A Comparison of Container Systems for Machine Learning Scenarios: Docker and Podman

Konstantinos Voulgaris
Department of Digital Systems
University of Piraeus
Piraeus, Greece
kvoulgaris@unipi.gr

Athanasios Kiourtis
Department of Digital Systems
University of Piraeus
Piraeus, Greece
kiourtis@unipi.gr

Andreas Karabetian
Department of Digital Systems
University of Piraeus
Piraeus, Greece
adreaskar@unipi.gr

Panagiotis Karamolegkos
Department of Digital Systems
University of Piraeus
Piraeus, Greece
pkaram@unipi.gr

Yannis Poulakis
Department of Digital Systems
University of Piraeus
Piraeus, Greece
gpoul@unipi.gr

Argyro Mavrogiorgou
Department of Digital Systems
University of Piraeus
Piraeus, Greece
margy@unipi.gr

Dimosthenis Kyriazis
Department of Digital Systems
University of Piraeus
Piraeus, Greece
dimos@unipi.gr

***Abstract*—With the rising needs of corporations and their users, as well as the ever-increasing complexity of applications, Container engines and technologies have become a pivotal point in every modern technological infrastructure. This has standardized execution runtimes and when combined with modern virtualization techniques, it has lowered costs and given access to the power of cloud computing to many more people. The aim of this paper is to compare two of the most popular container engines to see what differences exist in performance and architectural levels between so-called "drop-in" replacements. To ensure consistency and replicability of testing, we standardize the benchmark environment with a custom-build tool that describes differences between container engines in the millisecond range. On this basis, we also present quantified results and compare differences with real-world metrics and pricing. From our results we observe a small but not insignificant difference between the container systems, in favor of Docker. Our results are quantified with real-world pricing of cloud resources to figure out how much more costly Podman would be in a cloud deployment.**

Keywords—containers, docker, podman, open container initiative, benchmarking, cloud computing

I. Introduction

Developing modern web applications has proven to be a difficult task for developers to maintain. Not only for their complex nature, with back-end and front-end systems communicating constantly and changing, but more importantly with the environment that those web applications will live on. Early virtualization efforts and investment in a cloud infrastructure provided a solution to sharing physical dedicated resources from a single machine to multiple clients, and later with the Virtual Private Server (VPS) which has the ability to serve multiple clients from a shared pool of resources and remains widely used even to this day [1]. Virtualizing and servicing multiple business customers from a single physical box has lowered costs, availability, and waste across the board. However, the rise of more complex data-driven web applications required new, much more viable solutions to be invented to satisfy an ever-increasing user base and computing requirements. With this, came the idea of a container system that aims to use advancements in virtualization technology to standardize and isolate the execution environment for applications. Such tools that arose are Docker, Podman, LXD, containerd, and more [2]. These tools also enable developers to make use of different, more innovative development approaches and architectures such as the one of microservices [3]. With a microservice-oriented architecture, the developer aims to split the most complex problems of the application into many easier ones. Thus, the developers not only have a clearer idea of the problems they are trying to solve but also make the whole developer experience much easier. In addition to the ease of solving problems, these tools give the ability for easier correction and therefore better maintenance of the mentioned applications. Due to the variety of tools available for containerization, it is important to perform comparison studies to find the areas in which each tool is more efficient, under different criteria. Thus, using that information, developers will know which tool will be the optimal for their use case. More specifically, while the most common tool used in the industry is Docker, Kubernetes during the creation of a cluster gives the ability to choose among tools like the ones, such as Docker and containerd, that will be used by all Kubernetes nodes [4]. This is a good example of the burden of choice that a Kubernetes Captain will have to make. As reported by RedHat, Kubernetes considered Docker's support to be deprecated [5] which RedHat had already done in its Linux distribution Red Hat Enterprise Linux (RHEL) and has moved on to Podman as its main container engine [6].

Therefore, in this paper, we aim to conduct a comparative study between Docker and Podman, to test their differences, both autonomously and in a scalable Kubernetes environment. In more detail, we aim to gather data for performance metrics of importance and quantity said results in a manner that will reflect the needs of modern infrastructure administrators. Moreover, the results will be mainly based on CPU usage using the virtual CPU (vCPU) time units of measurements that is the standard unit that calculates pricing across popular cloud providers.

