. 21–27
doi: 10.2478/acss-2018-0003
https://www.degruyter.com/view/j/acss

Containers for Virtualization: An Overview

Vitor Goncalves da Silva[1*], Marite Kirikova[2], Gundars Alksnis[3]
[1–3] *Riga Technical University, Riga, Latvia*

***Abstract* – Virtualization has enabled the commoditization of cloud computing, as the hardware resources have become available to run different environments and share computing resources amongst different enterprises. Two technology types are commonly used in virtualization of hardware, namely, hypervisor and container. The present paper concerns only container technologies.**

A comprehensive overview of the container technologies for virtualization and the dynamics of their popularity have not been available yet. Without such an overview, the informed fast choice of technologies is hindered. To fill this knowledge gap, a systemic literature review was conducted to reveal the most popular container technologies and the trends in their research.

***Keywords* – Application virtualization, distributed computing, distributed processing.**

I. Introduction

Virtualization has enabled the commoditization of cloud computing, as the hardware resources have become available to run different environments and share computing resources amongst different enterprises. As the adoption of cloud computing has become a norm on a large-scale enterprise level [1], the improvement of resource sharing is a logical next step. Container technology improves the sharing of hardware resources by eliminating one layer of the infrastructure configuration and setup.

Running a virtual instance of a computer system in a layer abstracted from the physical hardware is the basic concept of virtualization. The applications which run on top of the virtualized machine operate without knowledge of any intermediary layer [2].

Two technologies are commonly used in virtualization of hardware, the hypervisor and the container. The hypervisor provides virtual machines (VM), which requires the provisioning of an operating system. Containers do not require the provisioning of an operating system; however, they can also run inside a VM. Both technologies support isolation and multi-tenancy [3]. Linux containers share close concepts with virtual machines. However, their operational flow presents some differences.

Container technologies, such as Docker [4], offer unprecedented agility in developing and running applications in a cloud environment, especially when combined with a microservice-style architecture [5]. It enables the design of multiple tenancy applications with less overhead when compared with the hypervisor. There are some security implications with the use of containers. Thus, the benefit does not come without the side effects [6]. Docker is just one of the possible approaches for containers in virtualization; it offers the complete solution for a container platform. To see the spectrum of the container technology ecosystem, a systematic literature review has been conducted to obtain an overview of container platform components and their alternatives.

II. Research Methodology

The present paper aims at identifying technologies associated with container technology, summarising its adoption in cloud-based environments, grouping and classifying key components of this technology and finally identifying research trends in this field. To achieve the goals mentioned previously, a systematic literature review has been conducted.

Related work. As a first step for conducting a systematic review, a search for related literature has been performed.

Research questions. The goal of this survey is to review existing research in order to assess the adoption of container technology and identify trends in this research field. To achieve the goals, the following research questions have to be answered:

- What technologies are associated with the implementation, control, and monitoring of containers?
- How is the research distributed in the field?
  1. Publishing entity.
  2. Chronologically.
  3. Geographically.

Eligibility criteria. A list of criteria is derived in order to include or exclude certain articles that are found on the related subject. The word “container” has homographs and therefore must be searched in combination with lexicons to minimise unrelated papers appearing in the search results.

Mandatory criteria for including the article are:

- The study is published in English and available in Scopus [7].
- The study contains comprehensive conclusions of performed review or application of container technology.

One of the following additional criteria must be met to include the article in the research scope:

- The study considers the subject of Containers in the context of software systems.
- The study considers the subject of Microservices Architecture and its underlying infrastructure.
- The study considers the subject of DevOps and the adoption of container technology.

Matching one of the following criteria results in the exclusion of article from the scope of the survey:

- Studies were not peer-reviewed or published.

* Corresponding author’s e-mail: vitor.goncalves-da-silva@edu.rtu.lv

- Studies that did not answer the research question at least partially.
- Studies that did not derive potential future improvements from the research conducted.

Look-up concepts. Based on the questions that have to be answered during this survey and considering limitations proposed by the exclusion criteria, at the beginning of systematic review the domain related concepts and notions have been defined. These concepts have been used in the search:

- ALL ("containers" AND "cloud") AND (LIMIT-TO (SUBJAREA, "COMP"));
- TITLE-ABS-KEY ("microservices" AND ("infrastructure" OR "cluster")).

Look-up strategy. Systematic review guidelines suggest that searching in the title does not always result in all relevant publications. Therefore, the abstract or full-text of articles should also be considered. The search was restricted to the title and abstract of the studies.

