Using Docker Swarm to Improve Performance in Distributed Web Systems

Marian ILEANA
Interdisciplinary Doctoral School, Pitesti University Center
National University of Science and Technology POLITEHNICA Bucharest
Pitesti, Romania
marianileana95@gmail.com ,
ORCID 0009-0008-9624-2202

Maria Ioana OPROIU
Faculty of Engineering in Foreign Languages
National University of Science and Technology POLITEHNICA Bucharest
Bucharest, Romania
mariaioanaoproiu1@gmail.com ,
ORCID 0009-0002-0605-3106

Constantin Viorel MARIAN
Faculty of Engineering in Foreign Languages
National University of Science and Technology POLITEHNICA Bucharest
Bucharest, Romania
constantinvmarian@gmail.com ,
Corresponding author ,
ORCID 0000-0002-2846-8006

***Abstract*—This paper presents an approach to improving performance in distributed web systems by using Docker Swarm, a container orchestration tool built into Docker that allows for the management of a cluster of Docker engines. It is important to efficiently manage tasks and resources within a distributed environment due to the demand for scalable web services and increased availability. Docker Swarm provides a simple and efficient approach to managing Docker containers across multiple nodes, enabling the automatic distribution and scaling of web applications. In this article, we explore the benefits of using Docker Swarm in the context of distributed web systems, such as optimized resource management, scalability, and service reliability. Moreover, this text provides practical examples and case studies to demonstrate how the Docker Swarm framework can improve the performance and availability of distributed web applications in diverse production environments. Additionally, it outlines the benefits and difficulties related to using Docker Swarm and explores potential avenues for enhancing the performance of distributed web systems with this technology.**

Keywords—distributed computing, distributed information systems, computer performance, scalability, internet technology

I. INTRODUCTION

To create and modify complex distributed software systems, the software industry has adopted multi-cloud infrastructure [1]. This model works well when it comes to developing, deploying, and delivering distributed software. However, these cloud solutions are usually heterogeneous and cannot be used together. As a result, the complexity of the software development process and the management of multi-cloud systems has increased [2]. Virtualization based on Docker containers has recently emerged as a lightweight alternative for the software development process, and this method is gaining ground in the software development industry [3].

The software development process based on Docker containers can be scaled to multiple hosts in multiple clouds without Docker interoperability. As it represents a new approach to the cloud industry, distributed web development based on Docker Swarm has great power to provide a multi-cloud development environment without increasing system complexity [4].

Docker's ability to package applications and their dependencies into small containers that can be quickly transported, quickly started, and isolated between them [5].

Since the release of Docker, server-side web application development has changed dramatically. Thanks to the advent of Docker, it is now possible to develop applications using micro services that can be easily scaled and managed. Consider an example to better understand what a micro service is and how Docker helps deploy them. Imagine there are three programmers working in a web development team, each using a different operating system: macOS, Windows, and Ubuntu. Each of the aforementioned operating systems requires various specific adjustments. Instead, programmers must install and configure multiple distinct libraries for their programming languages. It is inevitable that libraries and programming languages will conflict in various environments. Adding additional servers to test and execute only increases the difficulty of ensuring uniform conditions in every development environment [6].

In chapter II, an analysis of the relevant literature is made. Two graphs will be attached and presented along with this analysis. The first indicates the most important authors for the keywords "Docker Swarm and Distributed Web Systems", and the second indicates the location of the countries where the authors work within the research centres. Chapter III, entitled "Methodology" initially deals with introductory theoretical aspects of Docker and its architecture. Docker Swarm features are also presented. Next, we initiate a distributed network via Docker. In chapter IV, we implement the concept of load balancing in the network to balance the tasks that are inside each node and prevent overcrowding. In the "Conclusion" chapter, we emphasize the importance of using Docker Swarm for an efficient orchestration of the available resources and their distribution in the cluster.