The rest of this document is structure as follows. Section II provides the related work regarding benchmarking, as well as the architectural differences between Docker and Podman. In Section III the overall comparison testing takes place, depicting the criteria and characteristics of the working environments, while Section IV includes the derived results. Section V concludes with a discussion of the experimental insights, as well as our next steps.

II. Related Work

A. *Software Testing*

Testing software alternatives for matters of the utmost importance tends to be an interesting yet neglected subject. At the time of writing this manuscript, very few have focused on testing the performance characteristics of container systems comparatively. Currently, research papers tend to focus more on isolated environments or use cases [7], and not on the many of the core differences presented at an architecture or performance-based level among containerization engines and schemes. Furthermore, benchmarking tends to be user-specific as most users (i.e., software engineers and developers) tend to use what works best for them either due to convenience, an already established level of familiarity, or limitations due to existing codebases [8]. On a secondary level, a lot can be discussed and argued in the interest of the benchmark test itself, as many questions tend to be raised, such as "what is a heavy workload?", or "will the test scale proportionally with hardware advancements?". Even with that in mind, another question, and perhaps the most important of all is raised, about "what do we plan on benchmarking?". This is because many scenarios tend to have different needs and priorities. For example, in [9] static analysis tools for the security of web applications are benchmarked and compared. We want to test purely performance metrics and as mentioned before, work has been done from a container-specific angle, much like in [9] where a benchmark tool for measuring Docker performance is described named "DocLite". We will focus on purely the performance-based differences from changing the container system in a fixed scenario (described in Section III), to see how faster or slower the application would be.

B. *Podman and Docker Arcitectural Differences*

Most container systems in the market comply with the Open Container Initiative (OCI) specification. This specification is split into two main sections, the Runtime Specification and the Image Specification [11]. Both Docker and Podman are OCI compliant, therefore we can consider each one to be a drop-in replacement for the other in many use cases. However, many differences exist on an architectural level and if we look in more detail, some differences are immediately noticeable.

Docker is designed to base its operations on a daemon, an ongoing program that runs in the background of the operating system. Podman uses a daemon-less architecture which means that it does not need to call a daemon and effectively performs its operations on a different system user. Expanding this, Docker uses a typical client-server logic, much like requests on the internet, due to the use of the daemon interface, whereas Podman has no need for such logic and routing. Furthermore, since Podman does not have a daemon to manage its activity, it also does not apply many root escalations to its containers. Contrary to that, Docker tends to be installed with some root escalations in mind, however recently it added a rootless mode to its daemon configuration. However, it will take some time for existing tools and installations to be updated, whereas Podman was developed and advertised with such behavior from its beginning. Rootless containers are considered safer since they will require extra work or clever use of software vulnerabilities to both escape the container isolation and enter the root user. Since Podman was designed both as daemon-less and lightweight there are some intricacies that might exclude some stakeholders. Take for example that Podman needs to manage the use of the services and init system of the operating system to assist its operations. In Linux systems, this tends to be Systemd which allows Podman to be deeply integrated with the system and thus provide Systemd to its containers as well. [12][13]

As it might have become clear, the major difference between these two can be summarized in the different philosophies that are presented. Docker uses a monolithic approach that aims to serve the user a single powerful and independent tool, handling all the containerization and imaging tasks throughout the entire lifecycle of development. On the other hand, Podman has a modular approach, relying on specific tools for specific duties which tends to comply better with the Unix philosophy for software which implies "Make each program do one thing well", as described in [14]. This point is made even more clear if it is being considered that Podman relies on the assistance of another tool called Buildah to build container images, while Docker handles it on its own [15].

Finally, if someone is considering Podman as their container engine, they can just alias Docker to Podman without any problems due to them both complying with the OCI specification. Users might want to start by using both tools interchangeably to see which parts of their architecture are better served by each option.

III. Testing

To quantify our results, we need to test the container systems across a standard testing environment. To achieve this, we built a benchmarking tool that floods an API with requests to see how each container environment would fare in low and high-intensity workloads (Fig. 1). Simultaneously, the benchmark tool can measure the delays between requests in the microsecond range using the Python Requests library and exports them as Comma Separated Values (CSV) files to be processed either by other scripts or by other any modern spreadsheet software. While benchmarking, two modes are supported, one for sequential requests and one for simultaneous requests. When benchmarking sequential requests, the tool sends requests a "n" number of times and stops when the test is either complete or if the requests start to fail. On the other hand, when benchmarking simultaneous requests, the tool creates multiple workers to handle a "n" number of simultaneous requests and stops when the test is either complete or if the requests start to fail, as well.