TABLE I

SELECTED ARTICLES FOR REVIEW ON THE SCOPE OF THIS SURVEY

Citation	Year	Title
[8]	2016	Network Function Virtualization: State-of-the-Art and Research Challenges
[9]	2015	Large-Scale Cluster Management at Google with Borg
[3]	et al., 2014	Containers and Cloud: From LXC to Docker to Kubernetes
[10]	et al., 2015	Hypervisors vs. Lightweight Virtualization: A Performance Comparison
[11]	et al., 2015	Containerization and the PaaS Cloud
[12]	et al., 2014	Skyport – Container-Based Execution Environment Management for Multi-Cloud Scientific Workflows
[13]	et al., 2015	An Architecture for Self-Managing Microservices
[14]	et al., 2014	The Research and Implementation of Cloud Computing Platform Based on Docker
[5]	et al., 2016	Container and Microservice Driven Design for Cloud Infrastructure DevOps
[15]	et al., 2015	Distributed Systems of Microservices Using Docker and Serfnode
[16]	et al., 2015	Leveraging Linux Containers to Achieve High Availability for Cloud Services
[17]	et al., 2015	Cloud Services in the Guifi.net Community Network
[18]	et al., 2015	ROAR: A QoS-Oriented Modelling Framework For Automated Cloud Resource Allocation And Optimization
[19]	et al., 2015	Integrating Containers into Workflows: A Case Study Using Makeflow, Work Queue, and Docker
[20]	et al., 2015	A REST Service Framework for Fine-Grained Resource Management in Container- Based Cloud
[21]	et al., 2015	Docker Containers Across Multiple Clouds and Data Centres

Although applying the restrictions previously mentioned the number of articles available to conduct this survey was present in a large number, to further narrow the subject of study the authors used the number of citations as a parameter to sort and prioritize articles subject to review.

The resulting list contains 16 articles that qualify according to the search criteria described previously (see Table I).

III. CONTAINER TECHNOLOGY STACK

Docker container platform has solutions for most layers of the container technology stack. The layers covered are container engine, scheduling, orchestration, image management and configuration management. Docker's market share position and breadth of solutions make its name ubiquitous with container technology [32].

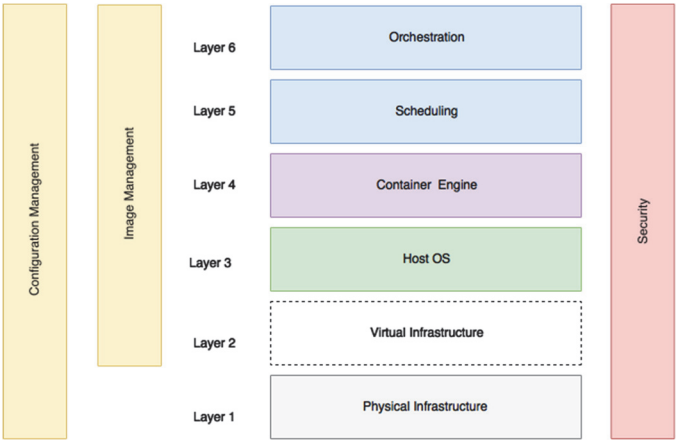

Fig. 1. Reference architecture of cloud containers.

Containers are an abstraction at the application layer that packages code and dependencies. Containers provide OS-level virtualization by leveraging kernel features to isolate processes and define system usage limits for resources such as CPU, memory, disk I/O and network. Multiple containers can run on the same machine and share the operating system OS kernel with other containers, each running as isolated processes in user space. Containers take up less disk and memory space than VMs, containers are also considerably smaller in terms of storage space required, with a smaller energy consumption footprint [27], and start almost instantly. The container has all the dependencies it needs to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows-based apps, the containerized software will always run the same, regardless of the environment [5], [11]. Those who want to deploy applications with the least infrastructure will choose a simple container-to-OS approach. This is why container-based cloud vendors can claim improved performance when compared to hypervisor-based clouds [9], [10].

The layering composition illustrated in Fig. 1. depicts the underlying components present in cloud container solutions. In this section those layers and associated technologies, identified by the conducted research as the most prominent, are described further and classified accordingly.

KVM [25] is situated in the Virtual Infrastructure layer (layer 2); this layer is optional when considering containers. KVM is a virtualization infrastructure for the Linux kernel that turns it into a hypervisor. Containers can run both, directly on host OS and on the guest OS that runs on virtualization technology, such as KVM.

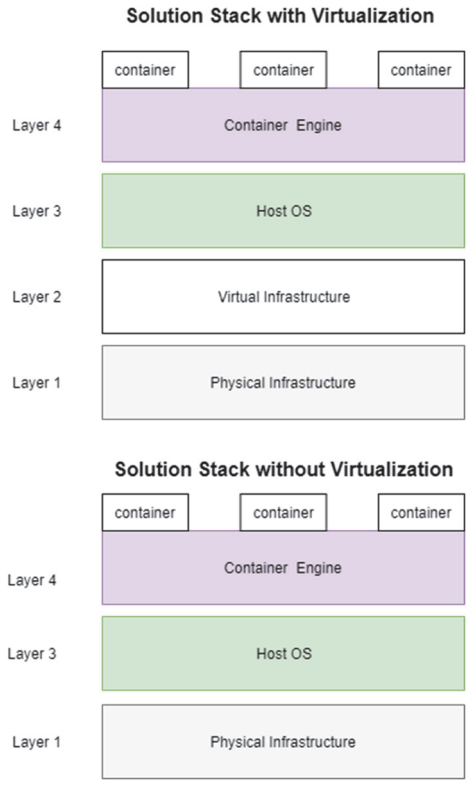

Fig. 2. Virtualization as an optional layer for the container solution stack.

