International Journal of Electronics and Communications Systems
Volume 3, Issue 2, 57 - 67.
ISSN: 2798-2610
http://ejournal.radenintan.ac.id/index.php/IJECS/index
DOI: 10.24042/ijecs.v3i2.19860

Analysis and Implementation of Comparison Between Podman and Docker in Container Management

Husain
Fakultas Teknik, Universitas Bumigora, Mataram, INDONESIA

Khairan Marzuki
Fakultas Teknik, Universitas Bumigora, Mataram, INDONESIA

Christopher Michael lauw
Fakultas Teknik, Universitas Bumigora, Mataram, INDONESIA

Lalu Zazuli Azhar Mardedi
Fakultas Teknik, Universitas Bumigora, Mataram, INDONESIA

Article Info

Article history:

Received: October 12, 2023
Revised: November 14, 2023
Accepted: December 29, 2023

Keywords:

Computer Network;
Container;
Docker;
Podman

Abstract

The increasing use of the internet makes the implementation process more accessible, but the problem is difficult to manage network management, with the emergence of container technologies such as Docker and Podman as efficient application management solutions. This research compares Docker and Podman regarding container management using the Network Development Life Cycle (NDLC) methodology. This study evaluates three parameters: accessing the Fedora project registry, handling images or ISOs, and user access in containers. The results show that Podman performs better regarding registry access, is slightly faster with images, and offers faster user creation. Overall, the study concludes that Podman is superior, demonstrating compatibility with Docker, and proving its efficacy in container management.

To cite this article: H. Husain, K. Marzuki, C. M. Lauw, and L. Z. A. Mardedi. "Analysis and Implementation of Comparison Between Podman and Docker in Container Management, " Int. J. Electron. Commun. Syst., vol. 3, no. 2, pp. 57-67, 2023.

INTRODUCTION

In Indonesia's rapidly evolving landscape of internet and computing technology, a significant shift is observed, with internet users reaching 205 million as of January 2022, marking a remarkable 73.7% increase, as highlighted by DataIndonesia.id [1]. This exponential growth has notably eased web application access and significantly streamlined deployment processes [2], [3]. Deployment, a critical phase involving the distribution of production applications, encompasses various challenges such as installation, server adjustments, and managing dependencies like operating systems, web servers, libraries, and databases [4], [5]. Traditional manual deployment methods pose considerable challenges, prompting the need for more efficient solutions [6].

In this context, container technology has emerged as a game-changer, offering lightweight isolation at the operating system level and significantly accelerating deployment processes [7]. Among the various container management solutions, Podman and Docker stand out. Docker, based on a client-server model, hinges on a daemon for communication, making its performance reliant on the presence of interdependent components. Conversely, Podman employs a novel rootless concept, offering compatibility without necessitating root access, thus presenting a unique approach to the container technology [8], [9].

This research aims to delve deeper into these two prominent container management tools by conducting a comprehensive analysis and implementation comparison between Podman and Docker [9], [10]. What sets this research apart is its focus on exploring the

• **Corresponding author:**
Universitas Bumigora, Mataram, INDONESIA. ✉ khairan.marzuki@universitasbumigora.ac.id

distinct operational frameworks of Podman and Docker, particularly examining how Podman's rootless approach contrasts with Docker's daemon-dependent architecture. This exploration into the operational nuances and performance metrics of Podman and Docker is at the forefront of current research in container management, marking a significant contribution to the field.

The methodology involves managing containers in both Podman and Docker using VMware Workstation, with CentOS as the host operating system [9], [11]. The analysis revolves around three key parameters: Registry redhat.com, image management, and user access within the container services [12], [13], [14]. The research meticulously examines these parameters, offering insights into the effectiveness and efficiency of both Podman and Docker in real-world scenarios.

This research takes a unique angle by undertaking an in-depth comparative analysis of Podman and Docker, focusing on critical aspects such as registry access, image handling, and user interaction within containers. These facets have not been thoroughly investigated in prior studies, making this exploration a significant stride forward in container management research. The findings are poised to illuminate which container management tool excels in performance, offering invaluable insights for those embarking on Kubernetes. Additionally, this study contributes substantially to understanding Podman's interoperability with Docker, showcasing the seamless application of Docker commands and images in Podman containers. This aspect of the research enriches the overall knowledge and understanding of the evolving domain of container management technologies.

