Index N0: CT-02-31

Database Management System Deployment on Docker Containerization for Distributed Systems

W.M.C.J.T. Kithulwatta (1st Author)
Faculty of Graduate Studies
Sabaragamuwa University of Sri Lanka
Belihuloya, Sri Lanka
Faculty of Technological Studeis
Uva Wellassa University of Sri Lanka Badulla, Sri Lanka
chirantha@uwu.ac.lk

K.P.N. Jayasena (2nd Author)
Department of Computing and Information Systems
Faculty of Applied Sciences
Sabaragamuwa University of Sri Lanka
Belihuloya, Sri Lanka
pubudu@appsc.sab.ac.lk

B.T.G.S. Kumara (3rd Author)
Department of Computing and Information Systems
Faculty of Applied Sciences
Sabaragamuwa University of Sri Lanka
Belihuloya, Sri Lanka
kumara@appsc.sab.ac.lk

R.M.K.T. Rathnayaka (4th Author)
Department of Physical Sciences and Technology
Faculty of Applied Sciences
Sabaragamuwa University of Sri Lanka
Belihuloya, Sri Lanka
kapilar@appsc.sab.ac.lk

***Abstract*—Containerization is a novel technology that brings an alternative for virtualization. Due to the most infrastructure-based features, most computer system administration engineers use Docker as the infrastructure level platform. On the Docker containers, any such kind of software service can be deployed. This study aims to evaluate Docker container based relational database management system container behavior. Currently, most scholarly research articles are existing for the database engine performance evaluation under different metrics and measurements of the database management systems. Therefore, without repeating them: this study evaluated the data storage mechanisms, security approaches, container resource usages and container features on the launching mechanism. According to the observed features and factors on the containerized database management systems, containerized database management systems are presenting more value-added features. Hence containerized database management system Docker containers can be recommended for the distributed computer systems for getting the benefit of effectiveness and efficiency.**

Keywords - containers, database management systems, distributed Systems, Docker, MySQL, PostgreSQL

I. INTRODUCTION

Virtualization is an old concept that provides on premise or cloud-based virtual machines to deploy any such software applications or system services. Virtual machines provide the facility to optimize the host server capacity by launching multiple different operating systems. Virtualization carries additional overhead since one virtual machine consists of a fully installed operating system. To optimize the whole virtual machine process, reduce the virtual machine weight and enhance the infrastructure performances: an alternative technology called container virtualization arrived.

Within the containerization, containers allow to deploy and run software applications and services without using or creating separate virtual machines. By sharing the host computer operating system kernel, separated multiple containers are executing on the infrastructure. To execute any software application or service, all necessary software dependencies, libraries and binaries are packaged into each container [1].

For the secure execution of the containers, basic Linux features are used for the containers. Those are *cgroups*, *chroot* and *namespaces*. Since containers are not using full operating system instances, containers require less CPU, memory and storage capacity according to the fundamental theory of container virtualization [1]. Fig. 1 presents the container architecture as a graphical notation.

Fully packaged independent containers are running on own container engine. Each container consists of its own independent subsystem for the file system, memory and network. The container engine is the component that has the authority to manage containers. Containers of the same container engine share the same host operating system. Therefore, the infrastructure supports launching a massive number of containers on a single operating system [2].

Within the practitioner of containerization, various container management technologies are available. Docker, Rkt and Linux containers are a few container management technologies [1]. Among them, Docker is the most trending and most popular container management technology [3]. According to the official Docker documentation, currently, eleven million developers are engaged with Docker and thirteen billion of Docker images have been downloaded [4]. Database Management Systems are the specific software packages that provide the dedicated technology and facility to store and retrieve data in efficiently and appropriately [5]. A Database Management System stores data in a most prominent way to retrieve, manipulate, manage and produce information.

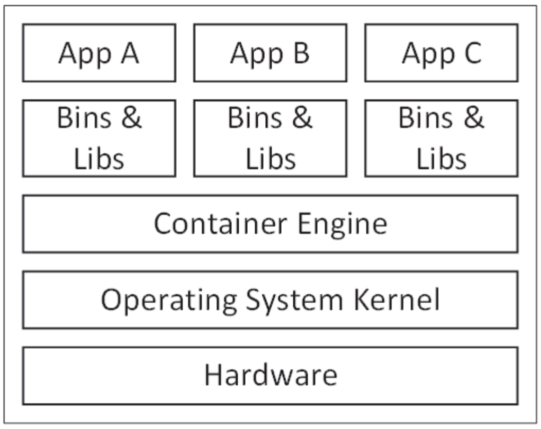

Fig. 1: Container architecture [2]

Relational Database Management Systems (RDBMS) are a specific database management system specification, which are based on the relational model and Structured Query Language (SQL). Most modern database systems are RDBMSs. MySQL, PostgreSQL, IBM DB2, MS SQL Server and Oracle are the best examples for the RDBMSs [6].