The technologies identified in the Container Engine layer (layer 4) as illustrated in Fig. 1, are Linux containers (LXC), Docker (container and Docker engine), Rocket (rkt) and CoreOS.

LXC [24] are similar to VMs in which they have a fully functional OS; data can be saved in a container or outside. LXC are filesystem neutral and by default store data in a stateful manner. It is possible to build loosely coupled or composite stacks.

Docker [4] is a solution that brings an alternative philosophy in comparison to LXC. In the Docker containers are designed to support a single application as opposed to composite stacks. Containers are made up of read-only layers via AUFS/Device mapper, data written by the applications into the container is handled by the Copy-on-write resource management technique. Container instances are ephemeral, persistent data are stored in blind mounts to host or data volume containers [34]. Recent releases of Docker have introduced the Volumes feature, it presents the ability to store stateful data.

Rocket (rkt) is a container engine technology developed by CoreOS as a lightweight and secure alternative to Docker [26].

Earlier versions of Docker (<1.11) presented security issues by downloading container images, launching container processes and exposing a remote API, through a centralised process running as root. Although such a design flaw has been mitigated by higher versions, there is a fundamental difference in both container engine process models [33].

Different container engines presented the risk of lack of interoperability; this risk was removed when the Open Container Initiative (OCI) was established by Docker and other leaders in the container industry [35]. Two specifications guarantee interoperability between different container engines, the Runtime Specification (runtime-spec) and the Image Specification (image-spec). The Runtime Specification outlines how to run a "filesystem bundle" that is unpacked on disk. At a high level, an OCI implementation would download an OCI Image then unpack that image into an OCI Runtime filesystem bundle. At this point, the OCI Runtime Bundle would be run by an OCI Runtime.

CoreOS Container Linux is designed to be managed and run at massive scale, with minimal operational overhead. Traditional Linux operating system distributions are built in accordance with their proposed usage.

Convenience is one of the factors to consider when building Linux distributions; thus, the inclusion of preinstalled packages is possible for various applications. Containers, in contrast, are designed for lightweight virtualization to run many identical machines as possible with the least amount of overhead in terms of memory, disk and CPU. To cater for the low overhead requirements that provide faster spin-up time, vendors have developed container-optimized builds. The result is minimalistic distributions containing the minimum requirements for containers to run.

Container Linux runs on nearly any platform whether physical, virtual, or private/public cloud.

Scheduling (layer 5) and Orchestration (layer 6) as illustrated in Fig. 1 are comprised by the following technologies: Kubernetes, Apache Mesos, Swarm, Marathon, YARN and Omega yield by the research results.

The orchestration and management complexity problems were solved by introducing orchestration and cluster management tools such as Kubernetes [22] and Apache Meso [23].

Kubernetes is an open source system for managing clusters of containers [3], [22]. It operates on the orchestration layer. It provides tools for deploying applications, scaling that application as needed, managing changes to existing containerized applications, and also plays a role in the optimization of the use of the underlying hardware beneath the containers. Kubernetes is designed to be extensible and fault-tolerant by allowing application components to restart and move across systems as needed.

Google has over a decade of experience running containerized workloads in production. Borg is the initial project responsible for Google's internal container-oriented cluster-management system [9]. Kubernetes was born by taking the best ideas from Borg with additional improvements [36].

Google has donated Kubernetes to the Linux Foundation to form the Cloud Native Computing Foundation. The community support has defined it as powerful and more robust than other technologies that operate in the orchestration layer. In addition to Google, other major players in container technology have announced support for Kubernetes: AWS [37], Azure [38] and Docker [39]. Docker Swarm remains available for container orchestration; it can co-exist with Kubernetes on edge-case scenarios [40].

Apache Mesos is a cluster manager that provides efficient resource isolation and sharing across distributed applications or frameworks [23]. Apache Mesos is open source software originally developed at the University of California at Berkeley. It is best suited to deploy and manage applications in large-scale clustered environments in an efficient manner. Apache Mesos relies on containerizers, and Marathon is a container orchestration platform for Mesos and DC/OS [29], [12]. A containerizer is a Mesos agent component responsible for launching containers, within which you can run a service. Running services in containers offer a number of benefits, including the ability to isolate tasks from one another and control task resources programmatically. It is possible to use a Docker containerizer, which delegates container management to the Docker engine. In Mesos, resources are offered to application-level schedulers. This allows for custom, workload-specific scheduling policies.

Omega operates on the scheduler layer and approaches scheduling in a different manner [31]. It positions itself as a shared state scheduler and grants full access to the entire cluster resources by removing the central resource allocator. The state of the cluster is shared among all the schedulers. Supporting independent scheduler implementations and exposing the entire allocation state of the schedulers, the architecture can scale up to a large number of schedulers and works with different workloads with their own scheduling policies. Opposed to this approach, Docker Swarm is a monolithic scheduler with a single, centralised scheduling algorithm for all jobs. This type of schedulers is not suitable for running heterogeneous modern workloads and other long-running jobs.