METHOD

In this study, the chosen methodology is the Network Development Life Cycle (NDLC), a structured yet flexible approach to network development. NDLC is renowned for its adaptability and ease of development, making it particularly suitable for projects requiring iterative refinement and testing [15], [16]. While NDLC typically encompasses six stages, this research strategically focuses on three specific stages: Analysis, Design, and Prototyping Simulation.

The decision to concentrate on these three stages is grounded in their relevance and efficacy for the study's objectives. The Analysis stage is crucial for understanding the existing environment and establishing requirements for Podman and Docker in container management. The Design stage allows for conceptualizing how these container technologies will be compared and evaluated. Lastly, the Prototyping Simulation stage is instrumental in practically implementing the comparison, enabling hands-on examination, and testing of the functionalities and performance of Podman and Docker. This selective application of the NDLC stages ensures a targeted and efficient research process, aligning with the study's focus on comparing and implementing container management solutions.

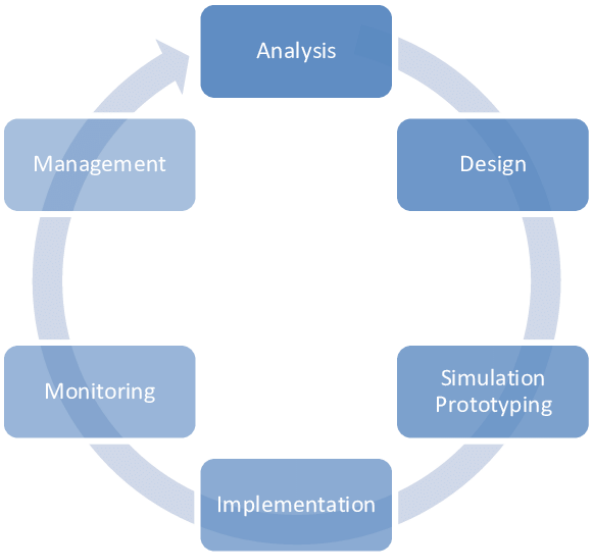

Figure 1 Network Development Life Cycle Methodology

At this stage, the author collected data using a literature study. Namely, the author read scientific articles, journals, and theses to obtain information on the comparative analysis and implementation of Docker and Podman in Container Management. This stage consists of 2 (two) parts, namely data collection and data analysis. Where data and theories regarding the comparison of Docker and Podman are collected, then the data that has been collected is analyzed and then processed so that the author focuses on the analysis and implementation of the comparison of Podman and Docker in container management.

Design

This stage consists of 6 (six) stages, namely designing the components that make Docker, designing the components that make up Podman preparing hardware and software, and IP addressing as support for the testing process, registering on the official Docker Hub and Fedoraproject sites.

- Docker Building Components

The components contained in Docker are as follows [17], [18]:

a) Docker Daemon is a service that runs on the host Operating System (OS). The function of the docker daemon is to build, distribute, and run docker containers.
b) Docker Client is a command line, command for running docker containers, for example, create containers, start/stop containers, delete (destroy), and so on in the docker daemon.
c) Docker Images is A Docker component in the form of read-only templates. Templates are an OS or ISO that has been installed. Docker images work to create a docker container, only one docker image, can make lots of docker containers.
d) A Docker Container is an image that can packaged, and read-write containers run on images. A docker container can also be considered a folder, where this docker container is created using a docker container. Every time a Docker container is saved, a new layer will be formed above the Docker image or base image.
e) Docker Registry redhat.com is a distribution repository for a collection of private and public docker images that can be accessed via Docker Hub.

- Podman Component Design

Podman is an open-source program available on all Linux platforms and GitHub [9], [19]. Podman is a tool for managing containers that aims to be an alternative to Docker. Podman is a daemon-less container program for developing, managing, and running the Open Container Initiative (OCI) [20]. The component image below shows that the redhat.com images registry, container, image storage, and kernel communicate directly with the Podman container via runC. As for components, the compiler has a Podman container as follows

- Images registry redhat.com. According to Microsoft.com, it is a service that saves all images and distributes container images and dependencies related to the image registry redhat.com, often called a library.
- Containers: In the inner containers case, Podman is a container image running or running by the user.
- Images are like the ISO or operating system that will be installed and the images in the container. Where is the image that will run the container?
- The kernel is the core component of an operating system. An operating system works with the kernel, which is responsible for managing data processing on each computer device.

Simulation Prototyping At this stage, configuration installation, IP addressing, internet connection testing, account registration, test results, test scenarios, and test results analysis on each Docker and Podman container service will be carried out [21], [22]. Installation and Configuration The installation and configuration stages are divided into several stages: installation and configuration on CentOS 8 Stream, installation and configuration of Mikrotik CHR, and installation on Docker containers and Podman containers [23], [24], [25].

