Technium Vol. 4, No.10 pp.31-43 (2022)
ISSN: 2668-778X
www.techniumscience.com

A Lightweight Portable Multithreaded Client-Server Docker Containers

Awse S. Mahmood Alsaffar[1], Ayad Hussain Abdulqader[2]

1*Computer Science and Mathematics, Mosul, Mosul, Iraq

2*Computer Science and Mathematics, Mosul, Mosul, Iraq

1*awse.21csp64@student.uomosul.edu.iq 2*ayad_alezzi@uomosul.edu.iq

Abstract. In terms of diversity in operating systems, environments, and platforms, and with limited host infrastructure resources to hold all operating systems and platforms, the need arises to design applications that run in many or we can say in all operating systems and platforms.

This paper deals with designing and implementing a lightweight multithreaded client-server application appears as a separate Docker container. Both client and server containers are based on alpine Linux and developed using Python programming language. The execution unit in the containers is Python program files. As a case study, the server acts as a Wikipedia server and it can serve many clients simultaneously.

The Docker will build the containers depending on a writing Dockerfile for each container and push them to the registry (docker hub). When pulled the image of the containers from the registry account, then it could be run the container on a host. The proposed containerized multithreaded client-server model will become a portable with limited capabilities other than using Virtualization. Resources and cost requirements to achieve portability is evaluated for both virtualization and containerization paradigms. The results showed the superiority of containerization over the virtualization in both resources and cost requirements

Keywords. Client-Server model, Portability, Virtualization, Containerization, Docker platform.

1: Introduction:

Developers realize that the process of developing a particular application may differ somewhat developing on the environment in which the application will be executed. Therefore, they must provide all requirements of the environment in which the application running before starting to develop it.

The process of providing operating environment requirements burdens Developers' work as well as time and cost. Therefore, developers always strive to build portable applications that can be run in any environment to save time and cost.

In the last few years, specifically in 2013, Kamel Founadi, Solomon Hykes, and Sebastien Pahl in Santa Clara at PyCon corporation released a platform that can develop applications as containers that can run in any operating environment. This platform is called Docker [1].

Docker is an open platform for developing, dispatching, and executing applications. Docker lets to separate applications from your infrastructure so you can provide software quickly [2].

In this research, the technique provided by Docker will be used to build portable lightweight multithreaded client-server applications that can run on any operating environment with minimum resources requirements. Section 2 discusses the literature and concepts related to the building of a Portability application including Virtualization and Containerization. Section 3 presents the theoretical and background concept regarding the building of portable application as containers using Docker platform. Section 4 covers the implementation details of the proposed project. Section 5, discusses the testing and evaluation of the proposed project regarding the requirements of resources and cost. Finally, Section 6 presents the conclusions and recommendations for future research.

2. Literature Reviews

In previous decades, researchers try to find methods, techniques, or tools to enable software to work in different environments (operating systems). Their works can be classified into two categories: Virtualization, and Containerization. This section reviews some of the literatures about these categories in subsections 2.1, 2.2, receptively.

2.1: Virtualization:

Virtualization is a technique that is used for splitting an infrastructure's host resources among OSs. Virtual Machines begins in 1964 with IBM. Top of infrastructure hardware and below the operating system or virtual machines there is a layer called Virtual Machine Monitor (VMM) or a Hypervisor. This is a virtual software layer responsible for monitoring and managing the created virtual machines (VMs) [3] – [6].

Yukihiko Sohda, et al., 2001 proposes a software architecture achieve portability Java on Distributed Shared Memory (DSM). By using Java called (JDSM). Java considers a language that has full code-level portability and runtime optimization technologies. This study tried to investigate two potentials: one is using dynamic program editing systems to customize the program offline or at load time, and two is employing JIT compilers to perform. JIT compile time customization cloud also use JDSM as the backend to Java OpenMP [7].

Mark F. Mergen, et al., 2006 Proposed Virtualization for high-performance computing, this study discusses the hardware virtual virtualization trends, motivations, and problems with a focus on their value in High Computing Performance (HCP) environments.