Apache Hadoop YARN [30] is used in conjunction with Kubernetes. YARN operates on the scheduling layer; it makes use of HDFS to enable common resource management across data centres and PaaS workloads in a seamless fashion. In this manner, hybrid cloud solutions are achieved with greater ease [13]. YARN has a two-level scheduler; it separates concerns of resource allocation and task placement.

Containers have the ability to be managed collectively, thus forming a cluster. The complexity of implementing multiple microservices that rely on several clusters with interlinked dependencies creates a complex problem for the provision and management of container clusters [12], [13].

IV. Results

A. Technologies Applied in Container Technology

The qualitative analysis of the selected articles shows that all articles mention Docker; and it is the most prominent technology for wrapping Linux containers and providing a minimal API for interaction. A full enumeration of technologies used is presented in the Table II below, depicting its reference and research article.

TABLE II

Most Prominent Technologies

Technology	Articles
Docker [4]	[3], [5], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21]
LXC [24]	[3], [5], [10], [11], [12], [14], [15], [16], [17]
KVM [25]	[5], [10], [12], [14], [16]
CoreOS [26]	[15], [19]
Rocket [33]	[5], [19]
Kubernetes [22]	[3], [5], [9], [11], [12], [13], [15], [19], [20], [21]
Apache Mesos [23]	[11], [12], [13], [15], [20], [21]
Swarm [28]	[5]
Marathon [29]	[12]
YARN [30]	[13]
Omega [31]	[13]

B. Trends in the Container Research Field

The selected articles have also been subject to a quantitative analysis which aims at showing which journals have the most presence (Publishing Entity), the chronological distribution of the articles and finally the identification of global research clusters.

Publishing entity. According to this criterion, the articles are grouped by publisher and listed in the Table III below.

TABLE III

Grouping by Publishing Entity

Publishing Entity	Articles	Total
ACM	[8], [13], [19]	3
IEEE	[3], [5], [9], [10], [11], [12], [14], [15], [16], [20], [21]	11
Science Direct	[17], [18]	2

Chronological publication. According to this criterion, the articles are grouped by year of publication and mapped on the chart and shown in the Fig. 3 below.

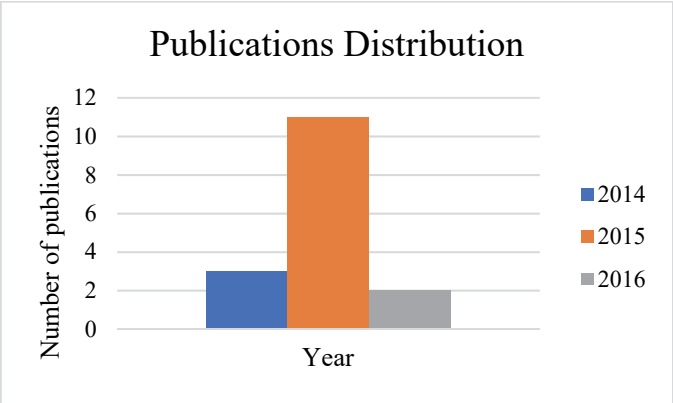

Fig. 3. Grouping by year of publication.

Geographical publication. According to this criterion, the articles are grouped by geographical publication and shown in the Fig. 4 below. The aim is to identify clusters of research. The authors of the researched articles were classified by their geographical location of research. In some articles, there is more than one author; thus, the sum is higher than the total number of research articles.

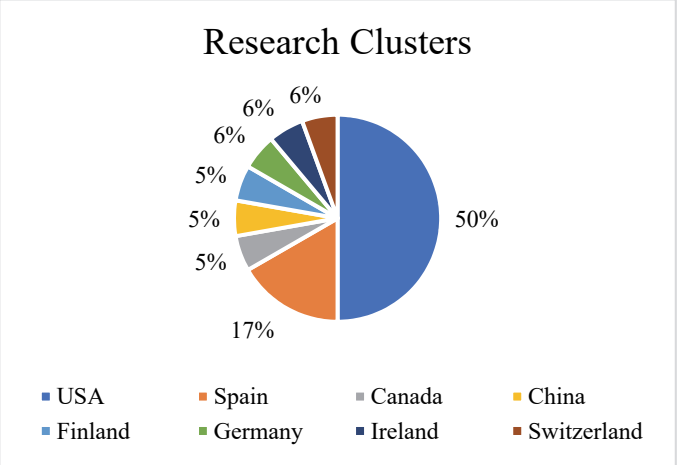

Fig. 4. Research cluster, geographical grouping.

To observe the trends for the technologies identified, the pool of articles was extended by maintaining the eligibility criteria from the research methodology. The look-up concepts used to retrieve results per technology are displayed in Table IV. For the look-up strategy previously defined, all restrictions were lifted to capture the full spectrum from 2014 to 2018.

