File system performance comparison of native operating system and Docker container-based virtualization

Borislav Đorđević, *Member, IEEE*, Darko Gojak, Nikola Davidović and Valentina Timčenko, *Member, IEEE*

***Abstract* - This paper aims to examine and compare the file system capabilities of container virtualization and the native host. Different virtualization categories are mentioned with a focus on OS level types. We have described the importance of container virtualization and its contribution to virtualization popularization. Also, the paper contains a detailed description of the Docker container-based virtualization, its mode of operation, as well as the advantages and disadvantages it possesses. Since the main purpose of this work is to measure the host and Docker file system throughput, one of the best open-source benchmarks is chosen and presented - FileBench, through which all tests were performed. With a practical example, we have shown the file system performance comparisons considering Docker containers and host physical machine.**

Keywords - Docker; containers; virtualization; benchmark; FileBench; file system; performance; comparison.

I. INTRODUCTION

There is rapid development in the IT industry, while hardware and software are changing daily. Hardware development is accompanied by software solutions that aim to make the most efficient use of performance. We strive for solutions that will meet today's standards, asking ourselves what the best use is and how to optimize the available resources so that the requirements and user needs are met.

Some of the most important characteristics in hardware manufacturing are the development costs and time [1]. The above brings us to one of the indispensable topics of today in the IT world - virtualization.

The question is whether virtualization is a better solution and how cost-effective it is, whether it is possible to achieve the desired results with virtualization, and what the limitations are.

There are several varieties of virtualization types, and it can be said for all of these varieties to be usable, with some being more simplified, that is, less decomposed than others. One of the variations is that virtualization can be divided into eight types: hardware virtualization, network virtualization, storage virtualization, memory virtualization, software virtualization, OS level virtualization, data virtualization, and desktop virtualization.

The type of virtualization covered in this paper is "OS level virtualization", whose instances are sometimes called containers. One of the most common associations when mentioning container instances is the well-known Docker [2]. In this paper, the Docker container's file system is examined and compared with the host file system performance.

As the popularity of container virtualization has been growing over time, so have questions about the performance of this type of virtualization. It is hard to talk about container virtualization without mentioning the increasingly prevalent Docker. The ease of installation and use, as well as simplicity of containers management, made Docker a good candidate for file system testing. Another benefit of using it is that Docker containers are lightweight, time savers (it takes less than a minute to build one instance) and besides that, they are consuming a small amount of disk space, so those instances will not affect the host significantly.

Thus, in this paper, the response of the file system of the native operating system and Docker container-based virtualization was researched, and then a comparison of the obtained results was made.

II. RELATED WORK, OBJECTIVE, AND MOTIVATION

As hardware is developing fast today, in terms of storage size, its response speed, as well as processing power, there is an inevitable question about the efficient use of physical machines, which are in most cases underused, or their full potential is not reached [3]. In this regard, scientific research deals with the consideration of further efficiency enhancements possibilities and the mentioned issues.

There is a growing debate about whether virtual solutions are always better and whether they can be expected to largely compete with physical machines [4], [5]. As a big part of the hardware resources in many cases remain unused, there is a lot of room left for the possibility of implementing virtual instances and consideration of the most optimal use.

As the main goal of this paper is to compare the performance of the file systems, with equal settings and the same conditions of the benchmark used for host and Docker

Borislav Đorđević – Institute Mihailo Pupin, Volgina 15, 11000 Belgrade, Serbia, (borislav.djordjevic@pupin.rs)

Darko Gojak – VISER, School of Electrical and Computer Engineering of Applied Studies, Belgrade, Serbia (darkogojak@gmail.com)

Nikola Davidović – University of East Sarajevo, Faculty of Electrical Engineering, Vuka Karadzica 30, 71123 East Sarajevo, RS, BiH, (nikola.davidovic@etf.unssa.rs.ba)

Valentina Timčenko - Institute Mihailo Pupin, School of Electrical Engineering, Belgrade, Serbia (valentina.timcenko@pupin.rs)

containers, we resorted to the method of comparative analysis through FileBench workloads, where four were selected, namely: fileserver, webserver, varmail and randomfileaccess. In our opinion, these are some of the best options for file systems workload testing procedures.