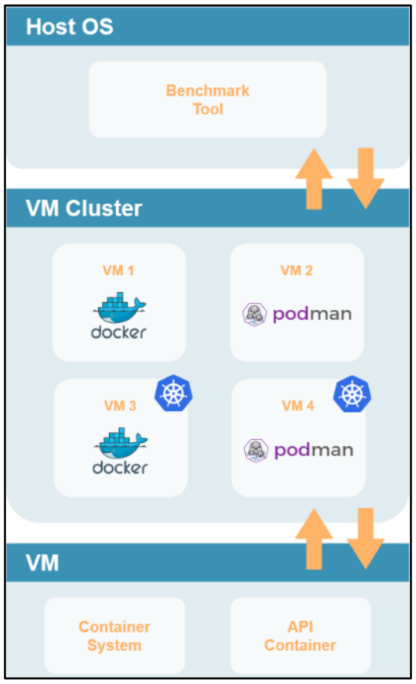

Fig. 1. *Benchmarking environment flow*

For our testing scenario, we are going to build an image classifier using the VGG16 model [16] as its base and use transfer learning methods to classify between healthy and pneumonia or COVID-infected human lungs from the dataset provided in [17]. A snapshot of this dataset is depicted in Fig. 2.

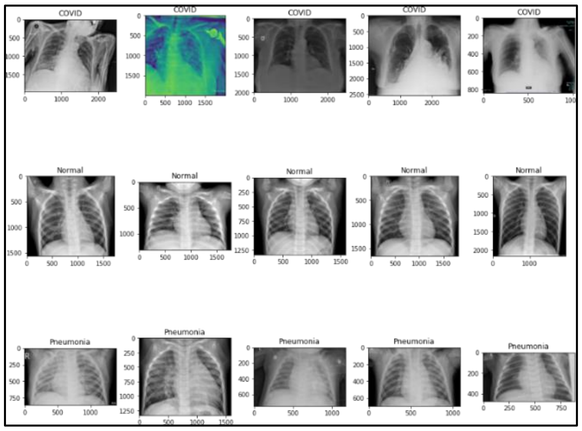

Fig. 2. *Dataset Sample Snapshot*

The dataset consists of 6000 X-ray images with appropriate train and test splits. To make the classifier model accessible, we also developed a user-facing API using the Python Flask micro web framework. The API consists of one index path, which accepts HTTP requests of the type POST with an image file and then returns the answer from the model's attempt to classify it. Finally, the container that houses the testing scenario uses the latest Ubuntu base image, the Gunicorn WSGI-compliant server to serve the Flask application.

IV. Results

Our benchmark results include both sequential and simultaneous testing using the benchmark tool described in Section III. To have a baseline environment, testing was performed on four (4) separate Virtual Machines (VMs) of 4 CPUs and 8GB of RAM, each with an isolated installation of each scenario, meaning one for Docker, one for Podman, and two each for their Kubernetes instances, respectively. The VMs were exposed to the internet and the tool was executed on a different machine in order to avoid any variables or slowdowns that would be caused by the processing requirements of the benchmark tool. We will present results from benchmarking performed in a Kubernetes environment for each container system where the application had a total of four (4) replicas in the Kubernetes cluster. Additionally, the results will be presented as a group of line graphs on a logarithmic scale and the moving average of the last 10% of results will be given where necessary for further clarity. The vertical axis will represent the milliseconds that it took for each request to complete, and the horizontal axis will represent the ever-increasing number of requests that were benchmarked.

A. *Sequential Results*

In this subsection, we will look at the results from sequential testing, both standalone and as well in the Kubernetes cluster. At a first glance at Fig. 3, both in sequential and simultaneous requests, both Docker and Podman see little or no deviation from each other and provide us little information as to which is better.