II. RELATED WORK AND LITERATURE REVIEW

A study by N. Marathe, A. Gandhi, and J. M. Shah [7]. Technology distribution computing that contributes to the implementation of a new computing infrastructure based on the virtualization of the necessary resources. By using the latest cloud application development paradigm, which allows expansive user access to these resources, workloads will grow rapidly on the server. This continuous expansion in volume leads to the efficient underutilization of available resources. This reason for the care of this technology is brought and presented to the public. The article's authors' objective is to balance the loads on each node of the tested network. This approach will facilitate the distribution of tasks among various nodes, avoiding overloading a single central point. Thus, this effort involves understanding the concepts of Docker and containers, which will help develop the concepts of container clustering and Kubernetes technology. In concluding the study, the authors' effort has demonstrated how services are

979-8-3503-4929-0/24/$31.00 ©2024 IEEE

accessed by nodes in a cluster using both Docker Swarm and Kubernetes and also explained the difference between them.

An article by Naik [8], the software development industry has adopted a multi-cloud infrastructure for designing and adapting highly complex and distributed software systems. This new hybrid cloud infrastructure allows combining and adapting cloud platforms and providers for various activities involving software development. There are many advantages to multi-cloud infrastructure, such as reducing reliance on a single provider and minimizing the risk of large-scale data loss or downtime. However, there are also multiple challenges, such as increased complexity due to different technologies, interfaces, and services. Docker has introduced a container based software development method in recent years and has gained a large share of the software industry market. It recently introduced a system development tool called Swarm, which extends the Docker container-based software development process across multiple clouds without encountering classic interoperability issues. Distributed software development based on Docker Swarm represents an innovative approach for the cloud industry; however, it has immense potential to provide a multi-cloud development environment, removing the worry of its complexity. The author's paper presents a simulation of building a virtual system of systems (SoS) for the distributed software development process on multiple clouds. Virtual SoS simulation is based on Docker Swarm, Virtual Box, Mac OS X, nginx, and Redis. However, the same SoS can be created on any of the clouds supported by Docker, simply by replacing the driver name with the preferred cloud name, like this: Digital Ocean, Google Compute Engine, Amazon Web Services, Microsoft Azure, etc.

The analysis carried out by Cerin et al. [9] presents initial concepts of a new scheduling strategy integrated into the Docker Swarm scheduler. The purpose of this paper is to introduce the fundamental principles and implementation details of a new scheduling strategy based on various Service Level Agreement (SLA) categories. Their proposed approach addresses the problems faced by companies that manage a private computing infrastructure and want to optimize the scheduling of multiple requests sent online by users. Each request represents a request to create a container. At the moment, Docker Swarm makes use of three fundamental scheduling techniques: spread, binpack, and random. Each of these techniques gives a container a set amount of resources. However, the novelty of the authors' proposed strategy is to use the user's SLA class to determine the resource allocation for the container, dynamically taking into account the number of CPU cores required according to the user's SLA and the workloads of the parallel computing machines within the infrastructure. Their testing of the proposed new strategy is done by emulating different components of our general frameworks and highlighting the potential of their approach for future developments.

Applying a dynamic analysis method to the topic of distributed web systems, especially Docker Swarm, specialty sites offers a wide range of articles signed by a large number of writers. This empirical study was conducted using the "Dimensions.ai" platform as a database and its search functionality. To facilitate the identification of authors who have published on this topic, the results were interpreted using the VOSviewer application. This resulted in a graph. The dataset contains 7141 authors. An author's relevance is determined by appearing as an author in at least three papers, and that author was also required to have at least two citations. A total of 585 authors met these criteria (see Fig. 1).

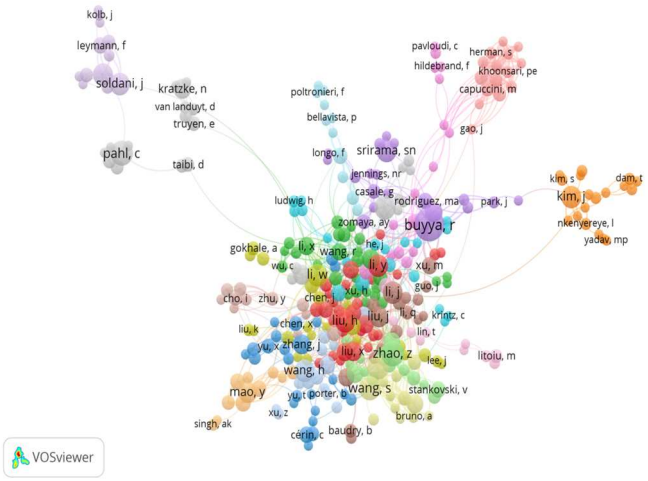

Fig. 1. A graphic map of authors for the search term "Docker Swarm and Distributed Web Systems"