- Installation

Installation on VMware Work Station uses 3 (three) Virtual machines (VM). VM1 (one) is installed on the first VM with the CentOS 8 Stream host operating system, and Internet Protocol (IP) addressing configuration on VMnet 1, located in the Windows adapter settings. VM1 with IP address 192.168.169.170. Netmask 24, and gateway 192.168.169.254. VM2 (two) is installed by installing the second VM with the CentOS8 Stream host operating system, and configuration. The VM2 IP address it has is 192.168.169.175. Netmask 24 and gateway 192.168.169.254.

- Mikrotik Installation and Configuration

VM3 (three) installed Mikrotik CHR 6.49.6. static IP addressing with the IP gateway used on the Host Operating System 192.168.169.254/24. On the 3 (three) VMware workstations that have successfully installed CentOS, the Docker container and

Podman container are then installed on VM2 and VM1.

- IP Addressing

This stage will carry out Internet Protocol (IP) addressing on the virtual network in the Windows adapter settings. The configuration is on VMnet1.

-Test Connection

At this stage, the author tested the internet network connection from VM1 (Podman installation), VM2 (docker installation), and VM3 (Mikrotik) by accessing the google.com site service.

Account Registration

- Docker Hub Registration

A previous registration process is carried out to create a Docker Hub account so that we can withdraw container images.

- Register Fedoraproject.org

A previous registration process is carried out to create a fedoraproject.org account so that we can withdraw container images.

The test scenario stages consist of 3 (three) parameters, namely accessing the redhat.com registry, images, and users. The following is the flow of this research trial: Access the redhat.com registry. The image below shows that the Docker and Podman containers will access the redhat.com registry. docker.io and fedoraproject.org to use the same command, namely "login registry name redhat.com".

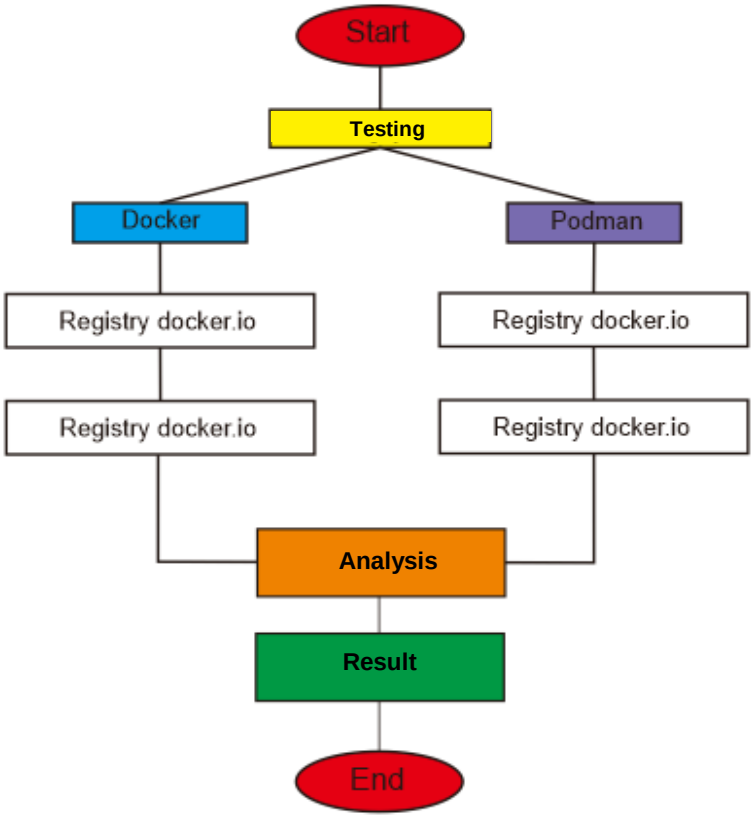

Figure 2 Rules of registry access

The image below shows that 5 trials were carried out using the image parameters pull, run, stop, verify, delete container, and delete images. The images used in the study are Hello-World, Nginx, Centos, Tomcat, and Fedora. A pull test was carried out withdrawal of images with these five images via docker containers and Podman. Then, the analysis and testing with run images Nginx using the name testnginx2 in docker and Podman containers. No use name, test next, namely stop or stop the Nginx containers running on the docker and Podman container services. Trials _ verify results from the existing container deleted via docker and Podman containers. Trials, Delete all containers, both medium ones going on, and what has been stopped (exited) on docker and Podman containers. The final test is to delete all the images downloaded in the docker and Podman containers.