Virtualization has the same possibility to benefit HCP applications in the dimensions of flexible OS diversity, productivity, performance, reliability, availability, security, and simplicity [8].

Fei Richard Yu, 2018 Proposes Virtualization for Distributed Ledger Technology (vDLT), This study presents a novel virtualization approach to addressing challenges in the current DLT systems, the basic resources (hardware, compute, storage, network, and so on) abstracted.

By providing a logical view of resources vDLT can considerably enhance performance, promote system evolution and simplify DLT management and configuration. And presents the architecture of virtualization for DLT [9].

In 1998 VMware company releases a VMware platform [10]. VMware platform is a hypervisor software that runs on Windows, Linux, and macOS. Installed VMware on the host infrastructure operating system and it allows multiple Virtual Machine (VM) can run its own operating system. VMware makes resources (CPU, Memory, HHD, and so on) dividing easy and flexible.

Virtual Box was created by Innotek GmbH, which was acquired by Sun Microsystems in 2008 [11]. Virtual Box can be installed on Windows, Linux, macOS, and other operating systems. It's a hypervisor software installed on the host's hardware operating system. VirtualBox is a multi-platform and open-source virtualization tool that let us create Virtual Machines. Every VirtualBox VM in turn can guest a particular operating system.

2.2: Containerization:

After using virtualization, some issues appear such as resource limitations, insecurity, and isolation. We can consider Containerization as the next virtualization, if we want to run some application no need to install the required OS and the dependencies, just we need to package the program with the operating system libraries and dependencies -these are required to run the application in any environment- in one container this container can be shipped and download it anywhere and anywhen [12].

Daniel Nüst, et al., 2020 Proposed packages and applications for Containerization with R, this research surveys downstream created upon the Rocker project images and achieves the current case of R packets for managing Docker images and controlling containers. Rocker project, which provides a widely-used suite of Docker images with customized R environments for particular tasks [13]. Dockerfile and Docker are the favourite approaches for cooperation between roles in an organization such as developers and IT drivers and between participants in knowledge sharing [14].

Christopher Bell, et al., 2020 Proposed developing single-use server containers. This study is about creating a one-to-one client-server model to defend against both persistent and unclear attacks by deputies. The system becomes possible by taking the advantage of the ability of Docker containers. An attack shuts down a single container without influencing the service of other containers or the host system. The study could not be tested within a more realistic backdrop [15].

Samuel P. Mullinix, et al., 2020 Surveys security measures for containerized applications imaged with Docker, in this survey reviewed the case of the security situation for the most famous container systems currently: Docker. Look at its origins from Linux technologies embedded in the operating system itself; look at intrinsic vulnerabilities like the implementation of Docker images, and provide an analysis of current tools and modern methodologies used in the field to assess and improve its security. Docker images are used to save and deploy programs [16].

Alowolodu Olufunso Dayo, 2021 Propose a multi-containerized application using Docker containers and Kubernetes clusters, in this research speaks about the development and deployment of applications in manufacturing and development environments that posed a huge problem. The deployed software is delivered at a slower pace with higher maintenance costs. Docker and Kubernetes to attenuate against this and it will be tested with the Fibonacci sequence calculator. This program uses Docker-compose to communicate the program's containers [17].

Balqees Talal Hasan Agha, 2022 Proposes improving parallel schedulers in heterogeneous distributed computing for containerized Big Data. This study proposed and implemented a new system which is named as Hetero-Kubernetes, which is categorize computational nodes in a cluster into two sub-clusters. Dealing with the big data framework and IoT applications through the container environment has provided several benefits, the most important of which are portability, scalability, and efficient use of computing resources [18].

3. Building portable software using Docker platform

This section is talking about building portable software as containers using Docker platform. Docker is an open platform used for developing, dispatching, and executing applications. Docker lets you to separate applications from your infrastructure so you can provide software quickly [19]. The following subsections explains, the Docker platform architecture and all its parts, how to make a docker image from a particular program language script, and how to produce a Docker container.