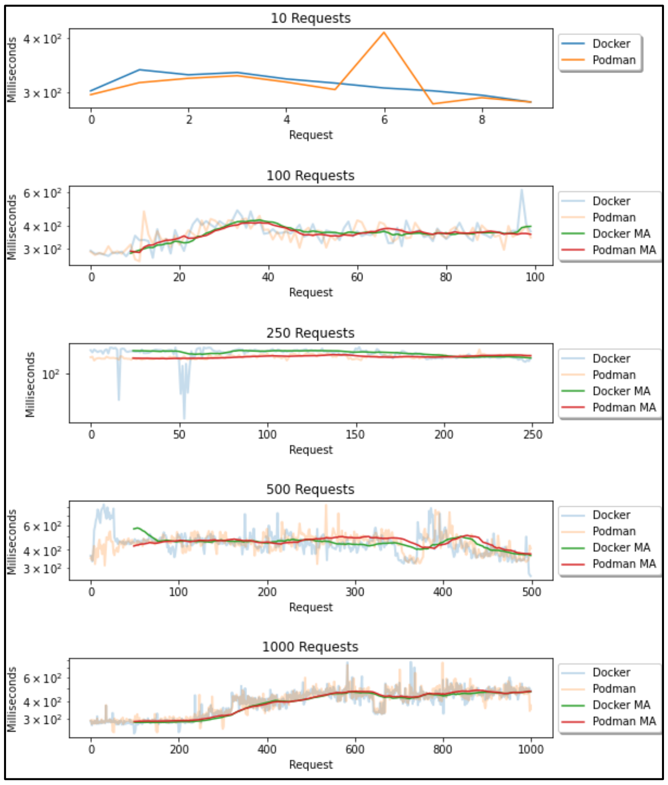

Fig. 3. *Sequential Results*

As it can be seen in Fig. 4, results differ a lot from standalone testing, and it becomes clearly obvious that Docker has the tendency to perform a lot better. While the difference for this remains unknown, our opinion is that Docker and Kubernetes have more time to mature when working together, in contrast to Podman support which is in its early stages.

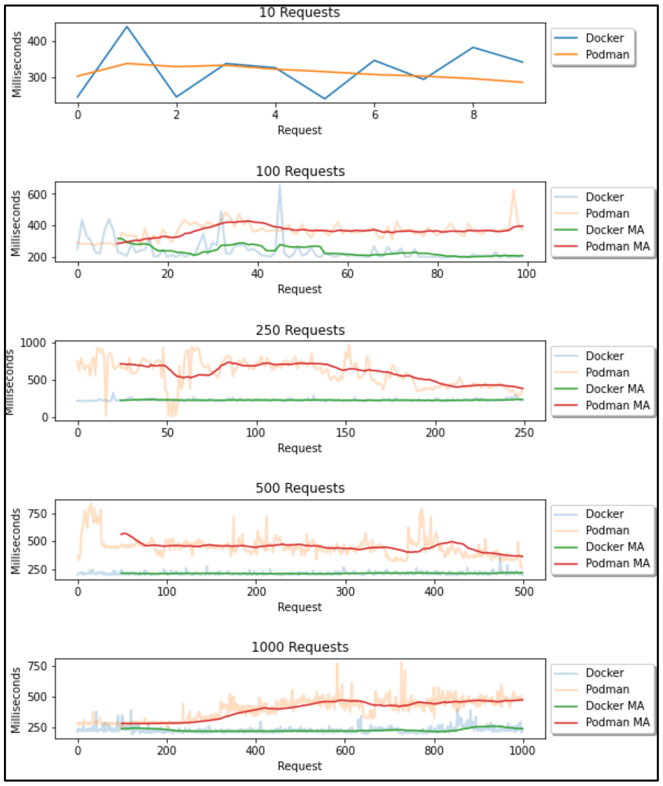

Fig. 4. *Sequential Results in a Kubernetes Environment*

B. *Simultaneous Results*

In this subsection, we will look at the results from simultaneous testing, both standalone and in the Kubernetes cluster. Similarly, with the sequential tests, both Docker and Podman see little or no deviation from each other (Fig. 5).

