Performance Evaluation of Container Management Tasks in OS-level Virtualization Platforms

Pedro Melo*, Lucas Gama*, Jamilson Dantas†, David Beserra‡, and Jean Araujo*
*Universidade Federal do Agreste de Pernambuco, Garanhuns, Brazil
†Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil
‡École Pour l'Informatique et les Techniques Avancées (EPITA), Paris, France
{pedro.melosilva, lucas.lopesgama, jean.teixeira}@ufape.edu.br*, jrd@cin.ufpe.br†, david.beserra@epita.fr‡

***Abstract*—Cloud computing is a method for accessing and managing computing resources over the internet, providing flexibility, scalability, and cost-efficiency. Cloud computing relies more and more on OS-level virtualization tools such as Docker and Podman, enabling users to create and run containers, which are widely used for application management. Given its significance in cloud infrastructures, it is crucial to have a better understanding of OS-level virtualization performance, especially in tasks related to container management (ex: creation, destruction). In this paper, we conducted benchmarking tests on Docker and Podman to evaluate their performance in various container management scenarios and with different image sizes. The results revealed that Podman excels in quickly instantiating small-sized containers, while Docker demonstrates superior performance with larger-sized containers.**

***Index Terms*—Containers, Docker, Performance Evaluation**

I. Introduction

Cloud computing is a widely adopted paradigm that involves the provisioning of computing resources as services over the Internet in a flexible and scalable way [1] [2]. Its key technology is virtualization, which enables service isolation by running them in abstractions called Virtual Machines (VMs) [3]. Nonetheless, virtualization introduces certain overhead due to hypervisors, which manage VM activities and CPU scheduling among virtual machines [4]. As an alternative to traditional virtualization, OS-level virtualization tools like Docker [5] and Podman offer similar functionalities, all while mitigating some of the overhead drawbacks.

Although many of its benefits are already known, the possible bottlenecks of OS-level virtualization tools still need to be fully understood [6]. So, in this paper, we conducted a performance evaluation of two well-known OS-level virtualization tools (Docker and Podman) from a system management perspective. The evaluation centers on measuring the execution time of key management-related variables, including : image pull time, container instantiation, container termination, container removal, and image removal. The study encompasses three test scenarios: downloading the image from a remote server, downloading the image on a local server, and working with an image pre-installed on the machine. Various image sizes are considered, ranging from 500MB to 3GB. The analysis provides insights into the average times of these variables and employs a Design of Experiments (DoE) approach for a comprehensive assessment of the data.

The remainder of this article is structured as follows: Section II presents some interesting related works, while Section III outlines the experimental methodology used. The results are presented in Section IV, and the final considerations are presented in Section V.

II. Related Works

The performance evaluation of container technology has been a subject of extensive study, with notable works like [7] and [8] demonstrating the superiority of containers over virtual machines, especially in scenarios involving concurrent execution of multiple abstractions on the same resource [5]. These studies have laid the foundation for understanding the benefits of containerization in various application domains, including big data processing and general I/O-intensive workloads [9] [10], as well as high-performance embedded systems [11] [12].

While these investigations have contributed significantly to the field, a critical aspect has often gone unnoticed: the impact of container image size on application performance. Notably, the literature lacks substantial exploration in this area, with existing studies typically focusing on the application-level performance of containers [13]. The role of image size, despite its potential significance, has received limited attention.

Prior studies, such as the work by [14] [15], have evaluated the performance of different image types, but in the context of traditional virtualization and didn't explicitly consider the nuances of container image size. In a related study, [12] assessed the performance of two container technologies in I/O-intensive applications, offering an indirect indication that the size of files stored within container images may affect performance. However, the direct influence of image size on container administration tasks remains unexplored in the current literature.

Additionally, prior research by [16] conducted an analysis of 200,000 Docker Hub images with varying image configurations to assess the impact of these features on container startup times. Notably, one of the parameters under consideration was the image size. The findings of this study concluded that no single dominant configuration feature exclusively determines a container's startup time on Docker platform. However, it's important to note that this study did not take into account different container platforms. Consequently, the conclusions

drawn from their research can not be generalized to containers in a broader context.