3.1: Docker components:

Docker is made up of four fundamental components [20]:

"Docker Client and Docker Server": The request from the Docker client is processed by the Docker server and then processes it correspondingly, the Docker client might be (the Command line or Docker Desktop). Docker daemon/server and Docker client can run on the same host, or a local Docker client can connect to a remote daemon or server running on a different host [21].

"Docker Images": The world of Docker is made up of images. You run your containers from the images.

"Docker Registries": The images you build are stored in registries by Docker. Registries fall into two categories: private and public. The common public registry is www.hub.docker.com [22].

Technium Vol. 4, No.10 pp.31-43 (2022)
ISSN: 2668-778X
www.techniumscience.com

"Docker Containers": they encapsulate a program as a single runnable package of application that packages a script code with all of the requirement files, libraries, and dependencies that are necessary for it to run [23].

3. 2: Docker Architecture:

Docker uses a "client-server architecture". The Docker daemon, which builds, runs, and distributes your Docker containers, is contacted by the Docker client. You can connect a Docker client to a remote Docker daemon, or the Docker client and daemon can run on the same operating system [24]. The architecture of Docker makes it possible to share resources with the OS kernel and access data within containers. [25].

Figure (1) below shows Docker architecture [24]:

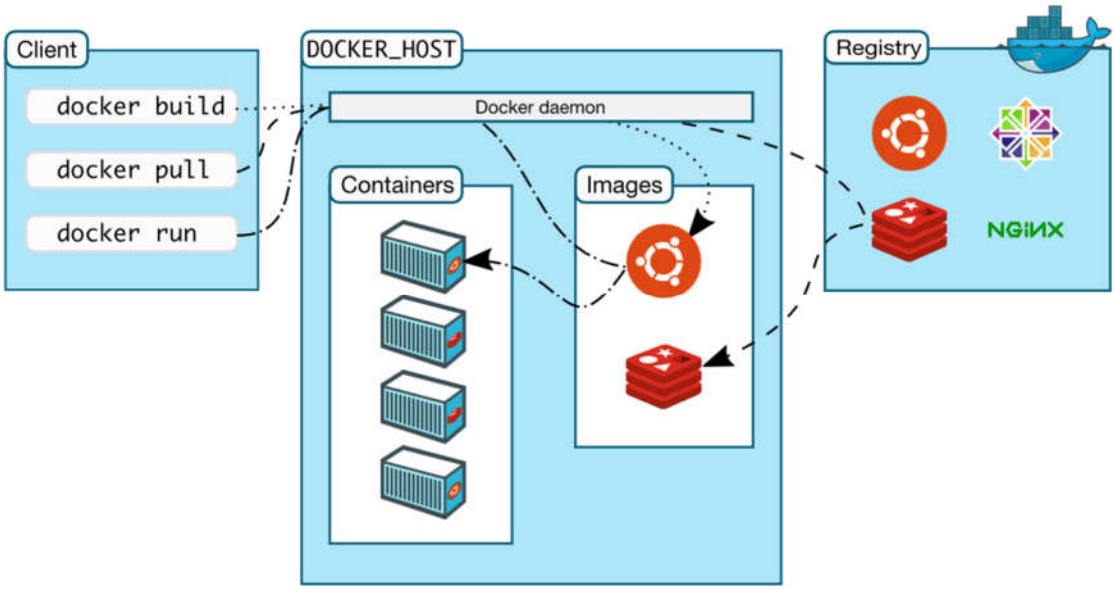


Figure (1) Docker architecture

3.3: How Docker works:

To work on Docker, should be taken care of:

- Should install Docker on the host whether physical host or virtual host, and should be attentive to the operating system on our host whether Linux, Windows, or macOS; due to each operating system has a special way to install Docker https://docs.docker.com/get-docker/ [26].
- On the terminal or Command prompt download the images which are wanted by the instruction (docker pull image-name).
- If a user is wanted to make a user image, he should write a Dockerfile. Docker file is a text file including an instruction to create a docker image, each line of the instructions is called a layer [27].
- Create a Docker image from the program script and the other requirements by (docker build) command.
- Finally, run this image by (docker run) command to create a container that considers the executable case of the Docker image and the program will be executed with all the requirements it needs [28]. The diagram in figure (2) shows the steps of creating a particular container using Docker platform.