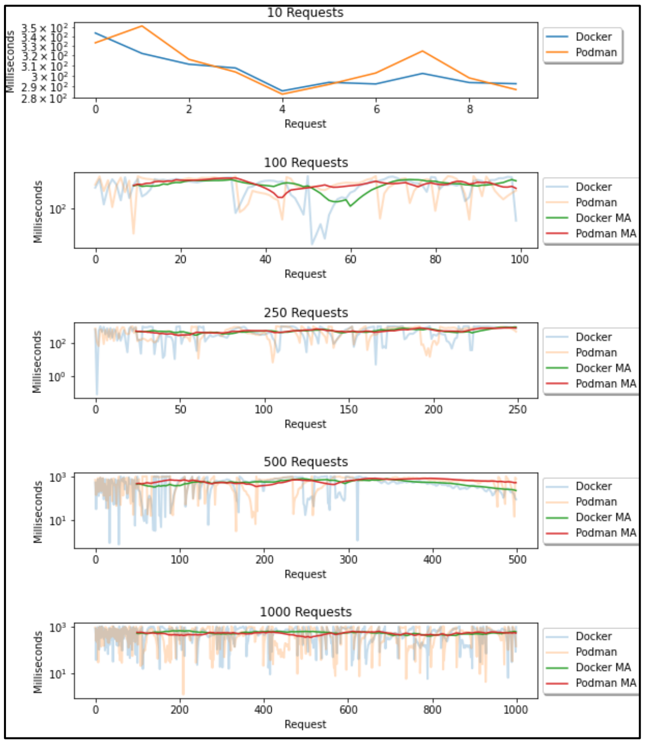

Fig. 5. *Simultaneous Results*

However, contrary to sequential tests in a Kubernetes Environment, simultaneous tests in a Kubernetes environment also perform very similarly between Docker and Podman, with Podman just slightly edging out Docker, based on the results depicted in Fig. 6.

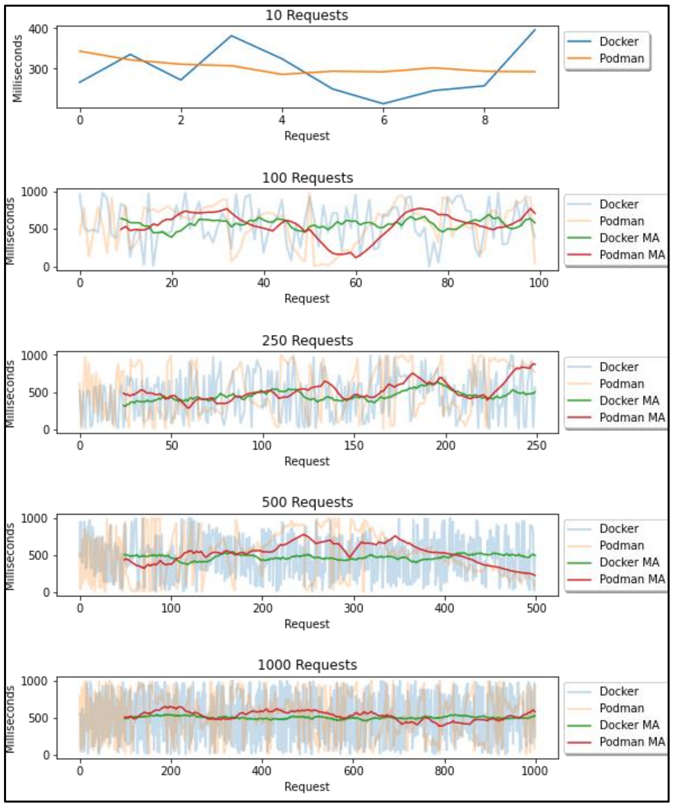

Fig. 6. *Simultaneous Results in a Kubernetes Environment*

C. *Quantifying Results*

In this subsection, we aim to quantify our results and graphs in real-world metrics. We will simplify our results by aggregating the milliseconds of processing time required for each level of request load. This is because the request load is the only variable that changes and scales across the benchmark and across all benchmark modes. This is presented in Fig 7., and looking at the differences between these two container systems, it becomes clear that Podman tends to be slower than Docker across the board.