After setting the hypothesis, where it was expected that the physical machine dominates in all fields of given loads comparing to the containers, we proceeded with the application of the experimental method and obtained results that fully justified the assumptions. Based on the comparative analysis method, the obtained results confirmed the initial estimates and expectations, which is proved through the given equations as well as through workloads.

For better understanding and a clearer picture of the container's service capacity, measurements were also performed by increasing the number of Docker instances that worked in parallel, starting from one, until reaching four instances, where all of those were used simultaneously. The decrease of their power was observed and examined.

III. HOST OS AND DOCKER

To install the Ubuntu 20.04 operating system on the host, in this case with hardware characteristics shown in Tables 1 and 2, 1024 MiB of RAM is required at least. With Desktop image, which is the most common, there is the ability to try Ubuntu without changing the current computer system. There is also a Server install image that can be only permanently installed on the machine, but without a graphical user interface.

Experienced users are increasingly opting for Ubuntu when it comes to container operations. We can say that the most important item for security, performance, and quality is the Linux kernel, which always has the latest versions of the kernel accompanied by up-to-date security features. All of the above-mentioned is the reason why the world's largest cloud operators opt for Ubuntu operating system to run their containers [6].

Most users will agree and say that Docker became synonymous with container technology, as it had the greatest impact on popularization. But container technology is not a new term, it has been built into Linux in LXC form for more than ten years, and similar virtualization at the operational level systems was offered by: FreeBSD jails, AIX Workload Partitions, and Solaris Containers [7].

Unlike hypervisor virtualization, container virtualization does not have a hypervisor that would be used as a layer of abstraction, isolation of operating systems and applications from the host operating system. There are two types of hypervisors: type 1, which is mounted directly on the hardware, whereas, on the other hand, we can say that the Docker engine is like type 2, which depends on the host operating system, where the Docker container would be in the virtual machine role (Figure 1) [8].

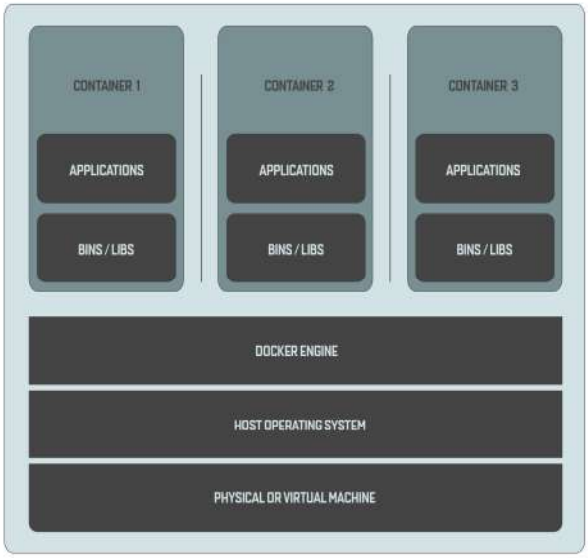

Fig. 1. Docker container-based virtualization

There is a belief that container virtualization is less secure compared with hypervisor virtualization because if weaknesses can be found in the host's kernel on which the containers are located, it could allow intrusion into the containers. The same can be said for the hypervisor, but since the hypervisor provides far less functionality than the Linux kernel (which usually implements file systems, networking, application process controls, etc.) it leaves much less space for attack. In recent years, great efforts have been made to develop software to improve container security. For example, Docker and other container systems now include a signing infrastructure that allows administrators to sign container images to prevent the deployment of unreliable containers [9].

Below is a simple description of docker client-server architecture. Docker client communicates through REST API, over network interface or UNIX sockets with Docker daemon which does building, running and distributing containers (Figure 2) [10]. It is not mandatory that Docker daemon has to run on the same operating system as the Docker client, which can also be connected to a remote daemon [11].