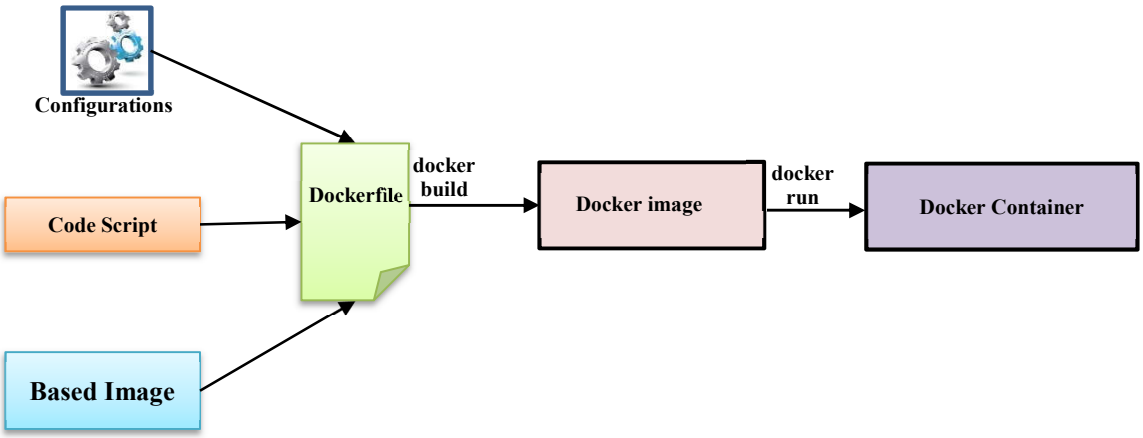


Figure (2) Building a particular container using Docker platform

4. Implementation of the proposed system:

To build an application as a Docker container, a Docker image should be built first. The Docker image contains a set of instructions, these instructions act as a template and each run of these instructions will produce a new Instance of a Docker container.

Building Docker images requires constructing a "Dockerfile". A "Dockerfile" is a text file that consist of a sequence of commands. Each command in the dockerfile is called a Layer [29].

Usually, the working mechanism of any client-server model is for the server to serve a number of clients at the same time, as shown in the figure (3). The proposed system consists of two containers: Server container and client container. We will talk about the details of building both the server container and the client container in subsections 4.1 and 4.2 respectively.