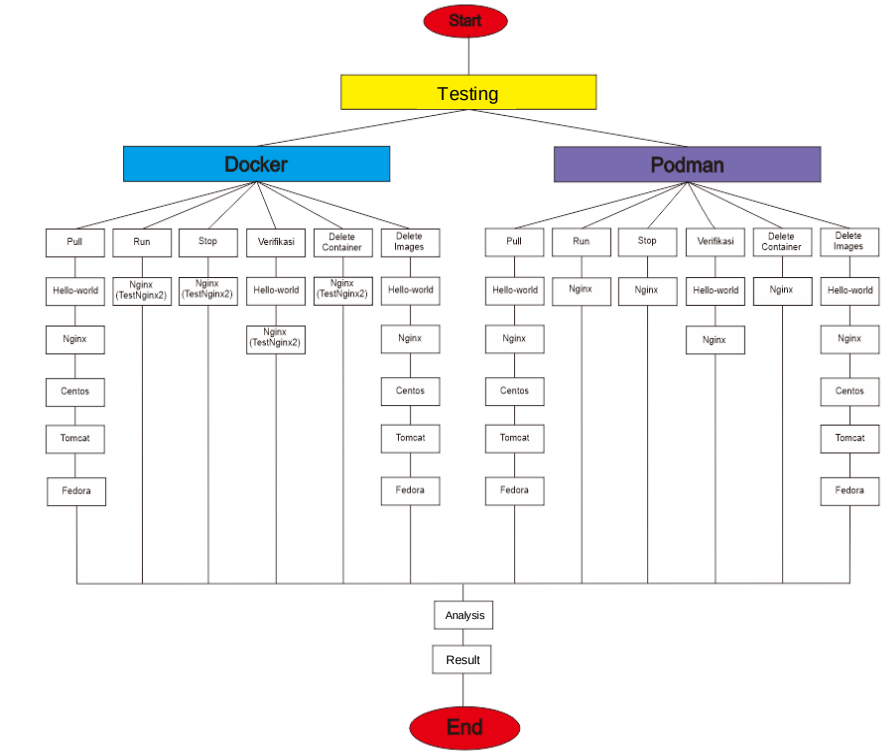

Figure 3 Rules of Image

Explanation Test flow The Figure above shows that testing was carried out with access to docker and Podman container services with existing users determined in the picture using user niki2 and Podman use iki.

RESULTS AND DISCUSSION

In this research, an experiment will be conducted to access the registry services of fedoraproject.org and docker.io through both Docker and Podman containers. The results indicate that both are capable of accessing the respective registries. However, a difference is observed in the Docker container, where a warning is issued, stating that the Docker daemon does not encrypt the password. This is evident in the Figure 4.

```
[root@closestack ~]# docker login registry.fedoraproject.org
Username: niki2309
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[root@closestack ~]#
```

Figure 4. Results Access Registry Service

Table 1 Results in a Registry Access

Containers	Registry	Speed	Response time
Docker	docker.io	9.24 mbps	22.75 sec
	Fedoraproject.org	9.24 mbps	20.23 sec
Podman	docker.io	Explanation	19.79 Sec
	Fedoraproject.org	Explanation	20.20 sec

This research involves accessing the docker.io and fedoraproject.org registries through Docker and Podman containers. As indicated in the table above, accessing docker.io with an internet speed of 9.24 Mbps takes slightly longer for Docker, with a time difference of 2.96 seconds slower than Podman.

Accessing the fedoraproject.org registry in both Docker and Podman containers at the same speed requires 20.23 seconds for Docker and 20.20 seconds for the Podman container. The difference between the two containers is only 3 seconds, as Podman defaults with the fedoraproject.org registry, while Docker does not, resulting in a slightly longer time. However, a more exciting aspect of accessing fedoraproject.org on Docker is the WARNING!

Command, indicating that the password is not well-encrypted in the Docker daemon (server). From the analysis of the experiments, it is concluded that the difference in registry access at the same speed is minimal, approximately 0.3%, with Podman being faster by 3 seconds. Therefore, for the registry parameters, Podman performs better. An experiment was conducted on Docker and Podman containers to obtain a comparison of the two containers, as seen in Table 2.