Within the existing research studies, the authors have evaluated the database management systems by considering the taken time to particular SQL queries and response time in commonly.

Currently, there is a trend to shift software services and applications to the container-native methodology. Hence database management systems are also shifting to the containers. There are a lot of scholarly research articles to evaluate and measure the performances of databases and database management systems. Therefore, this research activity was aimed to do an experimental study on the database management system deploying infrastructures for the distributed systems.

The overall research study provides answers to the below research questions.

RQ1: How to mechanize the Docker based Database Management Systems to have a persistence data storage approach?

RQ2: What kind of container-based infrastructure level security approaches can be applied to the Docker based Database Management Systems?

RQ3: How are the container resource usage and utilization from the host computer infrastructure for Database engine activities?

RQ4: What are the differences among manually deployed Database Management System containers over ready-made Docker image approaches?

II. METHODOLOGY

For the study, the experimental platform was launched on a cloud-based Linux environment. To launch the Docker service, an Ubuntu computer host was used. Canonical Ubuntu 18.04 operating system was used for the host computer infrastructure. Host computer-based architecture was GNU/Linux 4.15.0-112-generic x86_64. As well, the host computer was with two virtual CPUs, 15 GB internal memory and 1 Gbps network bandwidth. An external block volume was attached to the host computer. The directory path of the host computer: ***/home/$user*** was mounted to the block volume.

On the above-mentioned host computer, the Docker platform was launched. The launched Docker configurations are mentioned in table 1. (Within table 1, these abbreviations are used for the easiness of the representation: API = Application Programming Interface, OS = Operating System)

It presents the Docker version, Docker API version supporting based operating system architecture.

Table 1: Configured Docker details

Option	**Detail for configuration**
Client: Docker Engine - Community	
Version	19.03.9
API version	1.40
OS/Architecture	Linux/amd64
Server: Docker Engine - Community	
Version	19.03.9
API version	1.40
OS/Architecture	Linux/amd64

Within the above Docker platform, two internal Docker networks were established using the bridge drivers. One network (N_1) was with 172.17.0.0/16 as the subnet and 172.17.0.1 as the gateway. The second network (N_2) was with 172.22.0.0/16 as the subnet and 172.22.0.1 as the gateway. For the experiment study, two RDBMSs were used: MySQL and PostgreSQL. Those two RDBMSs were launched on two separate Ubuntu bionic Docker containers within the above mentioned two Docker networks. Table 2 presents the infrastructure details for the distributed RDBMS Docker farm. The term IP was abbreviated for the internet protocol.

Table 2: Docker container details

Container	Network	Container IP	Container port	Host port
MySQL conatiner	N_1	172.17.0.2	3306	3300
PostgreSQL container	N_2	172.22.0.2	5432	5400

For the experimental study, MySQL version 5.7.30 and PostgreSQL version 12.6 was used. For the MySQL RDBMS, a database with 24 tables was used. For the PostgreSQL RDBMS, a database with 30 tables was used.

For each container, the specific data and file paths were mounted to the path, ***/var/lib/docker/volumes*** on the host computer. By default, MySQL and PostgreSQL RDBMSs are serving through the port 3306 and 5432. Within the experimented approach, each port was mapped to 3300 and 5400 respectively for the host ports. By performing data retrieval operations on the RDBMSs, the host container resource usage and utilization was evaluated. To launch each RDBMS Docker container, two approaches were followed. Each approach is defined below, and each approach was evaluated with their behaviors.

Approach 1: Launch Ubuntu bionic Docker container and install the respective RDBMS service (Used as the main approach for the study)

Approach 2: Launch RDMS Docker container using Docker images/templates.

III. RESULTS AND DISCUSSION

After launching the experimental platform with the RDBMS Docker containers, specific operations and analyzing the proposed disturbed mechanism, was performed.

A. *Persistence data storage/archiving*

The Docker container-based platform was launched on a host computer infrastructure. To keep a more data persistence, the main directory path of the Docker ***(/var/lib/docker/)*** was linked to the path, ***/home/$user*** (to the block volume path) of the host computer infrastructure. Hence, specific objects and configurations of the Docker could be attached to the block volume. Those very specific Docker objects and configurations were container, images, volumes, network, Docker swarm, plugins, temporary files, etc. Therefore, as the primary mechanism, the whole Docker based infrastructure was with persistence data storage mechanism.