TABLE IV

TECHNOLOGY LOOK-UP CONCEPTS

Technology	Look-up concepts
Docker	ALL (“docker” AND “container”) AND (LIMIT-TO (SUBJAREA , “COMP”)) AND (LIMIT-TO (DOCTYPE, “cp”) OR LIMIT-TO (DOCTYPE , “ar”)) AND (LIMIT-TO (PUBYEAR, 2018) OR LIMIT-TO (PUBYEAR, 2017) OR LIMIT-TO (PUBYEAR, 2016) OR LIMIT-TO (PUBYEAR, 2015) OR LIMIT-TO (PUBYEAR, 2014))
LXC	ALL (“LXC” AND “container”) AND (LIMIT-TO (SUBJAREA, “COMP”)) AND (LIMIT-TO (DOCTYPE, “cp”) OR LIMIT-TO (DOCTYPE, “ar”)) AND (LIMIT-TO (PUBYEAR, 2018) OR LIMIT-TO (PUBYEAR, 2017) OR LIMIT-TO (PUBYEAR, 2016) OR LIMIT-TO (PUBYEAR, 2015) OR LIMIT-TO (PUBYEAR, 2014))
KVM	ALL (“KVM” AND “container”) AND (LIMIT-TO (SUBJAREA, “COMP”)) AND (LIMIT-TO (DOCTYPE, “cp”) OR LIMIT-TO (DOCTYPE, “ar”)) AND (LIMIT-TO (PUBYEAR, 2018) OR LIMIT-TO (PUBYEAR, 2017) OR LIMIT-TO (PUBYEAR, 2016) OR LIMIT-TO (PUBYEAR, 2015) OR LIMIT-TO (PUBYEAR, 2014))
CoreOS	ALL (“coreOS” AND “container”) AND (LIMIT-TO (SUBJAREA, “COMP”)) AND (LIMIT-TO (PUBYEAR, 2018) OR LIMIT-TO (PUBYEAR, 2017) OR LIMIT-TO (PUBYEAR, 2016) OR LIMIT-TO (PUBYEAR, 2015) OR LIMIT-TO (PUBYEAR, 2014)) AND (LIMIT-TO (DOCTYPE, “cp”) OR LIMIT-TO (DOCTYPE, “ar”))
Rocket	ALL ((“rocket” OR “rkt”) AND “container”) AND (LIMIT-TO (SUBJAREA, “COMP”)) AND (LIMIT-TO (PUBYEAR, 2018) OR LIMIT-TO (PUBYEAR, 2017) OR LIMIT-TO (PUBYEAR, 2016) OR LIMIT-TO (PUBYEAR, 2015) OR LIMIT-TO (PUBYEAR, 2014)) AND (LIMIT-TO (DOCTYPE, “cp”) OR LIMIT-TO (DOCTYPE, “ar”))
Kubernetes	ALL (“kubernetes” AND “container”) AND (LIMIT-TO (SUBJAREA, “COMP”)) AND (LIMIT-TO (DOCTYPE, “cp”) OR LIMIT-TO (DOCTYPE, “ar”)) AND (LIMIT-TO (PUBYEAR, 2018) OR LIMIT-TO (PUBYEAR, 2017) OR LIMIT-TO (PUBYEAR, 2016) OR LIMIT-TO (PUBYEAR, 2015) OR LIMIT-TO (PUBYEAR, 2014))
Apache Mesos	ALL (“mesos” AND “container”) AND (LIMIT-TO (SUBJAREA, “COMP”)) AND (LIMIT-TO (DOCTYPE, “cp”) OR LIMIT-TO (DOCTYPE, “ar”)) AND (LIMIT-TO (PUBYEAR, 2018) OR LIMIT-TO (PUBYEAR, 2017) OR LIMIT-TO (PUBYEAR, 2016) OR LIMIT-TO (PUBYEAR, 2015) OR LIMIT-TO (PUBYEAR, 2014))
Swarm	ALL (“docker” AND “swarm”) AND (LIMIT-TO (SUBJAREA, “COMP”)) AND (LIMIT-TO (DOCTYPE, “cp”) OR LIMIT-TO (DOCTYPE, “ar”)) AND (LIMIT-TO (PUBYEAR, 2018) OR LIMIT-TO (PUBYEAR, 2017) OR LIMIT-TO (PUBYEAR, 2016) OR LIMIT-TO (PUBYEAR, 2015) OR LIMIT-TO (PUBYEAR, 2014))
Marathon	ALL (“marathon” AND “mesosphere” AND “container”) AND (LIMIT-TO (SUBJAREA, “COMP”)) AND (LIMIT-TO (DOCTYPE, “cp”) OR LIMIT-TO (DOCTYPE, “ar”)) AND (LIMIT-TO (PUBYEAR, 2018) OR LIMIT-TO (PUBYEAR, 2017) OR LIMIT-TO (PUBYEAR, 2016) OR LIMIT-TO (PUBYEAR, 2015) OR LIMIT-TO (PUBYEAR, 2014))
YARN	ALL (“YARN” AND “container”) AND (LIMIT-TO (SUBJAREA, “COMP”)) AND (LIMIT-TO (DOCTYPE, “cp”) OR LIMIT-TO (DOCTYPE, “ar”)) AND (LIMIT-TO (PUBYEAR, 2018) OR LIMIT-TO (PUBYEAR, 2017) OR LIMIT-TO (PUBYEAR, 2016) OR LIMIT-TO (PUBYEAR, 2015) OR LIMIT-TO (PUBYEAR, 2014))
Omega	ALL (“omega” AND “scheduler” AND “container”) AND (LIMIT-TO (SUBJAREA, “COMP”)) AND (LIMIT-TO (DOCTYPE, “cp”) OR LIMIT-TO (DOCTYPE, “ar”)) AND (LIMIT-TO (PUBYEAR, 2018) OR LIMIT-TO (PUBYEAR, 2017) OR LIMIT-TO (PUBYEAR, 2016) OR LIMIT-TO (PUBYEAR, 2015) OR LIMIT-TO (PUBYEAR, 2014))