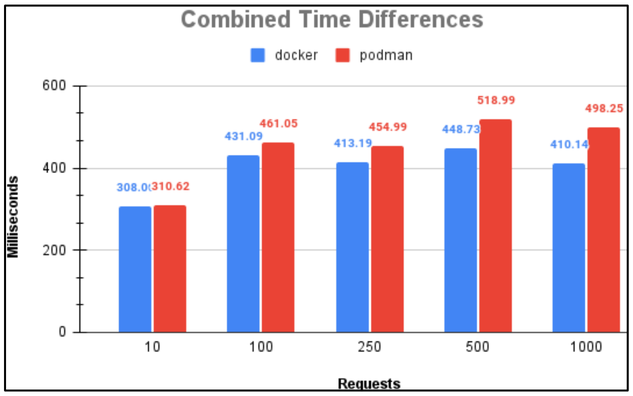

Fig. 7. *Aggregated Processing Time Across Benchmark Results*

However, the data in Fig. 7 does not tell us anything of real-world importance. In order to more accurately describe the differences, in Fig. 8 we present the differences in vCPU time usage when using a serverless provider to serve a containerized application. We assume that Docker has a baseline 20 vCPU time usage and calculate how much more vCPU time Podman will require when keeping in mind the extra milliseconds of processing that will occur.

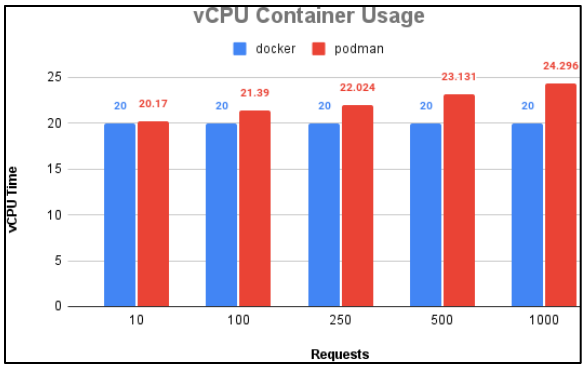

Fig. 8. *Calculated units of vCPU time between container systems*

While differences in this example do not result in any differences in charging, if we scale this out to a larger scenario and assume an average increase in request processing time with Podman of 10%, we can calculate that if we used a serverless instance on AWS [18] of 4 vCPUs and 8GB RAM and assuming a base 1000 vCPU time usage, we would conclude to an additional 11.22 USD in billing just for using a different container system.

V. Discussion and Concluding Remarks

In this paper, it has been compared two of the most popular container engines, namely Docker and Podman. As it has been previously discussed, since modern infrastructure makes heavy use of containers in order to better orchestrate and manage complex applications, it is mandatory to have a basis with which we can compare solutions to one another, while keeping in mind their strengths and differences. In order to complete and fairly compare this comparison, a benchmarking tool was created that standardizes and measures the differences between runtimes. Additionally, a machine learning with health data scenario was also constructed, in order to better represent real-world use cases instead of just brute-forcing results. Finally, as we have seen in our quantified results, while the container systems tend to behave and measure similarly when extrapolated to larger scales, these differences start to add up to an extent that Podman ends up proving itself to be more expensive to maintain when basing our pricing on popular cloud providers, at least when compared to Docker.

Currently, our results are limited to a single machine learning scenario and with just two container runtimes. As the OCI specification is an open standard, many more options are available that anyone can use to experiment even further. From our point of view, since it has been presented a standard baseline of comparing containerization engines and schemes, we aim to further expand our work in the future, to include additional scenarios and container engines, covering multiple sectors and domains. These include, but not limited to Data-as-a-Service scenarios, or Serverless scenarios [19], as well as the LXD, containerd engines which were mentioned in Section I. As soon as this will be achieved, we expect that this work will provide infrastructure admins and software engineers a complete scientific overview of the benefits and shortcomings cons of each container engine in order to better fit their needs.

Acknowledgment

This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH - CREATE - INNOVATE (project code: DIASTEMA - T2EDK-04612).

References

[1] M. Masdari, S. Nabavi, and V. Ahmadi, “An overview of virtual machine placement schemes in cloud computing,” Journal of Network and Computer Applications, vol. 66, pp. 106-127, 2016.

[2] N. Bachiega, P. Souza, S. Bruschi, and S. de Souza, “Container-Based Performance Evaluation: A Survey and Challenges,” IEEE International Conference on Cloud Engineering (IC2E), pp. 398-403, 2018.

[3] N. Alshuqayran, N. Ali, and R. Evans, “A Systematic Mapping Study in Microservice Architecture”, IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), pp. 44-41, 2016.

[4] “Container Runtimes”, 2022. Available: https://kubernetes.io/docs/setup/production-environment/container-runtimes/.