Furthermore, to keep a persistence data storage and archiving approach for each container separately, data volumes were mounted to each container. For the MySQL RDBMS Docker container, the paths ***/var/lib/*** and ***/var/log/*** are carried more specific data and configurations for the database engine. Hence those two directory paths were mounted to Docker data volumes. To mount those two paths, two different approaches were used, and those two approaches are defined below.

- Both ***/var/lib/*** and ***/var/log/*** directory paths were mounted to one Docker data volume.
- For ***/var/lib/*** and ***/var/log/*** data directory paths were mounted two separate Docker data volumes.

Between above two approaches, mounting two Docker data volumes was more strategic since if a Docker volume crashed, it does not affect the rest of Docker data volume.

Same as above, for the PostgreSQL RDBMS Docker container, the data directory path, ***/var/lib/postgresql/*** was identified as carrying most key data and configurations of the service. Hence, the identified directory path was mounted to a Docker data volume.

After making a stable database management system on two Docker containers, two Docker containers were crashed by stopping the containers and jamming with installing different unwanted packages and dependencies. Thereafter, respective data volumes were re-attached for the new Docker containers which carry the RDBMS service. Then without losing any data or configurations, the new Docker container was restored to previous data and configurations.

Without detaching the previously attached Docker data volumes from containers, attachments for new Docker containers were possible. But assignment of the previous host port to the new Docker container's host port was not possible even if the previous container was stopped on the Docker engine. Therefore, essentially, previous container needed to be removed to assign the host port for the new Docker container same as the previous container.

Those all-mounted Docker data volumes were directly linked with host computer infrastructure. Hence any file or directory of those data paths, could be faced for any such operation on the file system (copy/rename/delete/move).

B. *Secured approchess for the infrastructure*

For the experimental setup, each container port was mapped for a host port. For the MySQL RDBMS Docker container perspectival: 0.0.0.0:3306 → 3300/tcp and for the PostgreSQL RDBMS Docker container perspectival: 0.0.0.0:5432 → 5400/tcp was applied as the port mapping. As a usual practice, attacks or vulnerabilities are looking for default ports of any services. Hence by mapping each service port (same as container port in this case) was mapped to arbitrary port value. Hence guessing port values are not possible for this case and the proposed approach is bringing a valuable security concern for the infrastructure.

As well, the default service port of each container was changed for arbitrary values by changing the service installation configurations at the next stage. Then newly assigned ports were mapped for a new set of host ports. Therefore, default port values are not available for any of the layers. Hence, guessing port values from an external person or device was mitigated.

Both RDBMS Docker containers are in two isolated Docker networks. By using static network address translation (NAT) for each multiple private IP addresses of the containers, all inward and outward traffic was handled in a more securely. Without exposing the container IP address to the outside world, the public IP address of the host computer was exposed to the outside world. Therefore, without translating the container IP addresses to the external IP addresses, internal IP addresses could not be routed to the external world. Furthermore, the NAT mechanism was assured that all outbound traffic is from valid and known external IP addresses. Therefore, the approach helps to enhance the infrastructure security, all incoming and outgoing requests go through a translation process. The process ensures to qualify and/or authenticate all incoming traffic.

As described in section III.A, the main data and file directory of Docker was linked to the path: ***/home/$user.*** That directory path was privileged only for the super user. Therefore, without any command or operations could not be done for that directory path without super user credentials. Hence that ensures the security of the approach.

C. Container resource usage & utilization

To measure and evaluate the internal resource consumption of each RDBMS Docker container from the host computer infrastructure, main resource metrics were measured for the idle state and data operating states.

Table 3 presents primary details of each Docker container and resource usage from the host computer infrastructure. For the ease of documentation purposes, below abbreviations were used for table 3. [Container ID = the unique identifier or the Docker container within the launched Docker platform, Name = assigned container name, CPU% = the percentage of the container consuming CPU from the host computer, Memory% = the percentage of the container consuming memory from the host computer, MEM_Usage/Limit = total memory which is used by the container and the allowed total memory to use, NET I/O = the overall amount of the data which the container has sent and received over the network interface and PIDs = created the total amount of the processors or threads by the container][7].