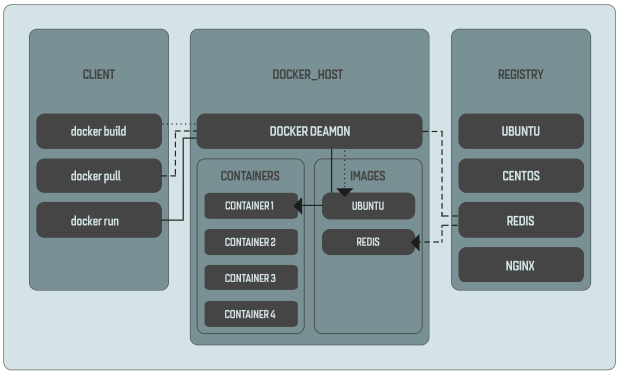

Fig. 2. Docker architecture

Some Linux distributions are designed for running containers and Docker such as Project Atomic [12], Photon OS, RancherOS, etc. [13]. Since 2016, Docker containers have also been able to run on Windows operating system and managed from any Docker client or through Microsoft PowerShell [14].

Docker can also work on popular cloud platforms [15], including Amazon Web Services, Google Compute Engine, Microsoft Azure, Rackspace, etc. [16].

IV. HYPOTHESIS OF EXPECTED BEHAVIOUR

To make it easier to understand how the results were obtained, the following formulas were derived:

$$T_{WKLD} = T_{RR} + T_{SR} + T_{RW} + T_{SW} \quad (1)$$

In equation 1, the *TWKLD* notation stands for the total processing time for each workload. This is followed by a random - *TRR* and a sequential - *TSR* reading time, while the *TRW* notation indicates the random write time, and *TSW* stands for sequential write time. The following formula represents the expected file system access time for each individual workload:

$$T_W = T_{DIR} + T_{META} + T_{FL} + T_{FB} + T_J + T_{HK} \quad (2)$$

The *TW* notation above represents the total time required to complete all operations on the ongoing workload. The following notations represent the time required to complete all operations related to: directory - *TDIR*, metadata - *TMETA*, free list - *TFL*, file block - *TFB*, journaling - *TFJ* and house-keeping - *THK*. There are two candidates for file system performances that are covered in this paper and they are:

1. native HostOS
2. native HostOS + Docker engine + containers

1. The Ubuntu 20.04 operating system is installed on the host with its default file system, and since the Docker containers are running on it, the native host will play a major role in terms of file system performance. For a better comparison with the host, Ubuntu image is pulled and run on all four container instances. Thus, benchmark and the host file system characteristics depend on the time needed to process benchmark-generated workload, and are noted in the following formula as *TW*:

$$T_W(nHost) = f(Bench, hOS\_FS) \quad (3)$$

2. The docker engine has the biggest impact on performance after the host and its file system where everything takes place. As mentioned, HostOS, Docker engine, and containers run on the host file system, except for Docker volumes and self-storage. The benchmark, the host file system characteristics, and Docker engine mapping depend on the time needed to process benchmark-generated workload, in the following formula noted as *TW*:

$$T_W(DOCKER) = f(Bench, D\_engine, hOS\_FS) \quad (4)$$

The obtained performance results of the file system of the host and Docker container were predicted by the given formulas. So, as expected, the host was in the lead through all workloads, which was confirmed by the calculation from equation 3. There are small differences in throughput in all segments between the single running container and the host. This lag in the performance of the container was caused by the Docker engine, which was also confirmed by equation 4. After monitoring the throughput of these instances, the following conclusion was made:

Single Docker container is slightly behind the host performances by all measurements, while for any increase of containers running in parallel by one instance, the deterioration in throughput power should be expected.

V. TEST CONFIGURATION AND BENCHMARK APPLICATION

There are various tools, benchmarks that can measure performance in order to examine the capabilities of physical machines as well as the capabilities of virtual solutions. Some benchmark tools are open-source, while others are commercial solutions. Depending on the purpose of the tests, we can opt for one of the most adequate tools. For these measurements, a FileBench is chosen as one of the most suitable benchmarks.

FileBench is a storage and file system benchmark. It uses its own Workload Model Language (WML) that can allow I/O specification of application behavior. It is one of the best-known open-source tools, which, unlike most of the tools that mainly rely on predefined workloads (which cannot be changed in most cases), allows workload modifications as well as adaptation to the specificities of the purpose for which the testing is performed.