[5] S. McCarty, “Kubernetes is Removing Docker Support, Kubernetes is Not Removing Docker Support”, 2022. [Online]. Available: https://cloud.redhat.com/blog/kubernetes-is-removing-docker-support-kubernetes-is-not-removing-docker-support.

[6] “Why Red Hat is investing in CRI-O & Podman”, 2022. Available: https://www.redhat.com/en/blog/why-red-hat-investing-cri-o-and-podman.

[7] A. Potdar, N. D G, S. Kengond, and M. Mulla, “Performance Evaluation of Docker Container and Virtual Machine,” Procedia Computer Science, vol. 171, pp. 1419-1428, 2020.

[8] H. Knoche, and W. Hasselbring, “Using Microservices for Legacy Software Modernization”, IEEE Software, vol. 35, no. 3, pp. 44-49, 2018.

[9] P. Nunes, et al., “Benchmarking Static Analysis Tools for Web Security”, IEEE Transactions on Reliability, vol. 67, no. 3, pp. 1159-1175, 2018.

[10] B. Varghese, L. Subba, L. Thai and A. Barker, “DocLite: A Docker-Based Lightweight Cloud Benchmarking Tool”, 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 213-222, 2016.

[11] “About the Open Container Initiative - Open Container Initiative”, 2022. Available: https://opencontainers.org/about/overview/.

[12] “Docker Documentation”, 2022. Available: https://docs.docker.com/.

[13] “What is Podman? - Podman documentation”, 2022. Available: https://docs.podman.io/en/latest/.

[14] E. Raymond, “The art of Unix programming”, Boston: Addison-Wesley, 2004.

[15] “Podman and Buildah for Docker users | Red Hat Developer”, 2022. Available: https://developers.redhat.com/blog/2019/02/21/podman-and-buildah-for-docker-users.

[16] K. Simonyan, et al., “Very Deep Convolutional Networks for Large-Scale Fig. Recognition”, arXiv preprint arXiv:1409.1556, 2014.

[17] D. Kermany, et al., “Identifying Medical Diagnoses and Treatable Diseases by Fig.-Based Deep Learning,” Cell, vol. 172, no. 5, pp. 1122-1131.e9, 2018.

[18] “EC2 On-Demand Instance Pricing – Amazon Web Services”, 2022. Available: https://aws.amazon.com/ec2/pricing/on-demand/.

[19] D. Kyriazis, et al., “Policycloud: analytics as a service facilitating efficient data-driven public policy management,” IFIP international conference on artificial intelligence applications and innovation, pp. 141-150, 2020.

Authors' background

Name	*Prefix*	*Research Field*	*Email*	*Personal website*
Konstantinos Voulgaris	Undergraduate Student	Data Management, Cloud Computing, Machine Learning	kvoulgaris@unipi.gr	https://dac.ds.unipi.gr/voulgaris/
Athanasios Kiourtis	Postdoctoral Researcher	Data Interoperability, Data Exchange Protocols	kiourtis@unipi.gr	https://dac.ds.unipi.gr/kiourtis/
Andreas Karabetian	Undergraduate Student	Visual Programming Environments, Web Applications	adreaskar@unipi.gr	https://dac.ds.unipi.gr/karabetian/
Panagiotis Karamolegkos	Undergraduate Student	Data Management, Cloud Computing	pkaram@unipi.gr	https://dac.ds.unipi.gr/karamolegos/
Yannis Poulakis	PhD Candidate	Automated Machine Learning, Optimization, Clustering, Semantic Descriptions	gpoul@unipi.gr	https://dac.ds.unipi.gr/poulakis/
Argyro Mavrogiorgou	Postdoctoral Researcher	Internet of Things, Cyber-Physical Systems, Data Analysis	margy@unipi.gr	https://dac.ds.unipi.gr/mavrogiorgou/
Dimosthenis Kyriazis	Associate Professor	Cloud computing, Data Management	dimos@unipi.gr	https://www.ds.unipi.gr/en/faculty/dimos-en/

Note:

[1] **This form helps us to understand your paper better; the form itself will not be published.**

[2] *Prefix*: can be chosen from Master Student, PhD Candidate, Assistant Professor, Lecture, Senior Lecture, Associate Professor, Full Professor