What sets our work apart is our focus on the often-overlooked factor of image size and its potential impact on the performance of container administration-related activities. Furthermore, our research aims to investigate whether the influence of image size varies noticeably depending on the type of container used, providing a more nuanced understanding of this relationship.

III. Experimental Methodology

In this section, we present the experimental methodology used in this work. We start by outlining the goals of the study, followed by some general considerations about the experimental design required to achieve these objectives. Finally, we provide detailed information on the experiments conducted from an operational perspective and describe the hardware and software infrastructure used.

A. Goal

The main goal of this work is to verify whether the size of images used in containers affects the performance of activities related to container management and whether such variation depends on the OS-level virtualization tool used.

B. Design Of Experiments - DoE

In a designed experiment, intentional alterations are made to input variables to observe their impact on the output [17]. This involves categorizing factors as independent variables and responses as dependent variables. It necessitates thoughtful factor selection and a fitting experimental plan to assess resulting effects, like the comprehensive full factorial plan, which examines all independent variable levels.

We have chosen to monitor the following dependent variables: container image download time, container instantiating time, container stop time, and container and image removal times. These metrics are vital when considering the long-term use of containers. For short-term services, processing efficiency may not pose a significant concern, but when a platform remains active for over a year, unoptimized metrics can result in substantial infrastructure costs. The service (Docker or Podman), test scenarios, and image sizes serve as the independent variables.

Optimizing the dependent variables can be achieved by adjusting the independent variables. A response optimization model is commonly employed to identify the optimal variable combination. Desirability functions, a multivariate analysis technique, are used to simultaneously evaluate and optimize multiple response variables in an experiment or process. This approach is often applied in experiments where the goal is to determine the optimal input variable configuration to maximize or minimize dependent variables.

The desirability function is defined as a transformation that converts a response (dependent) variable y into a desirability value d ranging from 0 (undesirable) to 1 (highly desirable), i.e. $0 \leq d_i \leq 1$. There are many desirability functions that can be used, such as the linear function or the quadratic function. In this work, we have used the Desirability function offered by Minitab software, which is:

$$\begin{array}{ll} d_i = 0 & \hat{y}_i > \mathrm{U}_i \\ d_i = \left(\left(\mathrm{U}_i - \hat{y}_i\right) / \left(\mathrm{U}_i - \mathrm{T}_i\right)\right)^{r_i} & \mathrm{T}_i \leq \hat{y}_i \leq \mathrm{U}_i \\ d_i = 1 & \hat{y}_i < \mathrm{T}_i \end{array}$$

where, $\hat{y}_i$ predicted value of the i^a response; T_i target value of the i^{a} response; U_i highest acceptable value for the i^a response; d_i *desirability* for the i^a response; D *desirability* composed [18].

C. Experiments Performed

To achieve the pursuited goals, the case studies are based on the virtualization of open-source OS-level virtualization tools used in private and hybrid clouds. The initial stage of the methodology was based on a study to understand how the Docker and Podman software works and in what context these tools are used. Next, the test environment, monitoring strategy and workloads to be used were defined.

To this end, the solution chosen for monitoring software time metrics was to develop pure Shell scripts, without adding extra tools, since their main function is to automate repetitive and complex tasks [19]. Therefore, this study considers three different scenarios for the software mentioned: downloading the image from a remote server, downloading the image on a local server and with the image already installed on the machine. To ensure a realistic experiment, a workload was used that behaves similarly to a real scenario, which seeks to use the machine's computing resources at high load.

Figure 1 illustrates all the test scenarios developed, considering the types (Normal, RMI, Local RMI), the services studied (Docker and Podman) and the different image sizes used in the analysis (500MB to 3GB). RMI refers to the scenario in which the images and containers are removed at the end of each run, while in the Normal scenario only the containers are removed each cycle, and Local refers to downloading the images over a local network.