Table 2. Pull Image

Docker		**Podman**	
Image	**Command**	**Image**	**command**
Hello-World	Docker pull images….	Hello-World	Podman pull images….
Centos	Docker pull images….	Centos	Podman pull images….
Tomcat	Docker pull images….	Tomcat	Podman pull images….
Fedora	Docker pull images….	Fedora	Podman pull images….

The analysis results from the table above are from the command side, where the commands used are relatively the same, but this research also conducted research in terms of speed and response time, as seen in Table 3.

Table 2 Pull Image Hello

Containers	**Name**	**Size /kB**	**Internet**	**speed**	**response**	**time**
Docker	Hello-world	13.3	Wifi.id	82.66	08.05 sec	12.30
Podman	Hello-world	19.9	Wifi.id	82.66	08.07 sec	11:52

The table 3, states that the volume of Docker images is smaller than that of Podman, so withdrawing from Podman takes longer. The difference is around 0.02%, so the hello-world docker withdrawal is better.

Table 3 Pull Nginx

Containers	**Name**	**Size /kB**	**Internet**	**speed**	**response**	**time**
Docker	Nginx	142	Wifi.id	142	38.10 minutes	12.37 WITA
Podman	Nginx	146	Wifi.id	66.14	34.30 minutes	12.37 WITA

The table 4, states that the volume of Docker images is smaller than that of Podman, but compared with Podman it has a relatively short speed compared to Docker with a smaller image container capacity. The difference occurs with a difference of 3.20 seconds, so it can be concluded that Podman is better.

Table 4 Pull Centos

Containers	**Name**	**Size /kB**	**Internet**	**speed**	**response**	**time**
Docker	Centos	213	Wifi.id	76.67	30.31 minutes	13.05 WITA
Podman	Centos	215	Wifi.id	76.67	28.40 minutes	13.05 WITA

The table 5, states that the volume of Docker images is smaller than that of Podman, but compared with Podman it has a relatively short speed compared to Docker with a smaller image container capacity. The difference occurs with a difference of 1.91 seconds, so it can be concluded that Podman is better. The difference is close to 2 seconds.

Table 5 Pull Tomcat

Containers	Name	Size /kB	Internet	speed	response	time
Docker	Tomcat	483	smartfren	8.82	1:30: Minute	15.00 WITA
Podman	Tomcat	488	Smartfren	8.82	1:28: 4 Min	15.00 WITA

The table 6, states that the volume of Docker images is smaller than that of Podman. Of all the images used in this research, Tomcat is the image with the largest capacity, so it takes up to hours. The difference is that Podman is around 2 seconds faster. The capacity of Podman ISO (images) is larger, namely 5 Mbps.

Table 6 Pull Fedora

Containers	Name	Size/kB	Internet	speed	response	time
Docker	Fedora	163	smartfren	10.00	15.46 minutes	3.55 WITA
Podman	Fedora	163	Smartfren	10.00	13.36 minutes	3.55 WITA

In this table 7, there is a similarity in image capacity between Podman and Docker. Still, in speed, Podman again gets a shorter speed by a difference of 2.4 seconds, so it is said that Podman is better than Docker. From all the pull tests found, it was concluded that if the registry was the same it would produce images with relatively the same capacity but the speed of Podman was superior by around 0.2%, a slight difference.

The comparison in this research is using names and IDs on containers. The docker container service runs with the names testnginx2 added, while Podman uses neither. In terms of commands, it can be seen in the table 8.

Table 7 Run Image

DOCKER COMMAND	*PODMAN* COMMAND	ANALYSIS
docker run -d –name *testnginx2* -p 80:80 nginx	*Podman run* -d -p 80:80 nginx,	*Docker* uses names and *Podman* does not

In the testing parameters for this running stage, differences were found in Podman, which did not use IDs and names on Nginx containers, so names were given randomly, as seen in Table 9 of container running verification.

Table 8 Run Hello-world

Containers	Name	Size/kB	Internet	speed	response	time
Docker	Hello-world	13.3	smartfren	8.03	07.64 seconds	6.50 WITA
Podman	Hello-world	19.9	Smartfren	8.03	06.3 seconds	7:24 WITA

After the pull test was carried out, it was found that Podman only took 1.61 seconds to run because the capacity of Hello-World was relatively small. So Podman is faster.

Table 9 Run Nginx

Containers	Name	Size/kB	Internet	speed	response	time
Docker	Nginx	142	smartfren	12.55	04.06 minutes	727 WITA
Podman	Nginx	146	Smartfren	12.55	03. 52 minutes	7:27 WITA

After pulling the pull test, it was found that the Podman took only 0.54 seconds. So Podman is better than Docker. Stop This research carried out the process of stopping the Nginx container that was running after testing docker and Podman and found slight differences as seen in the table 11.