Table 3: Container resource usage and utilization

Container measurement	**Measurement value of each container**	
	MySQL container	**PostgreSQL container**
Container ID	c35b95254633	db7f7cee8390
Name	MySQL Container	PGSQL Container
CPU% (for idle states)	0.25%	0.38%
Memory%	0.07%	0.15%
MEM_Usage/Limit	10.51MiB / 14.68GiB	21.85MiB / 14.68GiB
NET I/O	12.9GB / 2.23GB	98.8GB / 5.85GB
PIDs	50	8

The table 4 presents how the container host computer infrastructure is behaving for host resources while running Docker engine and other embedded services.

Table 4: Host computer resource usage for container host

Host computer measurement	**Measurement value of single host computer**
CPU% (for idle states)	0.9%
Memory%	0.4%
PIDs	197

For further evaluation purposes, the same MySQL and PostgreSQL RDBMS services were launched on a separate computer instance. For that computer instance was with the same configurations of the Docker hosted computer infrastructure. As well, the same databases were used for the computer instance-oriented study. This case was named as the β case. The table 5 presents the measurement values for the computer host for the β case for the idle state of the RDBMSs.

Table 5: Host computer resource usage for case β

Host computer measurement	**Measurement value of single host computer**
CPU% (for idle states)	2.1%
Memory%	2.7%
PIDs	124

According to table 4 and table 5, Docker host computer infrastructure is consuming lower resource usage from the host computer. However, when compared with Docker host and case β host, slightly lower usage is for Docker host.

D. Docker conatiner expanding & shrinking

Docker containers are running by using minimal resources from the host computer infrastructure. For any such kind of heavy processes, Docker containers consume higher resources from the host computer infrastructure. Within the experimental study, for data retrieve operations, container expanding and shrinking was visualized.

For the MySQL RDBMS container, four hundred forty-eight thousand data records were retrieved. The fig. 2 presents CPU usage: before data retrieval, while data retrieving and after retrieving the data.

In the fig. 2, the x-axis presents the time in the GMT +5.30-time zone and the y-axis presents the CPU usage as the percentage.

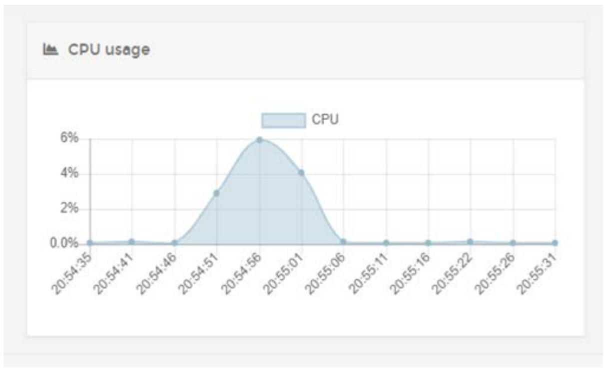

Fig. 2: CPU usage for data retrieving for MySQL container

Up to the peak point of the graph (20:54:56), the container was expanded for the operation of data retrieval. After generating the results, the container was shrunken.

For the PostgreSQL RDBMS container, three hundred thousand data records were retrieved. The fig. 3 presents the CPU usage: before data retrieval, while data retrieving and after retrieving the data for the PostgreSQL Docker container.

In fig. 3, the x-axis presents the time in the GMT +5.30-time zone and the y-axis presents the CPU usage as the percentage.

Up to the peak point of the graph (21:57:43), the container was expanded for the operation of data retrieval. After generating the results, the container was shrunk the same as for the MySQL Docker container.

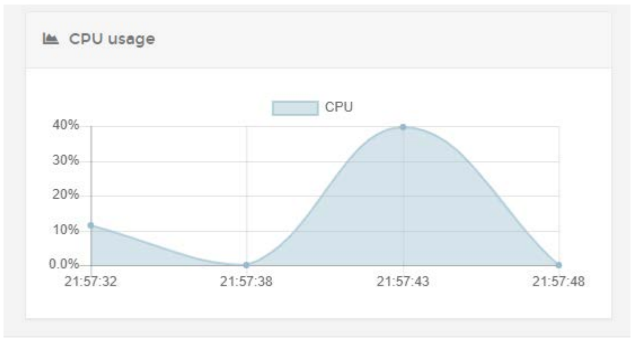

Fig. 3: CPU usage for data retrieving for PostgreSQL container

According to the fig. 2 and fig. 3, the two graphs are with few differences due to the internal architectural differences of database management engines.

Most scholarly articles were presented that the Docker containers are shrinking and expanding while shipping and operating on the container engine. Hence above graphical representation of the container expanding and shrinking are the most theoretical proof of the container stretching feature.

E. Feature differentiate between Approach 1 & 2

Within the experimental evaluation, to launch the RDBMS Docker container, two approaches were applied. Those are presented in Approach 1 and Approach 2 under section II. The table 6 presents the feature to differentiate between the launched container approaches for the Approach 1 & 2.