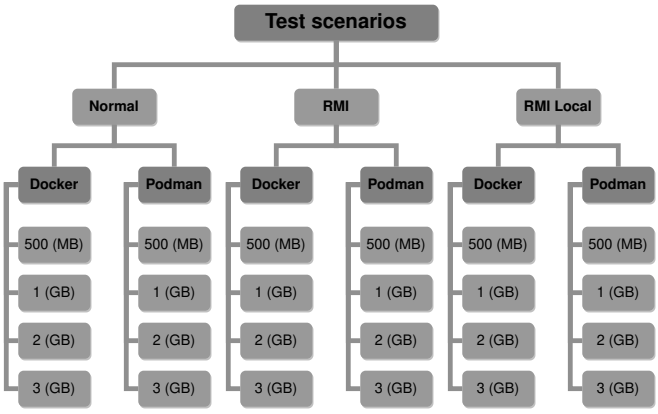


Fig. 1. Illustration containing all the test scenarios carried out

D. Testbed environment

The test environment consists of a single HP Compaq 6005 machine, with a 2.8 GHz AMD Athlon II X2 processor, 8 GB of main memory and an additional 1 GB of *swap*. The operating system used was Debian 11 Bullseye (*kernel* 5.10.0-19-amd64) with no graphical interface with only the standard system utilities and a *ssh* server installed. Docker and Podman software were used in versions 23.0.1 and 3.0.1, respectively. For the images, in order to offer a broad view of the testing possibilities, 4 main sizes were chosen: 500MB, 1GB, 2GB and 3GB. Three different scenarios were taken into account: testing with an image already installed in the environment (Normal), testing with an image pull from a remote server (RMI), testing with an image pull from a server on the local network (Local).

In addition, all the images were provided directly from the Docker servers and configured in advance. The images have a nginx server in version 1.23.3 and a file with random bytes generated by the command *dd*. Each file had its size calculated to take into account the size of the nginx image and the Docker compression algorithms when doing *push* to Docker Hub. The images are available on Docker Hub with the following identifiers: pedrmelo/software-aging:500mb, pedrmelo/software-aging:1gb, pedrmelo/software-aging:2gb, pedrmelo/software-aging:3gb. Finally, for the Local tests, the image server configuration used was a Positivo MOBO 5950 notebook with a 1.6GHZ Intel Atom N2600 and 2GB of RAM and the OpenMediaVault system installed, a system designed to be used as a network storage platform (NAS) to manage and share files. At the end of each test, any resident images and containers were removed and the test machine rebooted.

Figure 2 shows the script execution flow used during the execution of experiments involving the *pull* of a server's image, which consisted of: performing the *pull* of the image, instantiating the container, stopping it, and removing the container and image. In order to ensure data integrity and that the previous experiment didn't affect the next one, a 30 seconds wait time was added after the container was instantiated and another 60 seconds wait time after the image was removed. Finally, everything is done in a conditional loop with a maximum of 100 interactions (n == 100). The error handling presented is due to the fact that there is a *pull* limit on the Docker servers and also to get around possible infrastructure problems related to the Internet used.

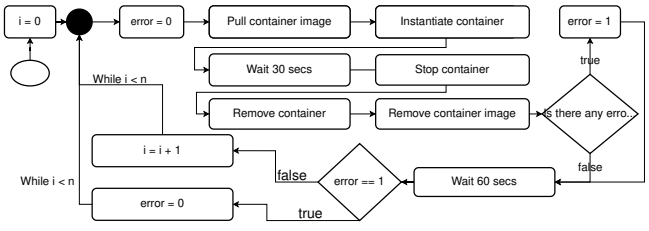


Fig. 2. Script flow

For the scenarios in which the *pull* was not performed directly from a server, the *script* flow shown in Figure 2 was slightly altered, with neither the *pull* of the image nor its removal occurring within the main iterative *loop*, so the *pull* and image removal times were not measured for this scenario. Wait times were also reduced to 10 and 20 seconds respectively.