Installing a FileBench benchmark is quite simple after downloading the software package. However, on Ubuntu, it requires a few more commands than on Centos operating system, for instance, where it is possible to install it with a simple "yum install filebench" command. Additionally, there is a difference in the installation of the benchmark between two versions covered in this paper. In the first part of the installation, as the configuration files are not included in the repo, they have to be created. Therefore, for the last stable version, it is necessary to run the following commands if they are not installed, respectively: libtoolize, aclocal, autoheader, automake, --add-missing, autoconf.

The second part of the installation requires the installed gcc, flex and bison in order to run FileBench [17]. This part is the same as in the 1.5-alpha3 version, except that in this version it is the only step and it involves running the following commands, respectively: ./configure, make, make install.

In order to measure as accurately as possible and to obtain as better results as possible, Ubuntu 20.04 operating system was installed on the host (hardware shown in Tables 1 and 2) only for this file system test purpose, which after the installation of the benchmark had no other applications that could disrupt the operation of this tool in any way. Also, containers had nothing but installed FileBench.

After everything is set, there is still one thing left to do and that is disabling ASLR (address space layout randomization) by changing the value of randomize_va_space to "0" (zero), otherwise, the workloads will be blocked in the stage of running.

TABLE I
HARDWARE CONFIGURATION OF THE HOST

Component	Characteristics
Processor	AMD Ryzen 5 3600X, 3.8GHz – 4.4GHz, 6 Core, 12 Thread
Cache	L1 Cache 384KB, L2 Cache 3MB, L3 Cache 32MB
Memory	16Gb DDR4, 3200MHz
SSD	Kingston A2000 SA2000M8/500GB
Motherboard	GIGABYTE B450M DS3H

TABLE II
SSD characteristics

Capacity	500GB
DRAM	DDR4
Interface	NVMe™ PCIe Gen 3.0 x 4 Lanes
Form factor	M.2 2280
NAND	3D TLC
Sequential Read/Write	up to 2.200/2.000MB/s
Random 4K Read/Write	up to 180.000/200.000 IOPS

VI. TESTS AND RESULTS

Each measurement was done in three rounds per host and per each container instance, after which the average value was taken for results. The obtained measurements of individual container performances were then compared with the results obtained while testing the host. The throughput of each container was observed in cases when only one container instance was started, when two instances were running in parallel, and when three and then four containers were running at the same time.

File system performance tests were conducted on the latest stable version of FileBench - 1.4.9.1 and 1.5-alpha3 version where throughput was measured in MB/s. For the purposes of this experiment, four of the over fifty predefined workloads were selected. On both versions, the performance of the filesystem was tested via three workloads that were used to emulate applications, namely: fileserver, webserver and varmail. On the last stable version, an additional workload was included - radnomfileaccess. The following is a brief description of workloads that were used and covered with formulas (1) and (2): *Fileserver* – It mimics the elementary I/O activity of a file server. It performs a sequence of creating, deleting, adding, reading, writing, and attribute operations on a directory tree; *Webserver* - Mimics elementary I/O activity of a web server. Produces an open-read-close sequence on multiple files in a directory tree, plus appends a log file; *Varmail* - Imitates elementary I/O activity of a mail server that saves each e-mail in an isolated file (/var/ mail/server). It contains a set of multiple threads of the following operations in a particular directory: create-add-sync, read-add-sync, read, and delete; *Randomfileaccess* - Uses random variables that are user-defined entities, and these entities are formulated by a random distribution that is used to select a random value that is returned with each use [18].

It is hard not to mention virtual clusters when Docker containers are used. Testing could take on a completely different dimension if any container orchestration platforms such as Kubernetes were used, where containers would combine and pool their serving powers [19]. But the purpose of these tests was to compare the file system performance of the host and individual container.

The parameters shown in Tables 3 and 5 are set with default values. The values for the four specified parameters (number of files - nfiles, average file width, and size - (meandirwidth, meanfilesize), as well as the number of threads - nthreads) are the same in both versions of the benchmark. The time for executing each of the workloads is set to 60 seconds, which is the default value for most of the predefined workloads.