Table 10 Stop Image

DOCKER COMMAND	PODMAN COMMAND	ANALYSIS
stop *testnginx2* .	*Podman* stop 16af050565de	Differences in ID and Names
stop *testnginx2* .	*Podman* stop 16af050565de	Differences in ID and Names

The table 11, shows that the difference is only in naming and ID, but if you run Podman and docker the opposite way, for example, docker with ID and Podman with names can run. In the testing process, it can be seen that Docker uses names, and Podman uses IDs. The output produced by both is the same according to the command entered.

Table 11 Stop Image

Containers	Name	Size/kB	Internet	speed	response	time
Docker	Nginx	142	smartfren	12.55	01.90 seconds	07.34 WITA
Podman	Nginx	146	Smartfren	12.55	1.75 seconds minute	7:24 WITA

After the pull test was carried out, it was found that the Podman took only 0.25 seconds. This difference was the slightest difference produced by the Podman.

The test parameters carried out in this research were deleting information or displays from the Nginx container running on the Docker and Podman containers. The comparison can be seen in the table 13.

Table 12 Delete Container

DOCKER COMMAND	PODMAN COMMAND	ANALYSIS
Docker container rm *testnginx2*	*Podman* rm 16af050565de	Differences in ID and Names

In the trial of deleting a running container, it was found that similarities in terms of syntax were the same as using the "rm (remove) command."

Table 13 Delete Container

Containers	**Name**	**Size /kB**	**Internet**	**speed**	**response**	**time**
Docker	Nginx	142	smartfren	12.66	01.45 seconds	07.36 WITA
Podman	Nginx	146	Smartfren	12.66	01.00 seconds minutes	7:37 WITA

After the pull test was carried out, it was found that the Podman took only 0.45 minutes to produce a better Podman. The following parameter is tested to delete all container images that have been previously withdrawn. As seen in the table 15.

Table 14 Delete Image

DOCKER COMMAND	PODMAN COMMAND	ANALYSIS
Docker system prune -a	Podman rmi -a	Differences in commands

Docker uses the system prune command to carry out the process of deleting unused data and objects. Docker cannot run the command "rmi -a" that is, it is asked to use the help option to look for the correct command related to "rmi" because in docker the command cannot be found. As in the image 5.

```
[root@closestack ~]# docker rmi -a
unknown shorthand flag: 'a' in -a
See 'docker rmi --help'.
[root@closestack ~]# _
```

Figure 4 Delete Image

Meanwhile, in Podman, the system prune -a command runs, and the output displayed is the same as that in Docker as seen in Figure 6, namely the command runs. The author can conclude that docker commands can be run on the Podman container.

```
[root@openstack ~]# podman system prune -a
WARNING! This command removes:
        - all stopped containers
        - all networks not used by at least one container
        - all images without at least one container associated with them
        - all build cache

Are you sure you want to continue? [y/N] y
Deleted Images
Total reclaimed space: 0B
[root@openstack ~]#
```

Figure 5 Delete Docker Image

Test parameters try to verify what is done in research. This covers verification, i.e., displays results of image download, results running containers, and web server access, including the Image docker images command. In terms of docker and Podman commands, the command is the same. The only difference is in terms of aspects the container just, but the output is displayed differently where in docker as in the Figure 7.

```
[root@closestack ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
nginx         latest    670dcc86b69d   32 hours ago    142MB
tomcat        latest    451d25ef4583   3 weeks ago     483MB
hello-world   latest    feb5d9fea6a5   10 months ago   13.3kB
centos        latest    5d0da3dc9764   10 months ago   231MB
[root@closestack ~]#
```

Figure 6 Verify on Docker

Docker displays the repository related to the image name while Podman is shown in the image 8.

```
[root@openstack ~]# podman images
REPOSITORY                      TAG       IMAGE ID       CREATED         SIZE
docker.io/library/alpine        latest    d7d3d98c851f   2 days ago      5.82 MB
docker.io/library/nginx         latest    41b0e86104ba   8 days ago      146 MB
docker.io/library/tomcat        latest    451d25ef4583   3 weeks ago     488 MB
registry.fedoraproject.org/cheese latest  54d257998cca   7 months ago    19.7 MB
docker.io/library/hello-world   latest    feb5d9fea6a5   10 months ago   19.9 kB
registry.centos.org/centos      latest    2f3766df23b6   19 months ago   217 MB
[root@openstack ~]# _
```

Figure 7 Verification on Podman

Podman displays the Name from the container images at a time from the registry source redhat.com. Which has been downloaded, so it is possible to do a downloading return.