IV. Experimental Results

This section presents and analyses the results obtained on this study. The analysis of the results is divided into subsections to facilitate its comprehension.

A. Experimental Results

The analysis of the data obtained was grouped according to the three types of test carried out. In addition, each graph was divided into four sectors representing the sizes of the images used, each related to the average times of each variable studied in relation to the size of the image used. To compare the times, the Docker data is in a dark shade and the Podman data is in a light shade.

Figure 3 shows the results of the experiments in Scenario #1, in which the containers were instantiated with images already present on the experiment machine. There was no *pull* image from a server, so the time taken to remove the images is zero. The results indicate that the stop time, container removal and image removal times of the two tools were very similar. The biggest difference was observed with regard to Docker's instantiating time, which is relatively longer than Podman's for the 500MB, 1GB and 2GB image scenarios, where Docker's average instantiating time was almost twice as long as Podman's with the 2GB image (0.7s for Dodman and 0.35s for Podman). In addition, most scenarios showed better results with Podman for the average time to *pull* and when removing the containers.

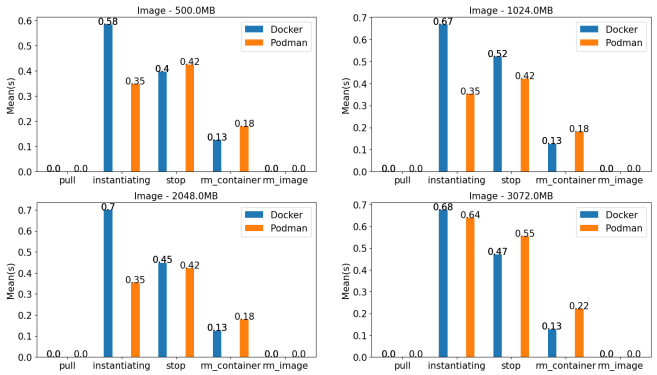


Fig. 3. Normal Scenario Results #1

In the Scenario #2, images are tested doing a *pull* image from a remote server. For this case, we used Docker Hub, since the same images can be used for both Docker and Podman. In Figure 4, we can see that the biggest difference in time was found in the *pull* time between the two tools, with 229.78s and 267.65s for Docker and Podman, respectively. In this scenario, Docker showed better results for most of the variables, with

the exception of container instantiating using 2GB and 3GB images, with a value of 3.52s for the 2GB image and 28.48s for the 3GB image, while Podman showed 0.72s with a 2GB image and 5.74s with a 3GB image in relation to instantiating time. Thus, Docker ended up showing a difference more than 4 times greater than Podman in the container instantiating variable related to the 2GB and 3GB images. In the other variables, Docker showed better results in all values, being overall twice as fast as Podman.

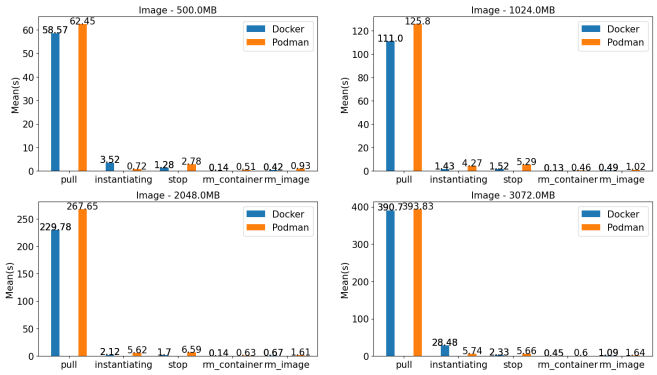


Fig. 4. RMI Scenario Results #2

Finally, Scenario #3 tests images by doing *pull* from a local server. In Figure 5, we see a pattern similar to the one found in the analysis of the previous graph: Podman had a better performance for the instantiating time of a 500MB container when compared to Docker, and Docker did better than Podman with regard to the other variables, even if sometimes showing a slight difference. There was also the same pattern as in the previous analysis with the variable of instantiating with a 3GB image, Podman had a time almost 7 times better than Docker (4.08s and 28.96s respectively).