In addition, we created a map of the countries from which these authors, who write articles about Docker Swarm and distributed web systems, come from using the same data set. For a country to be considered relevant, it had to have at least three fives, each with at least three citations. The total number of countries from which the authors originated is 88; after applying the previously mentioned criteria, 60 countries met the conditions (see Fig. 2).

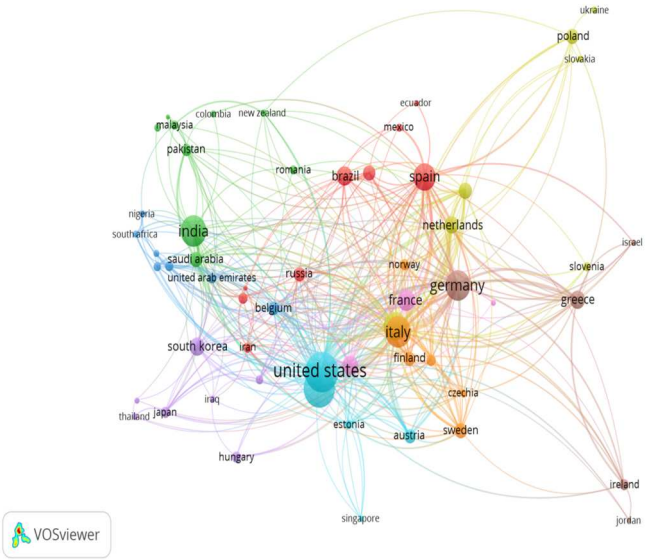

Fig. 2. A country map of authors for the search term "Docker Swarm and Distributed Web Systems"

III. METHODOLOGY

A. Theorical introduction

1) Docker: Docker is a platform that makes it easy for developers to package their software in containers. The application itself, as well as all the libraries and other dependencies it needs to function, are gathered into a single

unit in these containers. This makes it much simpler to deploy and consistently run applications in different software development environments [10].

Major benefits of using Docker [11]:

- Portability: Regardless of the underlying operating system, Docker containers can operate on any deployed Docker system. This allows developers to develop and test applications on their local devices, confident that they will run smoothly on production servers.
- Isolation: Applications cannot interfere with each other or the host system because each Docker container runs in an isolated environment. This can improve stability and security.
- Reproducibility: Docker containers are built according to instructions, so they are easy to reproduce. This makes it easy to ensure that all developers and testers are working in an identical environment.

2) Docker Architecture: The architecture of virtual machines and Docker software containers (see Fig. 3) is comparable [14]. The stack on the right is a virtual machine that uses a hypervisor to emulate the guest operating system. The application software, dependencies, and guest operating system are all included in a virtual machine (VM). Each requires a separate VM, dependencies, and the application host operating system to be deployed. The stack on the left illustrates the Docker container software on a Linux host. Docker bundles the program and its dependencies into modular containers using the host Linux operating system. No VM is required, and OS resources for the two application stacks are shared between different containers. The middle stack illustrates Docker on a non-Linux system. Docker needs Linux to function, so in order to execute Docker and enclose the software containers, a basic virtual machine (VM) with a mini-Linux guest OS is needed. This still has the advantage that there is only one VM and a guest Linux system required, regardless of the number of containers.