The test parameters carried out in this research include displaying, as the title of this thesis suggests, management. In terms of

output, it is relatively the same as seen in Figures 7 and 8. However, in terms of commands, they are different, but the meaning is the same. Based on the official Docker website, explains that previously, the ps -a verification command was the same as Docker, but a new command was released, namely docker container ls -a.

Table 15 Delete Image

Containers	**Name**	**Size/kB**	**Internet**	**speed**	**response**	**time**
Docker	Nginx	142	smartfren	12.66	02.39 seconds	05.40 WITA
Podman	Nginx	146	Smartfren	12.66	06. 32 seconds minute	5:4 WITA

It was found that Podman has a very low speed when compared to Docker, the difference is 4.7 minutes. From this, it was found that docker is better. The results of the analysis of the image parameters found that the Podman container was better than the docker container, the speed of Podman was superior with a difference of 2 minutes faster than docker. The images parameter shows that the command being executed is the same until the verification test, but then deleted the command starts to change where the same command cannot run on docker but on Podman the command runs. This proves that Podman is compatible with Docker and also proves that Podman is better than Docker.

As is known on the official Podman page, Podman is a container that runs using the rootless concept but does not rule out the possibility of running it with root access. At the trial stage, accessing the Podman service was carried out using the root service. Docker is a container service that runs with daemon and Docker CLI components, but in the test parameters of this research, container services were accessed using a predetermined user. The test results found that the differences in the user creation process were visible at the trial stage when the Docker user performed the installation process again, as when installing Docker for the first time. However, in Podman, it is different. Enter the IP of the host Operating System from VM1 and then log in as in Figure 7 so we can directly access the Podman container. The following is an analysis of making a user-accessible to that user.

Table 16 Create User

Containers	**user**	**Internet**	**speed**	**response**	**time**
Docker	Niki2	smartfren	12.82	40.20 minutes	06.47 WITA
Podman	iki	Smartfren	12.82	05. 30 minutes	06.47 WITA

By creating a user on both containers, it can be seen in the table above that Docker can get the user to the user access stage in a long time because when creating the user, we have to run the container to create the user, for example in this study using Centos. However, it differs from Podman in that users only need to run commands without having to run containers, making Podman faster than Docker. The time difference is very far, up to 35.10 minutes, so in terms of creating a user until the user is accessed, Podman is better. The table is analyzed in terms of container access via Docker and Podman.

CONCLUSION

Based on the analysis of the trials that have been carried out, the following conclusions can be drawn. The first parameter related to registry access carried out on docker and Podman containers found that Podman was better than docker in that there were no warnings, and required a relatively short time with the same speed as docker. The second parameter related to images found that Podman was slightly superior to Docker with a time of 2 seconds, however, in terms of delete, Docker was faster than Podman. They were overall won by Podman. The last parameter regarding accessing the container with a predetermined user was found to be that Podman was better than Docker. This was proven by the huge time difference when Docker took up to 1 (one) or 2 (two) hours but Docker took minutes to access the container. Podman. So the overall conclusion is that Podman is better than docker and the commands in docker are compatible with docker. This research has also accessed alpine

images with the docker.io registry on both containers with a capacity of 5.82MB. The time used is that Podman is 2 seconds faster with the same capacity, namely 5.82MB.

REFERENCES

[1] R. M. Akbar and R. R. W. Giri, "Analysis Of Digital Skill Confirmation Factors In The Use Of Mobile Banking Services In Bogor Regency," *Jurnal Ekonomi*, vol. 12, no. 04, pp. 313–326, 2023.

[2] B. Cherinka *et al.*, "Marvin: A tool kit for streamlined access and visualization of the SDSS-IV MaNGA data set," *The Astronomical Journal*, vol. 158, no. 2, p. 74, 2019.

[3] A. R. Kunduru, "Cloud BPM Application (Appian) Robotic Process Automation Capabilities," *Asian Journal of Research in Computer Science*, vol. 16, no. 3, pp. 267–280, 2023.

[4] R. Khattar, R. Hales, D. P. Ames, E. J. Nelson, N. L. Jones, and G. Williams, "Tethys App Store: Simplifying deployment of web applications for the international GEOGloWS initiative," *Environmental Modelling & Software*, vol. 146, p. 105227, 2021.