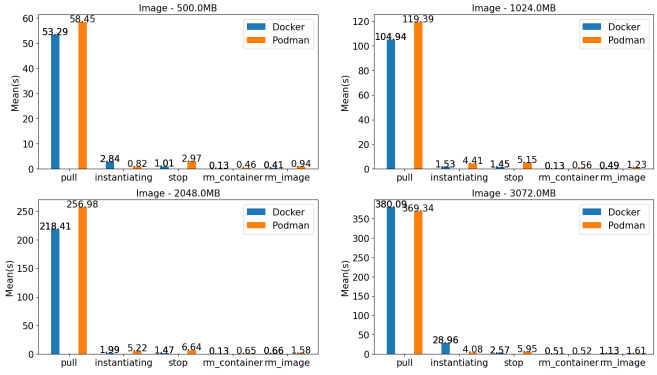


Fig. 5. RMI Local Scenario Results #3

Table I lists all the average times of the variables studied in relation to the test scenarios.

B. DoE analysis

Analyzing the Pareto chart for *pull*, in Figure 6, we can see that all the factors and their combinations up to the third interaction are important for the DoE analysis. In the graph, we have the vertical reference line as a value of 1.96, and all the vertical bars exceed this line, thus indicating that for all the responses in our model, the effects are statistically significant. This pattern was confirmed for all the other dependent variables and, for simplicity, only the Pareto chart for *pull* has been included.

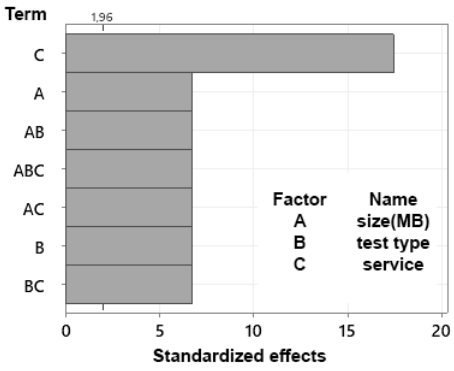


Fig. 6. Standardized Effects Pareto Chart - *pull*

The following DOE analysis was divided into two parts, since in the previous data analysis Podman showed a shorter time when raising smaller containers (*pull* time and instantiation) and Docker in the downtime of the services for the other containers (downtime, container removal and image removal). Interaction graphs were used for the independent variables: image size (size); test type (type), with (1) the test adopted as Normal, (2) the RMI test and (3) the Local RMI test; service (service) with (1) for Docker and (2) for Podman. For the first square of each graph, the line with circles represents the Normal scenario, the one with squares the RMI scenario and the one with triangles the Local RMI scenario. For the remaining two, the circled line indicates the Docker container virtualizer and the squared line Podman.

Figure 7(a) represents the averages of *pull* time and instantiating for each image size considering the 3 different test scenarios. Each level of the independent variables is shown on the horizontal axis and the averages are shown on the vertical axis. By analyzing this graph, we can see some similar trends for the 3 factors. Both the size * type, size * service and type * service graphs did not cross at any point, so these factors did not affect the dependent variables.