TABLE III
PARAMETERS OF THE SOURCE CODE *.F FILES (1.4.9.1 VERSION)

Workload (runtime 60s)	Fileserver	Webserver	Varmail	RFA
nfiles	10.000	1.000	1.000	10.000
meandirwidth	20	20	1.000.000	20
meanfilesize	128k	16k	16k	Random
nthreads	50	100	16	5

TABLE IV
BENCHMARK RESULTS (MB/S), 1.4.9.1 VERSION

Instance	Fileserver	Webserver	Varmail	RFA
Host	3866.6	1001.5	187.4	19081.8
1 container	3746.1	962.8	180	18190.1
2 containers	1764.4	695.9	158.3	9438.5
3 containers	1170.5	528.5	137.2	5300.7
4 containers	651.7	458.8	117.2	3809.8

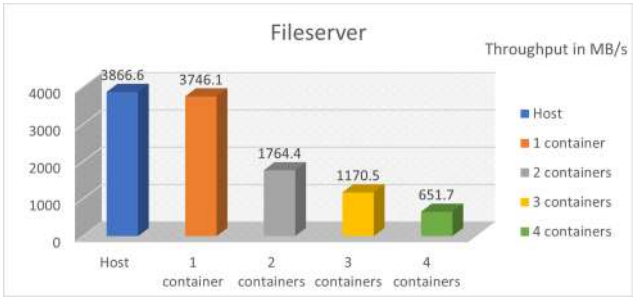

Fig. 3. Fileserver test results from Table 4

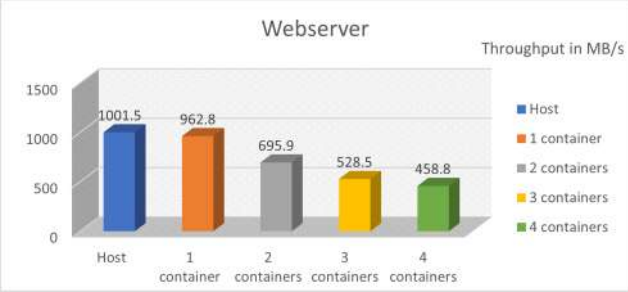

Fig. 4. Webserver test results from Table 4

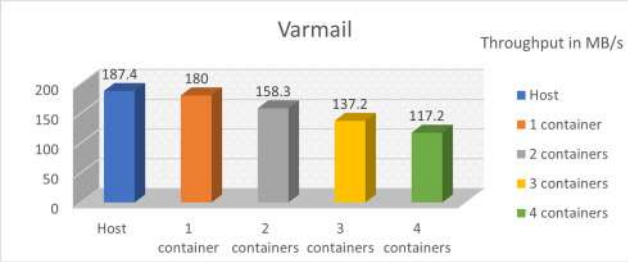

Fig. 5. Varmail test results from Table 4

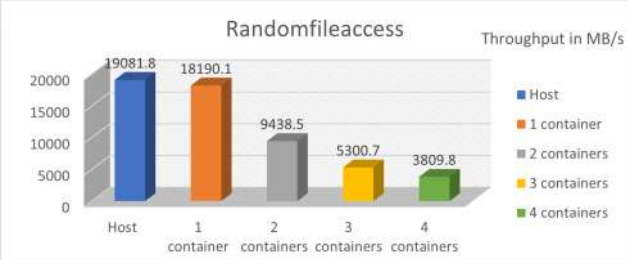

Fig. 6. Randomfileaccess test results from Table 4

A. Measurements performed on version 1.4.9.1

The host had better performance in all four categories which is shown in Table 4. The obtained results were proved by formulas (3) and (4). Starting with the fileserver environment, there is a small throughput difference of 3 % in favor of the host compared to a single container. Then, as expected, by increasing the number of containers by one, the serviceability also decreases, so that the performance of the two running containers drops by more than twice, i.e. 54%. Performance with three running containers deteriorated by 70% and with four instances the results showed it to be 83% (Figure 3).

For webserver tests, the results are as follows. The throughput at the host instance is 4% higher when compared to a single running container, while for two running containers that gap is 30%. With three and four containers in running state, we can see the degradation of 47% and 54%, respectively (Figure 4).

In the case of varmail environment, the single running container has lower performances by 4%, two containers by 16%, and three and four containers by 27% and 37% compared to the host (Figure 5).