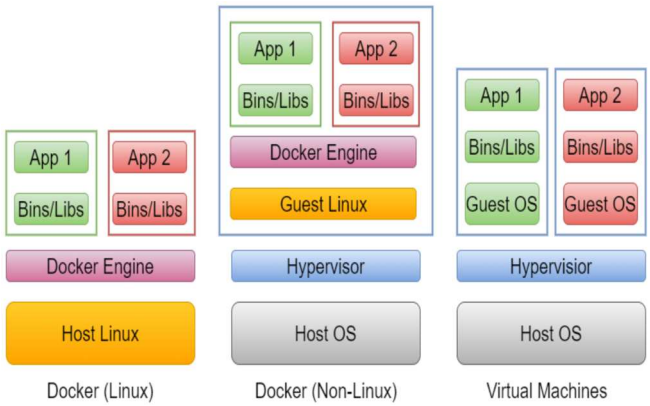

Fig. 3. Diagram of the Docker architecture depending on the development environment

3) Docker Swarm: Docker Swarm is a tool that Docker Engine includes to orchestrate containers. It allows you to manage a unified Docker cluster of multiple physical or virtual machines. Each machine that is part of the cluster is called a node. One or more nodes manage the swarm and supervise the activity of the other nodes in the cluster [12]

Key features of Docker Swarm [13]:

- Clustering: Swarm enables containers to run as a single distributed system by grouping together multiple machines. Since a single failed container will not prevent the entire application from running, this ensures high scalability and redundancy.
- Simple to use: Swarm is quite simple to set up and use because it is built into the Docker Engine. With a few Docker commands, you can create a Swarm cluster.
- Scheduling automation: Swarm ensures efficient resource utilization by automatically distributing containers to cluster nodes. It can also restart failed containers on other nodes, making the application more accessible.

Organizations looking for a simple container orchestration solution that is already integrated with Docker will find Swarm a great option [12]. However, other tools like Kubernetes may be better for more complex or large-scale orchestrations [7].

B. Load Balancing

Internal load balancing and the ingress routing mesh are the basic methods used to route traffic and manage incoming requests. By employing the container DNS records, for example, a round robin technique is used by default for load balancing (e.g. App.1 > App.2 > App.3 > App.1 > App.2 . . . and so on) [15].

A bridge network cannot be used with swarms since it is specific to a single Docker host. Table 1 represents the architecture of the system where we deploy this experiment.

TABLE I. COMPUTER HARDWARE COMPONENTS

Hardware Component	***Specification***
Processor	Intel Core i7-1165G7 2.8 GHz
Memory	16 GB DDR4
Graphics card	NVIDIA GeForce GTX 1650 Ti 4 GB DDR6
Storage / Hard drive	1 TB NVMe SSD
Operating system	Windows 10 Pro 64-bit

(*Main computer components*)

In Fig. 4 is represented the installing Chocolatey, using PowerShell. Chocolatey is used to install Docker, using the last command in the figure. Chocolatey is a package manager in Windows. It allows users to automate the process of installing, updating and managing software packages on the Windows machines.

Similar to package managers in Linux distributors like apt-get or yum, Chocolatey provides a command-line interface CLI through which users can search for installing, updating and uninstalling software packages with ease.

Chocolatey utilizes a repository of software packages, which are maintained by the Chocolatey community. Users can access this repository to find and install a wide range of software applications, tools and utilities.

Chocolatey shows as some advantages, one of them is its ability to streamline the software management process, allowing also users to easily install and update multiple

software packages with a single command line. This can be useful also for administrators, developers and power users frequently work with software installations and updates on Windows machines.

```
PS C:\WINDOWS\system32> Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]: SecurityProtocol = [System.Net.ServicePointManager]:
SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).
DownloadString('https://chocolatey.org/install.ps1'))
Getting latest version of the Chocolatey package for download.
Getting Chocolatey from https://chocolatey.org/api/v2/package/chocolatey/0.10.15
Downloading 7-Zip comandline tool prior to extraction.
Extracting C:\Users\Maria Ioana\AppData\Local\Temp\chocolatey\chocInstall\chocolatey.zip to
C:\Users\Maria Ioana\AppData\Local\temp\chocolatey\chocInstall…
Installing chocolatey on this machine
Creating ChocolateyInstall as an environment variable (targeting 'Machine')
    Setting ChocolateyInstall to 'C:\PrpgramData\chocolatey'
WARNING: It's very likely you will need to close and reopen your shell before you can use
choco.
Restrincting write permissions to Administrators.
We are setting up the Chocolatey package repository.
The packages themselves go to 'C:\ProgramData\chocolatey\lib'
    (i.e. C:\ProgramData\chocolatey\lib\yourPackageName).
A shim file for the command line goes to 'C:\ProgramData\chocolatey\bin' and points to an
executable in 'C:\ProgramData\chocolatey\lib\yourPackageName'.

PS C:\WINDOWS\system32> choco install docker-destop --pre
```

Fig. 4. Docker Swarm Cluster architecture

Also, Chocolatey ensures consistent installations of software packages by handling dependencies and configurations automatically. This helps prevent issues related to missing dependencies or incompatible software versions.