The trends observed were grouped into container technology core and container technology scheduling and orchestration.

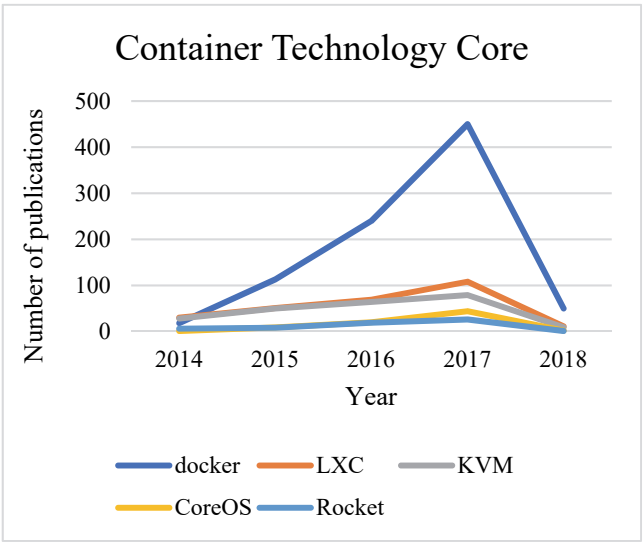

Fig. 5. Container Technology Core – Layers 2, 3, and 4.

Docker is the preferred choice for the container engine layer due to its ease of use and popularity provided by its container platform. As it reaches increase maturity, the adoption has been increasing.

Linux containers (LXC) which is a building block of the Docker engine has seen a modest increase in interest.

CoreOS and Rocket (rkt) offer an alternative architecture and to the container engine, although its adoption has been modest compared with Docker.

KVM virtualization has maintained a steady interest; there is promise of revival of virtualization with unikernels [41].

The Open Container Initiative has paved way for the standardisation of container engines.

Alternatives to the core layers still have security improvements to make when compared with hypervisor based virtualization. In terms of the security, Rocket is ahead of Docker.

There is a drop in 2018; the numbers are expected to change until the end of the current year. Nevertheless, they provide insight for the period between January and March.

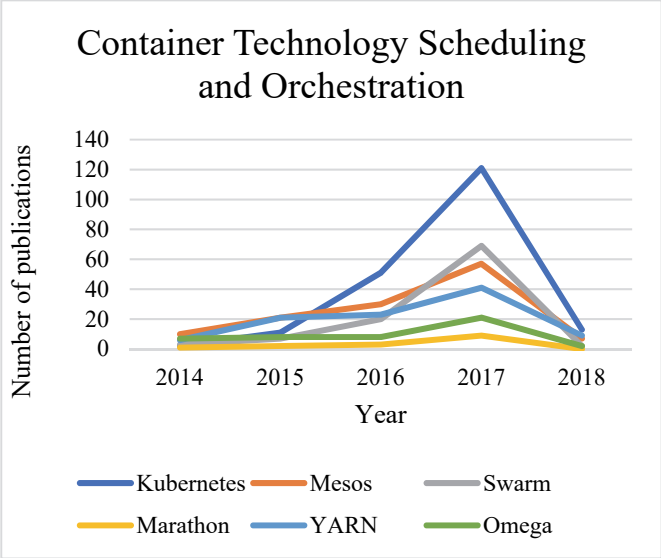

Fig. 6. Container technology core – Layers 5 and 6.

Benefiting from many years of usage within production environment, Borg [9] was open-sourced giving life to Kubernetes. Kubernetes dominates the adoption on the scheduling and orchestration layers; it has monolithic scheduling properties.

Docker Swarm is an offering from the Docker container platform. It is no match for Kubernetes in terms of ease of use and functionality.

Other technologies have more modest interest and adoption; they are a better fit for more complex systems when two-level and share-state scheduling is required.

V. Conclusion

Docker is the most prominent technology in cloud containers and its use is widely adopted. This area of research is maturing and as expected, several production ready study cases have become available since 2014.

With the Open Container Initiative [35] a standard was established for container engines to achieve interoperability.

Kubernetes has dominance on the layers of scheduling and orchestration; it is offered by the major cloud providers including Docker.

There are still many research opportunities, especially in the security area and in the orchestration, management, and control of distributed clusters [15], [19].

The adoption of containers is widespread; the next logical step for cloud service providers is to use container technology and abstract its implementation by providing serverless computing. The authors will carry out further research on this promising field.