The randomfileaccess workload also had poorer container results, showing performance declines of 5, 51, 72, and 80% when having 1, 2, 3, and 4 containers in running state, respectively (Figure 6).

TABLE V
PARAMETERS OF THE SOURCE CODE *.F FILES (1.5-ALPHA3 VERSION)

Workload (runtime 60s)	Fileserver	Webserver	Varmail
nfiles	10.000	1.000	1.000
meandirwidth	20	20	1.000.000
meanfilesize	128k	16k	16k
nthreads	50	100	16

TABLE VI
BENCHMARK RESULTS (MB/S), 1.5-ALPHA3 VERSION

Instance	Fileserver	Webserver	Varmail
Host	4072.6	3333.8	163.5
1 container	4007.1	3080.1	133.4
2 containers	1696.7	1563.5	111.2
3 containers	704.6	1160.8	88.5
4 containers	451.4	952.6	76

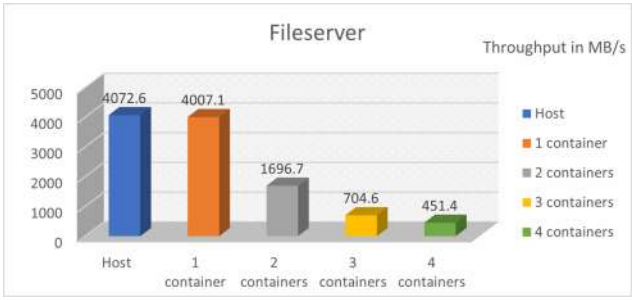

Fig. 7. Fileserver test results from Table 6

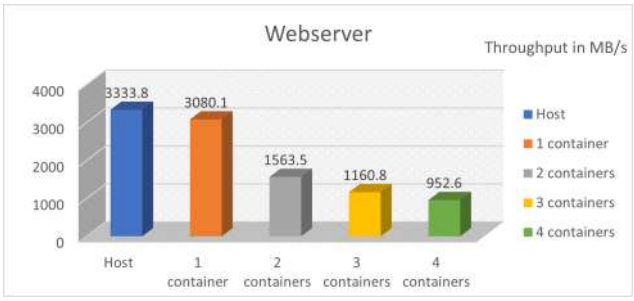

Fig. 8. Webserver test results from Table 6

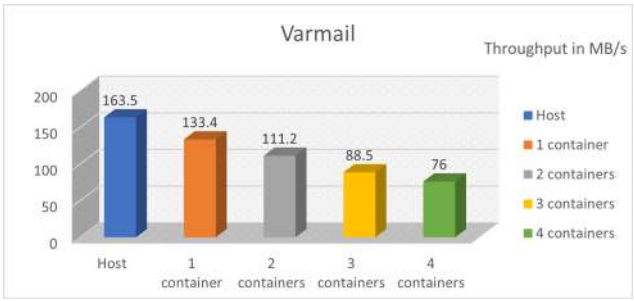

Fig. 9. Varmail test results from Table 6

B. *Measurements performed on version 1.5-alpha3*

Within a FileBench version 1.5-alpha3, the expected results were obtained, which is verified by formulas (3) and (4). As well as in the latest stable version the host dominates (Table 6). In the fileserver case, a single container performance does not significantly differ from the host and it is lower by 2%, while for two container instances in running state the drop is much bigger, 58%. For three and four instances it is 83% and 89%, respectively per container (Figure 7).

With webserver workload tests we have a throughput deterioration comparing to host, namely 8% for a single container, 53% in the case of two instances, 65% for three running containers, and 71% per instance in the case of four containers running (Figure 8).

As for varmail, the host throughput is higher by 18% compared to a single container, while for two instances there is gap of 32% per instance, it is 46% for three instances and 54% for all four containers (Figure 9).

VII. CONCLUSION

According to the shown tests, the host had better performance in all segments compared to Docker containers which justifies the hypothesis. During performance monitoring through all four workloads, a slight differences in throughput between the host and single container is noticeable. As we can see in the obtained measurement results, the increase of the number of container instances decreases their service power, which also differs from workload to workload. Those are expected results, and accordingly, depending on the load, we can determine whether containers are suitable and if they will meet the requirements for which container instances were originally intended.