Chocolatey can be easily integrated into various scripts and automated workflows, allowing for efficient software deployment and configuration management. This is particularly useful for system administrators and DevOps professionals who need to manage software installations across multiple machines.

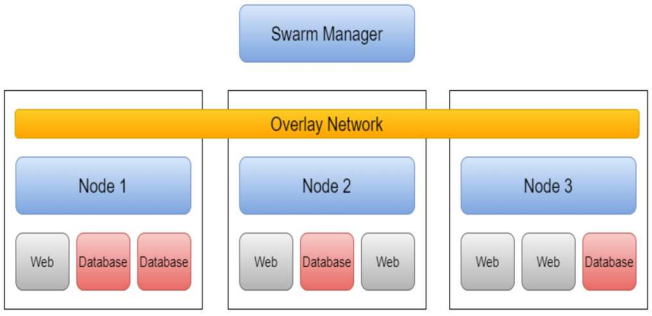

Fig . 5. Docker Swarm Cluster architecture

Using Docker and Chocolatey has multiple advantages. Docker ensures consistency in application environments across different stages of development lifecycle.

By combining Docker with Chocolatey, we can not only maintain consistency within Docker containers, but also ensure that the software installed within these containers is managed across development, testing and production environments using Chocolatey. Chocolatey simplifies the installation and management of software packages on Windows.

By using Chocolatey within Docker containers, developers can easily specify and manage dependencies required by their applications. This ensures that the required software components are installed and configured correctly within the Docker environment, reducing the risk of compatibility issues.

Instead of using an overlay network, which is dispersed across numerous hosts, we must utilize a swarm, which employs multiple hosts [16] (see Fig. 5).

```
PS C:\Users\Maria Ioana\Desktop\Articol\learning-docker-master\Proiect_Docker> docker stack
deploy -c php-mysqli-apache.yml php.mysqli-apache
Creating service php-mysqli-apache_tut10-db
Creating service php-mysqli-apache_tut10-www
PS C:\Users\Maria Ioana\Desktop\Articol\learning-docker-master\Proiect_Docker> docker stack
ls
NAME                SERVICES        ORCHESTRATOR
php-mysqli-apache     2                    Swarm
PS C:\Users\Maria Ioana\Desktop\Articol\learning-docker-master\Proiect_Docker> docker stack
ps php-mysqli-apache
ID                  NAME                    IMAGE                                    NODE
      DESIRED STATE          CURRENT STATE               ERROR             PORTS
tpczubdpra41      php-mysqli-apache_tut10-www.1   dockermaria30/php-mysqli-apache:1.1 nodeman
ager    Running             Preparing 40 seconds ago
Wxjuke3w2cwr      php-mysqli-apache_tut10-db.1    dockermaria30/php-mysqli-apache:1.1 nodeman
ager    Running             Preparing 42 seconds ago
hujxvznkqohb      php-mysqli-apache_tut10-www.2   dockermaria30/php-mysqli-apache:1.1 nodeman
ker     Running             Preparing 40 seconds ago
bglsc3mjjuif      php-mysqli-apache_tut10-www.3   dockermaria30/php-mysqli-apache:1.1 nodeman
ager                        Preparing 40 seconds ago
x4s4gpfhzblh      php-mysqli-apache_tut10.www.5   dockermaria30/php-mysqli-apache:1.1 nodeman
ker    Running              Preparing 40 seconds ago
```

Fig 6. Representation of starting the Docker instances

We are going to deploy the stack using the docker stack deploy command, "-c" is our compose file. If we run this command from inside the correct container because we have already attached it to the node manager, it should be deployed on the node manager itself.