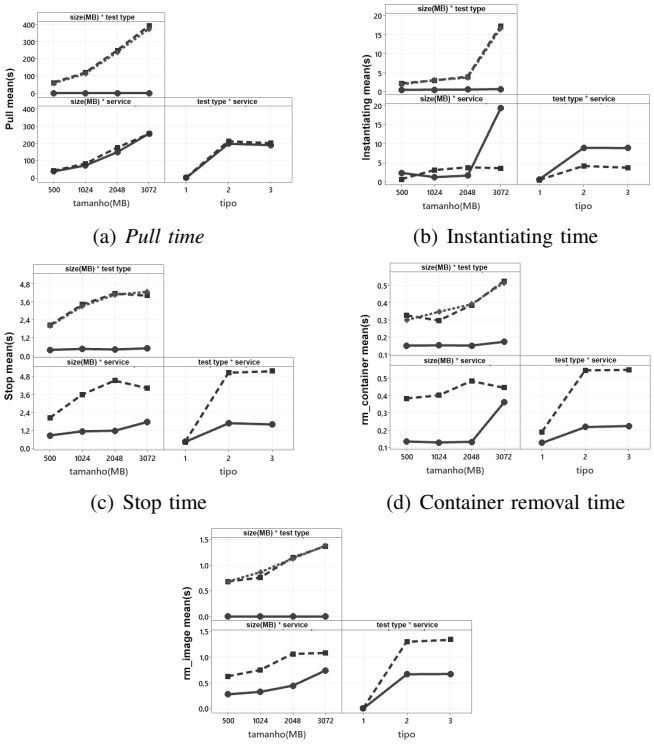
In addition, the 7(b) instantiating graph followed a similar pattern to the *pull* graph, however the graph between size * service showed that there is a strong interaction between the service used and the size of the image, since the average instantiating time decreased noticeably when we have Docker and a 3GB MB image when compared to the RMI scenario with an image of the same size.

As for Figures 7(c), 7(d) and 7(e), we noticed that the pattern of interactions we studied in the analyses of *pull* and instantiating was also maintained for the graphs of interactions for stop time, container removal and image removal times.

Software	Test type	Image size (MB)	Pull (s)	Instantiating (s)	Stop (s)	Container removal (s)	Image removal (s)
Docker	Normal	500	0.00	0.58	0.40	0.13	0.00
Podman	Normal	500	0.00	0.35	0.42	0.18	0.00
Docker	Normal	1024	0.00	0.67	0.52	0.13	0.00
Podman	Normal	1024	0.00	0.35	0.42	0.18	0.00
Docker	Normal	2048	0.00	0.70	0.45	0.13	0.00
Podman	Normal	2048	0.00	0.35	0.42	0.18	0.00
Docker	Normal	3072	0.00	0.68	0.47	0.13	0.00
Podman	Normal	3072	0.00	0.64	0.55	0.22	0.00
Docker	RMI	500	58.57	3.52	1.28	0.14	0.42
Podman	RMI	500	62.45	0.72	2.78	0.51	0.93
Docker	RMI	1024	111.00	1.43	1.52	0.13	0.49
Podman	RMI	1024	125.80	4.27	5.29	0.46	1.02
Docker	RMI	2048	229.78	2.12	1.70	0.14	0.67
Podman	RMI	2048	267.65	5.62	6.59	0.63	1.61
Docker	RMI	3072	390.70	28.48	2.33	0.45	1.09
Podman	RMI	3072	393.83	5.74	5.66	0.60	1.64
Docker	RMI Local	500	53.29	2.84	1.01	0.13	0.41
Podman	RMI Local	500	58.45	0.82	2.97	0.46	0.94
Docker	RMI Local	1024	104.94	1.53	1.45	0.13	0.49
Podman	RMI Local	1024	119.39	4.41	5.15	0.56	1.23
Docker	RMI Local	2048	218.41	1.99	1.47	0.13	0.66
Podman	RMI Local	2048	256.98	5.22	6.64	0.65	1.58
Docker	RMI Local	3072	380.09	28.96	2.57	0.51	1.13
Podman	RMI Local	3072	369.34	4.08	5.95	0.52	1.61

TABLE I

TABLE WITH THE AVERAGE TIMES FOR EACH FACTOR STUDIED AND THEIR TEST SCENARIOS



(a) *Pull time*

(b) Instantiating time

(c) Stop time

(d) Container removal time

(e) Image removal time

Fig. 7. Interaction graph

We can see that the lines remained parallel or very close, not crossing at any time, indicating that there are no strong interactions between the independent variables. On the other hand, only one interaction escaped this analysis: we noticed that in Figure 7(d) the size * type graph showed a strong interaction. Starting from the 500 MB image in the Local RMI scenario, the average time increased for all images. Furthermore, in the RMI scenario there was also an increase, but only from the 1GB image onwards, indicating that the relationship between the size of the image and the type of test used directly influences the container removal time up to these levels. All the other interactions in the 7(c) and 7(e) graphs maintained the same pattern, not showing strong interactions between the dependent variables.