This is only a small segment in testing the host and Docker container capabilities, as there are over forty predefined tests left, as well as many variations of modifying existing and writing your own workloads that can be processed. Since FileBench workloads can be easily managed it leaves a lot of room for future measurements and comparisons with the results of other benchmarks that are not so flexible in terms of tests.

Today, it is known that hardware development is increasingly focusing on multi-core solutions that can process many instructions in a very short time. That leaves plenty of room for further processing of power and resources, which is suitable for the normal and smooth operation of virtual solutions. Virtualization is not always the answer to everything, for some purposes virtualization simply does not achieve the desired results so in that case, the only choice is a physical machine. But in most cases, security, productivity, and cost-reducing benefits outweigh all problems, and therefore Docker virtual solutions and virtualization, in general, are increasingly gaining in popularity.

ACKNOWLEDGMENT

The work presented in this paper has partially been funded by the Ministry of Education, Science, and Technological Development of the Republic of Serbia.

LITERATURE

[1] C. Walls, "Hardware and software development; what's the cost?," 2018 [online]: https://www.embeddedcomputing.com/technology/software-and-os/hardware-and-software-development-what-s-the-cost

[2] IBM Cloud Team, IBM Cloud. Containers vs. VMs: What's the difference? IBM, 2020. [online]: https://www.ibm.com/cloud/blog/containers-vs-vms

[3] Spiceworks. The 2020 State of Virtualization Technology, 2019. [online]: https://www.spiceworks.com/marketing/reports/state-of-virtualization/

[4] K. Thompson, "Hardware vs. Software development: Similarities and Differences," Cprime, 2015. [online]: https://www.cprime.com/resources/blog/hardware-vs-software-development-similarities-and-differences/

[5] T. Collins, "Virtual servers vs physical servers: Which is best? 10 March," Atlantech, 2020. [online]: https://www.atlantech.net/blog/virtual-servers-vs-physical-servers-which-is-best

[6] Canonical. Why is Ubuntu #1 OS for containers? Ubuntu, 2018. [online]: https://ubuntu.com/containers

[7] S. Hogg, "Software Containers: Used More Frequently than Most Realize,", Networkworkd, 2014 [online]: https://www.networkworld.com/article/2226996/software-containers--used-more-frequently-than-most-realize.html

[8] U. Hiwarale, "Anatomy of Docker," [online]. 2018 Nov [Accessed 24 February 2021]. Available from: https://itnext.io/getting-started-with-docker-1-b4dc83e64389

[9] P. Rubens, "What are containers and why do we need them?,", Cio, 2017 [online]: https://www.cio.com/article/2924995/what-are-containers-and-why-do-you-need-them.html

[10] N. Poulton, Docker Deep Dive, JJNP Consulting Limited, Lean Publishing. Leanpub book, 2018.

[11] Docker. Docker overview. [online]: https://docs.docker.com/get-started/overview/#docker-architecture

[12] SP. Kane, K. Matthias, "Atomic hosts," in Docker: Up and Running. 2nd ed. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472; 2018.

[13] JM. Scheuermann, "A Comparison of Minimalistic Docker Operating Systems," Inovex, 2015. [online]: https://www.inovex.de/blog/docker-a-comparison-of-minimalistic-operating-systems/

[14] M. Friis, "Build and run your first Docker Windows Server container," Docker, 2016. [online]: https://www.docker.com/blog/build-your-first-docker-windows-server-container/

[15] C. Ward, "The Shortlist of Docker Hosting," CloudBees, 2016. [online]: https://www.cloudbees.com/blog/the-shortlist-of-docker-hosting/

[16] J. Turnbull, "Installing Docker," In The Docker Book. CC BY-NC-ND 3.0; 2018.

[17] G. Amvrosiadis, "FileBench," GitHub, 2016 [online]: https://github.com/filebench/filebench

[18] V. Tarasov, "Predefined personalities. [online]. 2016 Jul [Accessed 24 January 2021]. Available from: https://github.com/filebench/filebench/wiki/Predefined-personalities

[19] IM. Aidan, H. Sayers, "Using a Kubernetes cluster," in Docker in Practice. Manning Publications Co. 20 Baldwin Road PO Box 761 Shelter Island, NY 11964; 2016.