Regarding the displayed result after running this command, we notice that it will not create the networks. Further, to see the stacks we have available in our folder, we used the stack ls command. After running this command, we notice that we have a stack, on which two services are running, within Orchestration Swarm [17].

After we run the docker stack ps php-mysqli-apache command, we notice that we have 6 running instances, all of them are in the state of running. This is the way we want to have it. To see which nodes are connected to the processes, we will run the command docker-machine ssh nodemanager "docker ps", which will show us which Docker containers are running on the host (see Fig. 6) [18].

IV. RESULTS

If we try to unlock localhost, we will not succeed; we will have an error because there is no host that is currently running on our machine. What we can do now, in order to work, is to find the IP of the host, which we will find using the simple command docker-machine ip nodemanager. Thus, we can see the displayed IP, namely 192.168.1.29. This is how Swarm works. Finally, if we want to destroy this Swarm, we can do it simply with the help of a command, namely docker stack rm php-mysqli-apache [19].

Thus, we can delete a swarm, but it will not delete the network because it was created outside the stack. After deleting the stack, if we run again the command to display elements within a folder, namely docker stack ls, we will be able to see what is left without any stack, because we deleted them. Finally we execute the command docker-machine ssh nodeworker "docker ps" and these processes are not running, they are stopped and will close (see Fig. 7) [20].

The novelty of this paper was through the design and implementation of the system. We have developed an architecture so that the Docker Swarm solution can be implemented, in order to achieve load balancing. We evaluated the performance of the solution through tests and measurements, resulting in data on the improvements brought

by the use of this solution. We also analyzed and interpreted the collected data, highlighting the advantages and disadvantages of Docker Swarm.

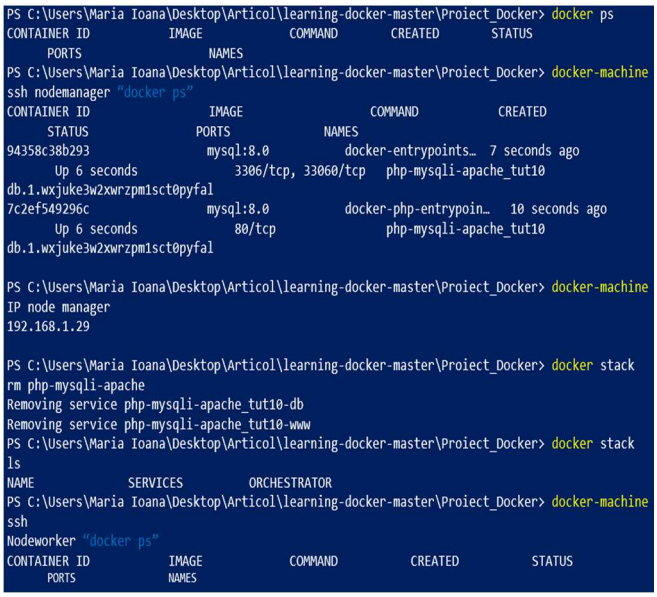

Fig 7. Load balancing result

V. CONCLUSIONS

In conclusion, using Docker Swarm within distributed web systems is a practical and effective method for improving performance and scalability. By managing containers with Docker in a distributed environment, Docker Swarm will facilitate efficient resource orchestration and load distribution across the cluster. Thus, it allows the dynamic adaptation of the infrastructure to the fluctuating requirements of web applications and ensures the continuous availability of web services.

In addition to failing to secure the Swarm with TLS and improperly managing sensitive data, other common issues in Docker Swarm deployments include failing to implement proper authentication and authorization mechanisms, leaving the cluster vulnerable to unauthorized access and potential data breaches. Furthermore, insufficient network segmentation and firewall policies might expose internal services and data to external attacks, jeopardizing the Swarm's overall security posture. Failure to update and patch Docker engines, Swarm components, and host operating systems on a regular basis might result in security vulnerabilities and attacks.

Using Docker Swarm in distributed web systems enables efficient orchestration of their Docker containers across multiple nodes, facilitating scalability and load management in a simplified and robust manner.