[5] H. Husain, A. Anggrawan, H. Santoso, H. T. Sihotang, D. Pyanto, and F. R. Hidayat, "Pengaturan Bandwidth Management Dan Time Limitation Berbasis User Manajer Mikrotik," *Jurnal Mantik Penusa*, vol. 2, no. 2, 2018.

[6] L. Baier, F. Jöhren, and S. Seebacher, "Challenges in the Deployment and Operation of Machine Learning in Practice.," in *ECIS*, 2019.

[7] K. Zandberg and E. Baccelli, "Femto-containers: Devops on microcontrollers with lightweight virtualization & isolation for iot software modules," *arXiv preprint arXiv:2106.12553*, 2021.

[8] S. Abraham, A. K. Paul, R. I. S. Khan, and A. R. Butt, "On the use of containers in high performance computing environments," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, IEEE, 2020, pp. 284–293.

[9] S. Kaiser, M. S. Haq, A. Ş. Tosun, and T. Korkmaz, "Container technologies for ARM architecture: A comprehensive survey of the state-of-the-art," *IEEE Access*, 2022.

[10] S. Kaiser, A. Ş. Tosun, and T. Korkmaz, "Benchmarking Container Technologies on ARM-Based Edge Devices," *IEEE Access*, 2023.

[11] B. Ward, "SQL Server 2022 on Linux, Containers, and Kubernetes," in *SQL Server 2022 Revealed: A Hybrid Data Platform Powered by Security, Performance, and Availability*, Springer, 2022, pp. 389–412.

[12] S. Kadri, A. Sboner, A. Sigaras, and S. Roy, "Containers in bioinformatics: applications, practical considerations, and best practices in molecular pathology," *The Journal of molecular diagnostics*, vol. 24, no. 5, pp. 442–454, 2022.

[13] A. K. Mishra, E. S. Pilli, and M. C. Govil, "CONTAIN4n6: a systematic evaluation of container artifacts," *Journal of Cloud Computing*, vol. 11, no. 1, pp. 1–14, 2022.

[14] V. T. Tran, "Maintaining Linux Repositories For Closed Network Infrastructure," 2023.

[15] D. Siswanto, G. Priyandoko, N. Tjahjono, R. S. Putri, N. B. Sabela, and M. I. Muzakki, "Development of Information and Communication Technology Infrastructure in School using an Approach of the Network Development Life Cycle Method," in *Journal of Physics: Conference Series*, IOP Publishing, 2021, p. 12026.

[16] I. Kamu, M. T. Parinsi, M. W. Kuhu, and A. V. Mananggel, "Computer Network Design in Vocational School Using Network Simulator," *International Journal of Information Technology and Education*, vol. 2, no. 1, pp. 22–31, 2022.

[17] A. M. Potdar, D. G. Narayan, S. Kengond, and M. M. Mulla, "Performance evaluation of docker container and virtual machine," *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020.

[18] I. Miell and A. Sayers, *Docker in practice*. Simon and Schuster, 2019.

[19] F. Björklund, "A comparison between native and secure runtimes: Using Podman to compare crun and Kata Containers." 2021.

[20] S. Giallorenzo, J. Mauro, M. G. Poulsen, and F. Siroky, "Virtualization costs: benchmarking containers and virtual machines against bare-metal," *SN*

Computer Science, vol. 2, no. 5, p. 404, 2021.

[21] S. Prabakaran *et al.*, "Predicting attack pattern via machine learning by exploiting stateful firewall as virtual network function in an SDN network," *Sensors*, vol. 22, no. 3, p. 709, 2022.

[22] K. Voulgaris *et al.*, "A Comparison of Container Systems for Machine Learning Scenarios: Docker and Podman," presented at the 2022 2nd International Conference on Computers and Automation (CompAuto), IEEE, 2022, pp. 114–118.

[23] D. P. VS, S. C. Sethuraman, and M. K. Khan, "Container Security: Precaution levels, Mitigation Strategies, and Research Perspectives," *Computers & Security*, p. 103490, 2023.

[24] R. Keller Tesser and E. Borin, "Containers in HPC: a survey," *The Journal of Supercomputing*, vol. 79, no. 5, pp. 5759–5827, 2023.

[25] R. Kałaska and P. Czarnul, "Investigation of Performance and Configuration of a Selected IoT System—Middleware Deployment Benchmarking and Recommendations," *Applied Sciences*, vol. 12, no. 10, p. 5212, 2022.