C. Response optimization

The response optimization used was based on the method of *desirability* functions, specifically for the combination of responses in a single *desirability D* by aggregating the geometric mean, always seeking to minimize the dependent variables:

1) For container initialization (*pull* and instantiating - Tables II and III):

Response	Goal	Lower limit(s)	Target(s)	Higher limit(s)
instantiating	Minimize	-	0.3161	49.179
pull	Minimize	-	0.0000	484.481

TABLE II

VARIABLES USED FOR THE MINIMIZATION OF *pull* AND INSTANTIATING

Solution	size(MB)	type	service	Composite Desirability
1	500	1	2	0.999680

TABLE III

RESPONSE TABLE FOR THE MINIMIZATION OF *pull* AND INSTANTIATING

2) For stopping and removing containers times (stop, container removal and image removal - Tables IV and V):

Response	Goal	Lower limit(s)	Target(s)	Higher limit(s)
rm_image	Minimize	-	0.0000	4.4874
rm_container	Minimize	-	0.1096	2.5108
stop	Minimize	-	0.3444	22.4624

TABLE IV
VARIABLES USED FOR STOP TIME, CONTAINER REMOVAL AND IMAGE REMOVAL TIMES MINIMIZATION

Solution	size(MB)	type	service	Composite Desirability
1	500	1	1	0.997047

TABLE V
RESPONSE TABLE FOR STOP TIME, CONTAINER REMOVAL AND IMAGE REMOVAL TIMES MINIMIZATION

3) For all the dependent variables (Tables VI and VII):

Response	Goal	Lower limit(s)	Target(s)	Higher limit(s)
rm_image	Minimize	-	0.0000	4.4874
rm_container	Minimize	-	0.1096	2.5108
stop	Minimize	-	0.3444	22.4624
instantiating	Minimize	-	0.3161	49.179
pull	Minimize	-	0.0000	484.481

TABLE VI
VARIABLES USED TO MINIMIZE THE DEPENDENT VARIABLES

Solution	size(MB)	type	service	Composite Desirability
1	500	1	1	0.997127

TABLE VII
RESPONSE TABLE FOR THE GENERAL MINIMIZATION OF THE EXPERIMENTS

The optimization of the experiments in Table III showed that the best configuration for starting the container services is the 500MB image with the Normal test scenario and using Podman as the service. On the other hand, with regard to stopping and removing the services offered by the containers, the result of Table V, the best configuration is using an image of size 500 MB in the Normal test scenario and using Docker as the service. Overall, Table V gave us the complete result for all the independent variables, showing that the best configuration for minimizing the dependent variables is the 500 MB image, using the Normal scenario and using Docker as the container service.

V. FINAL REMARKS

In this study, we compared Docker and Podman in container management performance. Docker outperformed Podman in most metrics (stop time, container and image removal) across various scenarios. However, Podman excelled in two aspects (pull and instantiation times) for smaller images. Generally, Podman is ideal for rapidly containerizing small applications, while Docker is more efficient for larger containers.

For future work, we plan to expand our research to include other container solutions like LXC/LXD and OpenVZ. We aim to analyze aspects such as CPU, memory, and network usage to assess performance and efficiency in different scenarios, helping identify the best solutions for specific use cases.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, Rep. UCB/EECS, Tech. Rep., 2009.

[2] D. Beserra, S. Souto, J. Andrade, and A. de Araujo, "Comparativo de desempenho de um cluster virtualizado em relação a um cluster convencional sob condições equipotentes," in *IX Workshop Em Clouds, Grids e Aplicacoes-SBRC 2011*, 2011.