References

[1] A. Khajeh-Hosseini, D. Greenwood, and I. Sommerville, "Cloud Migration: A Case Study of Migrating an Enterprise It System to IaaS," in *2010 IEEE 3rd International Conference on Cloud Computing*, pp. 450–457. https://doi.org/10.1109/cloud.2010.37

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *SIGOPS Oper. Syst. Rev.*, 2003, vol. 37, pp. 164–177. https://doi.org/10.1145/1165389.945462

[3] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, issue 3, pp. 81–84, 2014. https://doi.org/10.1109/mcc.2014.51

[4] *Build, Ship, and Run Any App, Anywhere*. [Online]. Available: https://www.docker.com/what-docker [Accessed: 10 Dec. 2017].

[5] H. Kang, M. Le, and S. Tao, "Container and Microservice Driven Design for Cloud Infrastructure DevOps," in *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 202–211. https://doi.org/10.1109/ic2e.2016.26

[6] L. Rodero-Merino, L. M. Vaquero, E. Caron, A. Muresan, and F. Desprez, "Building Safe PaaS Clouds: A Survey on Security in Multitenant Software Platforms," *Computers Security*, vol. 31, issue 1, pp. 96–108, 2012. https://doi.org/10.1016/j.cose.2011.10.006

[7] *Scopus: The Largest Abstract and Citation Database of Peer-Reviewed Literature: Scientific Journals, Books and Conference Proceedings*. [Online]. Available: https://www.elsevier.com/solutions/scopus [Accessed: October. 18, 2017].

[8] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys and Tutorials*, vol. 18, issue 1, pp. 236–262, 2016. https://doi.org/10.1109/comst.2015.2477041

[9] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-Scale Cluster Management at Google With Borg," in *Proc. Tenth European Conference on Computer Systems, EuroSys '15*, ACM, New York, NY, USA, 2015. https://doi.org/10.1145/2741948.2741964

[10] R. Morabito, J. Kjallman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: A Performance Comparison," in *2015 IEEE International Conference on Cloud Engineering*, pp. 386–393. https://doi.org/10.1109/ic2e.2015.74

[11] C. Pahl, "Containerization and the PaaS Cloud," *IEEE Cloud Computing*, vol. 2, issue 3, pp. 24–31, 2015. https://doi.org/10.1109/mcc.2015.51

[12] W. Gerlach, W. Tang, K. Keegan, T. Harrison, A. Wilke, J. Bischof, M. DSouza, S. Devoid, D. Murphy-Olson, N. Desai, and F. Meyer, "Skyport – Container-Based Execution Environment Management for Multi-Cloud Scientific Workflows," in *2014 5th International Workshop on Data-Intensive Computing in the Clouds*, pp. 25–32. https://doi.org/10.1109/datacloud.2014.6

[13] G. Toffetti, S. Brunner, M. Blöchlinger, F. Dudouet, and A. Edmonds, "An Architecture for Self-Managing Microservices," in *Proc. 1st International Workshop on Automated Incident Management in Cloud – AIMC '15*, ACM, New York, NY, USA, 2015, pp. 19–24. https://doi.org/10.1145/2747470.2747474

[14] D. Liu and L. Zhao, "The Research and Implementation of Cloud Computing Platform Based on Docker," in *2014 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing (ICCWAMTIP)*, pp. 475–478. https://doi.org/10.1109/iccwamtip.2014.7073453

[15] J. Stubbs, W. Moreira and R. Dooley, "Distributed Systems of Microservices Using Docker and Serfnode," in *2015 7th International Workshop on Science Gateways*, pp. 34–39. https://doi.org/10.1109/iwsg.2015.16

[16] W. Li, A. Kanso, and A. Gherbi, "Leveraging Linux Containers to Achieve High Availability for Cloud Services," in *2015 IEEE International Conference on Cloud Engineering*, pp. 76–83. https://doi.org/10.1109/ic2e.2015.17

[17] M. Selimi, A. M. Khan, E. Dimogerontakis, F. Freitag and R. P. Centelles, "Cloud Services in the Guifi.net Community Network," *Computer Networks*, vol. 93, pp. 373–388, 2015. https://doi.org/10.1016/j.comnet.2015.09.007

[18] Y. Sun, J. White, S. Eade, and D. C. Schmidt, "ROAR: A QoS-Oriented Modeling Framework for Automated Cloud Resource Allocation and Optimization," *Journal of Systems and Software*, vol. 116, pp. 146–161, 2016. https://doi.org/10.1016/j.jss.2015.08.006

[19] C. Zheng and D. Thain, "Integrating Containers Into Workflows: A Case Study Using Makeflow, Work Queue, and Docker," in *Proc. 8th International Workshop on Virtualization Technologies in Distributed Computing - VTDC '15*, ACM, New York, NY, USA, 2015, pp. 31–38. https://doi.org/10.1145/2755979.2755984

[20] L. Li, T. Tang and W. Chou, "A REST Service Framework for Fine-Grained Resource Management in Container-Based Cloud," in *2015 IEEE 8th International Conference on Cloud Computing*, pp. 645–652. https://doi.org/10.1109/cloud.2015.91

[21] M. Abdelbaky, J. Diaz-Montes, M. Parashar, M. Unuvar, and M. Steinder, "Docker Containers Across Multiple Clouds and Data Centers," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, pp. 368–371.