Furthermore, inadequate logging, monitoring, and auditing methods may result in unreported security events or compliance breaches, impeding incident response and regulatory compliance efforts. Furthermore, a lack of disaster recovery planning and testing may expose the Swarm to data loss, extended downtime, and business continuity problems in the case of system failures or calamities.

For Docker Swarm monitoring collecting metrics is the base foundation. Monitoring Docker Swarm entails gathering a wide range of metrics to acquire a full understanding of the health and performance of services operating on the cluster. These metrics provide the foundation for evaluating resource use, detecting bottlenecks, and optimizing application deployment. CPU and memory consumption are important indicators for understanding workload and resource allocation across nodes and containers. Network traffic measurements are used to monitor data transfer rates, identify communication patterns, and discover abnormalities that may signal network faults or security risks.

Monitoring service response times also enables the evaluation of application performance and user experience, assisting in the identification of sluggish or inefficient services that need to be optimized. By collecting and analyzing these metrics, Docker Swarm administrators may proactively discover difficulties, fix problems, and optimize resource allocation to maintain their installations' stability, scalability, and efficiency.

Docker Swarm proves to be a useful solution for software development teams who want to optimize the performance and availability of distributed web applications, due to the benefits it offers.

However, it is essential to properly understand and manage the increased architectural complexity and increased maintenance costs associated with implementing and managing such a system that includes Docker Swarm.

As future development, Docker Swarm is beneficial when used for:

1) Web applications: Docker Swarm is great for deploying and administering online applications like e-commerce sites, social networks, blogs, and content management systems. Its horizontal scaling capability, traffic distribution over several nodes, and high availability ensure that web-based applications run reliably and efficiently. Furthermore, Docker Swarm's declarative service definition paradigm streamlines application deployment and upgrades, making the development and release process easier for web developers.

2) Microservices: Swarm is ideal for delivering and maintaining microservices designs, which divide programs into tiny and independently deployable services. Docker Swarm's service orchestration features enable the deployment, scaling, and load balancing of microservices throughout the cluster, resulting in increased agility, scalability, and resilience. Containerization allows each microservice to function in isolation, with its own dependencies and customizations, while yet benefiting from Docker Swarm's administration and resource usage [21].

3) Continuous Integration (CI) and Continuous Delivery (CD): Docker Swarm may play a critical role in automating the CI/CD process, allowing enterprises to deploy software changes more rapidly, reliably, and consistently. Docker Swarm interacts smoothly with CI/CD pipelines, enabling developers to automatically create, test, and deploy apps in containerized environments. Teams may use Docker Swarm capabilities like rolling updates, blue-green deployments, and automatic scaling to fulfill continuous integration and delivery goals, which reduces time-to-market and ensures software release quality [22].

REFERENCES

[1] J. Hong, T. Dreibholz, J. A. Schenkel, and J. Hu, "An overview of multicloud computing," in Advances in intelligent systems and computing, 2019, pp. 1055–1068. doi: 10.1007/978-3-030-15035-8 103.

[2] L. Gupta, R. Jain, M. Samaka, A. Erbad, and D. Bhamare, "Performance evaluation of Multi-Cloud management and control systems," Recent Advances in Communications and Networking Technology, vol. 5, no. 1, pp. 9–18, Dec. 2016, doi: 10.2174/2215081105999160523095630.

[3] A. Azab, "Enabling docker containers for High-Performance and Many-Task computing," 2017 Ieee International Conference on Cloud Engineering (Ic2e), Apr. 2017, doi: 10.1109/ic2e.2017.52.

[4] E. N. Preeth, J. P. Mulerickal, B. Paul, and Y. Sastri, "Evaluation of Docker containers based on hardware utilization," 2015 International Conference on Control Communication & Computing India (ICCC, Nov. 2015, doi: 10.1109/iccc.2015.7432984.

[5] T. N. Bui, "Analysis of Docker Security," arXiv (Cornell University), Jan. 2015, [Online]. Available: https://arxiv.org/pdf/1501.02967.pdf

[6] S. Wang, L. Zhu, and M. Cheng, "Docker-based Web Server Instructional System," 2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS), Jun. 2019, doi: 10.1109/icis46139.2019.8940219.