[3] D. Beserra, M. Espie, L. Tomasimo, H. de Poncins, H. Lacombe, T. Vondracek, and J. Araujo, "Could the topology of virtual processors affect the performance of a bsd-family os running in a vm?" in *2023 18th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2023, pp. 1–6.

[4] L. Chen, S. Patel, H. Shen, and Z. Zhou, "Profiling and understanding virtualization overhead in cloud," in *2015 44th International Conference on Parallel Processing*, 2015, pp. 31–40.

[5] D. Beserra, E. D. Moreno, P. T. Endo, J. Barreto, S. Fernandes, and D. Sadok, "Performance analysis of linux containers for high performance computing applications," *International Journal of Grid and Utility Computing*, vol. 8, no. 4, pp. 321–329, 2017.

[6] S. Ris, J. Araujo, and D. Beserra, "A systemic mapping of methods and tools for performance analysis of data streaming with containerized microservices architecture," in *2023 18th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2023, pp. 1–6.

[7] D. Beserra, E. D. Moreno, P. T. Endo, J. Barreto, D. Sadok, and S. Fernandes, "Performance analysis of lxc for hpc environments," in *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*. IEEE, 2015, pp. 358–363.

[8] R. Yadav, E. Sousa, and G. Callou, "Performance comparison between virtual machines and docker containers," *IEEE Latin America Transactions*, vol. 16, no. 8, pp. 2282–2288, 2018.

[9] N. Naik, "Docker container-based big data processing system in multiple clouds for everyone," in *2017 IEEE International Systems Engineering Symposium (ISSE)*. IEEE, 2017, pp. 1–7.

[10] D. Beserra, E. D. Moreno, P. T. Endo, and J. Barreto, "Performance evaluation of a lightweight virtualization solution for hpc i/o scenarios," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2016, pp. 004 681–004 686.

[11] D. Beserra, M. K. Pinheiro, C. Souveyet, L. A. Steffenel, and E. D. Moreno, "Performance evaluation of os-level virtualization solutions for hpc purposes on soc-based systems," in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 2017, pp. 363–370.

[12] D. Beserra, M. Pinheiro, C. Souveyet, L. Steffenel, and E. Moreno, "Comparing the performance of os-level virtualization tools in soc-based systems: The case of i/o-bound applications," in *2017 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2017, pp. 627–632.

[13] B. orević, V. Timčenko, M. Lazić, and N. Davidović, "Performance comparison of docker and podman container-based virtualization," in *2022 21st International Symposium INFOTEH-JAHORINA (INFOTEH)*. IEEE, 2022, pp. 1–6.

[14] D. Beserra, A. Borba, S. Souto, M. Andrade, and A. Araujo, "Desempenho de ferramentas de virtualização na implementação de clusters beowulf virtualizados em hospedeiros windows," in *X Workshop em Clouds, Grids e Aplicações-SBRC*, 2012, pp. 86–95.

[15] E. Sousa, P. Maciel, E. Medeiros, D. Souza, F. Lins, and E. Tavares, "Evaluating eucalyptus virtual machine instance types: A study considering distinct workload demand," *CLOUD COMPUTING*, pp. 130–135, 2012.

[16] M. Straesser, A. Bauer, R. Leppich, N. Herbst, K. Chard, I. Foster, and S. Kounev, "An empirical study of container image configurations and their impact on start times," in *2023 23rd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid). io*, 2023.

[17] J. Antony, *Design of experiments for engineers and scientists*. Elsevier, 2014.

[18] L. Minitab, "Response optmizer," url: https://support.minitab.com/pt-br/minitab/21/help-and-how-to/statistical-modeling/using-fitted-models/how-to/response-optimizer/methods-and-formulas/individual-desirabilities/, Mar. 2023.

[19] R. K. Michael, *Mastering unix shell scripting: Bash, Bourne, and Korn shell scripting for programmers, system administrators, and unix gurus*, 2nd ed. Wiley Publishing, 2008.