[22] *Kubernetes: Open-Source System for Automating Deployment, Scaling, and Management of Containerized Applications*. [Online]. Available: https://kubernetes.io, [Accessed: 20 Oct. 2017].

[23] *Apache Mesos: A Distributed Systems Kernel*. [Online]. Available: https://mesos.apache.org [Accessed: 20 Oct. 2017].

[24] *What's LXC?*. [Online]. Available: https://linuxcontainers.org/lxc/introduction/ [Accessed: Oct. 20, 2017].

[25] A. Shah, "Ten Years of KVM," 2016. [Online]. Available: https://lwn.net/Articles/705160/ [Accessed: 19 Mar. 2018].

[26] *CoreOS powers the world's container infrastructure*. [Online]. Available: https://coreos.com/why [Accessed: 19 Mar. 2018].

[27] R. Morabito, "Power Consumption of Virtualization Technologies: An Empirical Investigation," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, Limassol, 2015, pp. 522–527.

[28] *Swarm Mode Overview*. [Online] Available: https://docs.docker.com/engine/swarm [Accessed: 19 Mar. 2018].

[29] *Marathon: A container orchestration platform for Mesos and DC/OS*. [Online]. Available: https://mesosphere.github.io/marathon/ [Accessed: 19 Mar. 2018].

[30] *Apache Hadoop YARN*. [Online]. Available: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html [Accessed: 20 Oct. 2017].

[31] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, Scalable Schedulers for Large Compute Clusters," in *Proc. 8th ACM European Conference on Computer Systems (EuroSys '13)*, ACM, New York, NY, USA, 2013, pp. 351–364. https://doi.org/10.1145/2465351.2465386

[32] *State of the Cloud Report*. [Online]. Available: https://assets.rightscale.com/uploads/pdfs/RightScale-2017-State-of-the-Cloud-Report.pdf [Accessed: 18 Mar. 2018].

[33] *rkt vs other projects*. [Online]. Available: https://coreos.com/rkt/docs/latest/rkt-vs-other-projects.html [Accessed: 19 Mar. 2018].

[34] A. Fulay "Containers Deep Dive – LXC vs DockerG," Jan. 2017. [Online]. Available: https://robinsystems.com/blog/containers-deep-dive-lxc-vs-docker-comparison/ [Accessed: 18 Mar. 2018].

[35] *Open Container Initiative*. [Online]. Available: https://www.opencontainers.org/ [Accessed: 19 Mar. 2018].

[36] *Borg: The Predecessor to Kubernetes*. [Online]. Available: http://blog.kubernetes.io/2015/04/borg-predecessor-to-kubernetes.html [Accessed: 19 Mar. 2018].

[37] *Amazon AKS*. [Online]. Available: https://aws.amazon.com/eks/ [Accessed: 19 Mar. 2018].

[38] *Azure Container Service (AKS)*. [Online]. Available: https://azure.microsoft.com/en-us/services/container-service/ [Accessed: 19 Mar. 2018].

[39] *Docker Support for Kubernetes*. [Online]. Available: https://www.docker.com/kubernetes [Accessed: 19 Mar. 2018].

[40] B. Stewart, "Why Kubernetes vs. Swarm is the Wrong Question," 2017. [Online]. Available: https://www.wintellect.com/kubernetes-vs-swarm-wrong-question/ [Accessed: 19 Mar. 2018].

[41] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, "Consolidate IoT Edge Computing with Lightweight Virtualization," *IEEE Network*, vol. 32, no. 1, pp. 102–111, Jan. 2018. https://doi.org/10.1109/mnet.2018.1700175

Vitor Goncalves da Silva is a second-year Master student majoring in Business Informatics, Riga Technical University, Latvia (2016–2018). At present, he is a Sr. Software Engineer at Intexsys. Fields of interests include cloud computing, software design and systems development.
E-mail: vitor.goncalves-da-silva@edu.rtu.lv

Marite Kirikova, *Dr. sc. ing.*, is a Professor in Information Systems Design at the Department of Artificial Intelligence and Systems Engineering, Faculty of Computer Science and Information Technology, Riga Technical University, Latvia. She has more than 150 publications on the topics of requirements engineering, business process modelling, knowledge management, and systems development. She has done fieldwork at Stockholm University and Royal Institute of Technology, Sweden, Copenhagen University, Denmark, and Boise University, USA. In her research, currently she focuses on continuous information systems engineering in the context of agile and viable systems approaches.
E-mail: marite.kirikova@gmail.com

Gundars Alksnis received *Dr. sc. ing.* in information technologies (system analysis, modelling and design) from Riga Technical University, Latvia, in 2008. He is an Assistant Professor and Researcher at the Department of Applied Computer Science, Riga Technical University. His research interests include modelling in the context of model-driven software development and IT security.
E-mail: gundars.alksnis@rtu.lv