[7] N. Marathe, A. Gandhi, and J. Shah, "Docker swarm and kubernetes in cloud computing environment," 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), Apr. 2019, doi: 10.1109/icoei.2019.8862654.

[8] N. Naik, "Building a virtual system of systems using docker swarm in multiple clouds," 2016 IEEE International Symposium on Systems Engineering (ISSE), Oct. 2016, doi: 10.1109/syseng.2016.7753148.

[9] C. Cerin, T. Menouer, W. Saad, and W. B. Abdallah, "A New Docker Swarm Scheduling Strategy," In 2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2), Nov. 2017, doi: 10.1109/sc2.2017.24.

[10] W. M. C. J. T. Kithulwatta, K. P. N. Jayasena, B. T. G. S. Kumara, and R. M. K. T. Rathnayaka, "Docker incorporation is different from other computer system infrastructures: A review," 2021 International Research Conference on Smart Computing and Systems Engineering (SCSE), Sep. 2021, doi: 10.1109/scse53661.2021.9568323.

[11] T. Vase, "Advantages of Docker," Jan. 2015, [Online]. Available: https://jyx.jyu.fi/handle/123456789/48029

[12] B. Magableh and M. Almiani, "A self healing Microservices Architecture: A case study in Docker Swarm Cluster," in Advances in intelligent systems and computing, 2019, pp. 846–858. doi: 10.1007/978-3-030-15032-7 71.

[13] D. Sæther, "Security in Docker Swarm: orchestration service for distributed software systems," 2018. [Online]. Available: https://bora.uib.no/bora-xmlui/handle/1956/18649

[14] S. Noor, B. Koehler, A. Steenson, J. Caballero, D. Ellenberger, and L. Heilman, "IOTDOC: a Docker-Container based architecture of IoTEnabled cloud system," in Studies in computational intelligence, 2019, pp. 51–68. doi: 10.1007/978-3-030-24405-7 4.

[15] J. Qian, Y. Wang, X. Wang, P. Zhang, and X. Wang, "Load balancing scheduling mechanism for OpenStack and Docker integration," Journal of Cloud Computing, vol. 12, no. 1, Apr. 2023, doi: 10.1186/s13677-023-00445-3.

[16] N. Singh et al., "Load balancing and service discovery using Docker Swarm for microservice based big data applications," Journal of Cloud Computing, vol. 12, no. 1, Jan. 2023, doi: 10.1186/s13677-022-00358-7.

[17] J. Cito and H. C. Gall, "Using docker containers to improve reproducibility in software engineering research," Proceedings of the 38th International Conference on Software Engineering Companion., May 2016, doi: 10.1145/2889160.2891057

[18] C. Huang and C.-R. Lee, "Enhancing the Availability of Docker Swarm Using Checkpoint-and-Restore," 14th International Symposium on Pervasive Systems, Algorithms and Networks & 2017 11th International Conference on Frontier of Computer Science and Technology & 2017 Third International Symposium of Creative Computing (ISPAN-FCST-ISCC), Jun. 2017, doi: 10.1109/ispan-fcst-iscc.2017.69.

[19] О. Романов, V. Mankivskyi, L. Veres, and I. Saychenko, "Analysis of Performance in Docker Net deployed in AWS cloud," 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T), Oct. 2021, doi: 10.1109/picst54195.2021.9772184.

[20] A. R. Manu, J. K. Patel, S. Akhtar, V. K. Agrawal, and K. N. B. Murthy, "Docker container security via heuristics-based multilateral security-conceptual and pragmatic study," 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT), Mar. 2016, doi: 10.1109/iccpct.2016.7530217.

[21] N. Singh *et al.*, "Load balancing and service discovery using Docker Swarm for microservice based big data applications," *Journal of Cloud Computing*, vol. 12, no. 1, Jan. 2023, doi: 10.1186/s13677-022-00358-7.

[22] A. C. Codex, "Integrating Continuous Deployment Pipelines with Docker Swarm," Reintech, https://reintech.io/blog/integrating-continuous-deployment-pipelines-docker-swarm (accessed 16 April 2024).