Table 6: Feature difference between approach 1 & 2

Approach 1	**Approach 2**
Difficult to lauch	Easy to launch
Easy to cutomize the installation	Customized installation is difficult
Need to install external dependencis	No need to install external dependencies

IV. CONCLUSION

Docker containers are high trending computer infrastructure technology. To launch any software service or application is possible on the Docker based infrastructure. After launching the Docker engine on the Ubuntu host computer, MySQL and PostgreSQL RDBMS Docker containers were launched within two networks separately to obtain answers to the pre-defined research questions.

The experimental platform was with host computer perspective, Docker engine perspective and container perspectival different data storage mechanisms. Those are with mounting block volumes for the host computer and Docker data volumes for Docker containers. Therefore, those aspects are answered for the RQ1. The experimented setup was with different secured approaches for ports, network and data directory perspectival. Therefore, it denotes that, the established platform is with more secured approach. Hence those are answered for the RQ2.

For the Docker container perspectival, those containers used only limited and minimal resources from the host computer infrastructure. As well, only for the higher operations, the containers expanded and other idle states, containers were shrunken. Therefore, those are answered for the RQ3. To deploy service on Docker, image usage or deploying the service from scratch on an operating system container are possible. Most of the pros and cons are available for both mechanisms and those are answered for the RQ4.

Docker containers are currently used mostly for testing and deployment of software applications. But today the world is moving with data science, image processing, artificial intelligence, Internet of Things and etc. for containerization. Therefore, containers will play a major role in the Information Technology era.

REFERENCES

[1] John Paul Martin, A. Kandasamy, and K. Chandrasekaran. 2018. Exploring the support for high performance applications in the container runtime environment. Hum.-centric Comput. Inf. Sci. 8, 1, Article 124 (December 2018), 15 pages. DOI:https://doi.org/10.1186/s13673-017-0124-3

[2] B. I. Ismail et al., "Evaluation of Docker as Edge computing platform," 2015 IEEE Conference on Open Systems (ICOS), 2015, pp. 130-135, doi: 10.1109/ICOS.2015.7377291.

[3] 8 surprising facts about real Docker adoption, 2021. [Online]. Available: https://www.datadoghq.com/docker-adoption/. [Accessed: 25- Jun- 2021].

[4] Empowering App Development for Developers | Docker, Docker, 2021. [Online]. Available: https://www.docker.com/. [Accessed: 25- Jun- 2021].

[5] Database Management System Tutorial - Tutorialspoin, Tutorialspoint.com, 2021. [Online]. Available: https://www.tutorialspoint.com/dbms/index.htm. [Accessed: 25- Jun- 2021].

[6] SQL - RDBMS Concepts - Tutorialspoint, Tutorialspoint.com, 2021. [Online]. Available: https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm. [Accessed: 25- Jun- 2021].

[7] docker stats, Docker Documentation, 2021. [Online]. Available: https://docs.docker.com/engine/reference/commandline/stats/. [Accessed: 25- Jun- 2021].

[8] F. Paraiso, S. Challita, Y. Al-Dhuraibi and P. Merle, "Model-Driven Management of Docker Containers," 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), 2016, pp. 718-725, doi: 10.1109/CLOUD.2016.0100.

[9] J. Stubbs, W. Moreira and R. Dooley, "Distributed Systems of Microservices Using Docker and Serfnode," 2015 7th International Workshop on Science Gateways, 2015, pp. 34-39, doi: 10.1109/IWSG.2015.16.

[10] Peinl, R., Holzschuher, F. & Pfitzer, F. Docker Cluster Management for the Cloud - Survey Results and Own Solution. J Grid Computing 14, 265–282 (2016). https://doi.org/10.1007/s10723-016-9366-y

[11] D. Liu and L. Zhao, "The research and implementation of cloud computing platform based on docker," 2014 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing(ICCWAMTIP), 2014, pp. 475-478, doi: 10.1109/ICCWAMTIP.2014.7073453.

[12] M. T. Chung, N. Quang-Hung, M. Nguyen and N. Thoai, "Using Docker in high performance computing applications," 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE), 2016, pp. 52-57, doi: 10.1109/CCE.2016.7562612.

[13] M. G. Xavier, I. C. De Oliveira, F. D. Rossi, R. D. Dos Passos, K. J. Matteussi and C. A. F. D. Rose, "A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds," 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2015, pp. 253-260, doi: 10.1109/PDP.2015.67.