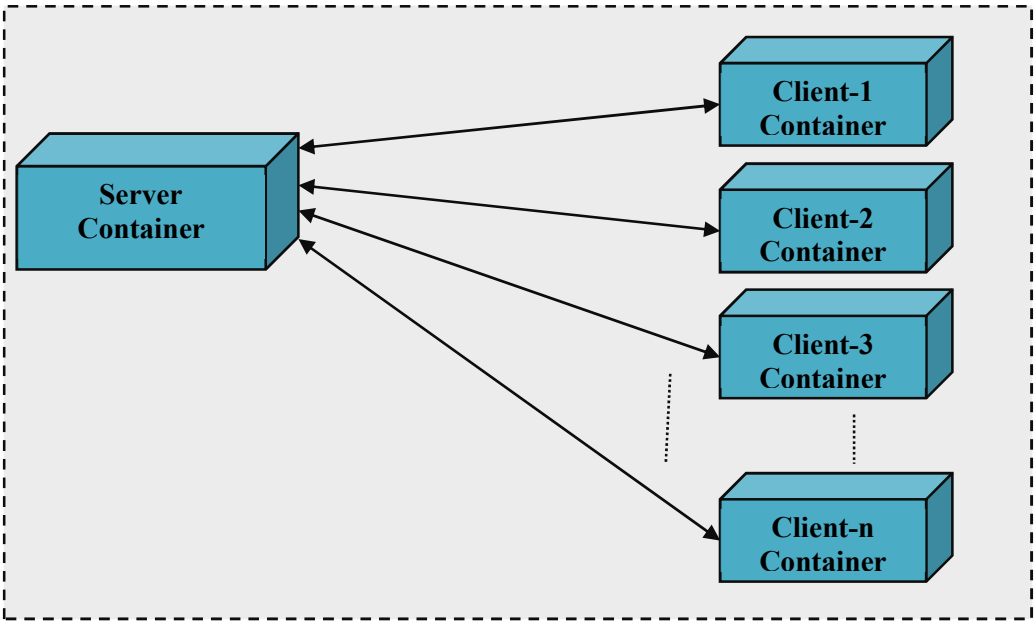


Figure (3) Multithreaded Client-Server Model

4.1. Implementation of the Server container.

This subsection explains the server container implementation which is written in Python programming language version (3.10). The server container consists of two main units: the connection unit and the serving unit as shown in figure (4).

Server container as well as client container are based on Linux Alpine and the connection between them depends on the Docker network.

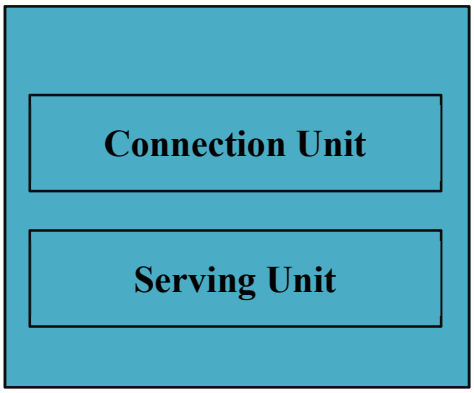


Figure (4) the server container components

The connection unit is a multithreaded TCP server socket, used to accept a client request and respond to that request. The serving unit contains a serving process that is designed to be any serving process that can be developed for a particular service, in this project, the serving contains a wikipedia module that takes the request as a word or a sentence from the specific client to the Wikipedia website by the server and brings a summary of that word or sentence to the client.

building Dockerfile for a server image follows the steps:

Step1: Using FROM command to define the based image. In this case, the Linux alpine 3.16 is the base image.

Step2: Installing all program dependencies which include the following:

1.Update the base image (alpine image) using the RUN command as follows: *RUN apk update*

2.Installing the Python 3.10 package as follows: *RUN apk add python3 py3-pip*

3.Upgrading the pip tool as follows: *RUN pip3 install - -upgrade pip*

4.Installing the Socket Module as follows: *RUN pip3 install sockets*

5.Installing the Wikipedia Module as follows: *RUN pip3 install wikipedia*

Step3: make the image directory like this: *RUN mkdir serverimage*

Step4: copy the server program file to the *serverimage* directory as follows:

COPY server.py serverimage/server.py

Step5: setting *serverimage* directory as a working directory like this: *WORKDIR /serverimage*

Step6: Determine the execution command ENTRYPOINT of execution of the container as follows:

ENTRYPOINT ["python3", "server.py"]

The resulting Dockerfile for building the server image is shown in figure (5):

```
# Get the Latest Base Image for the Dockerfile...
FROM alpine:3.16
# Install all Program Dependencies
RUN apk update
RUN apk add python3 py3-pip
RUN pip3 install --upgrade pip\
                wikipedia\
                sockets
# Make the Image Directory…
RUN mkdir serverimage
# Copy all Files to the Server Container...
COPY server.py serverimage/server.py
# Set a Directory for the Program…
WORKDIR /serverimage
# Run the Command…
ENTRYPOINT ["python3", "server.py"]
```

Figure (5) Dockerfile of a server image

4.2. Implementation of the Client container.

This subsection explains the client container implementation which is written in Python programming language version (3.10) too. The client container consists of two main units: the connection and the manipulation unit, Figure (6), the connection unit is the TCP socket that sends requests to the server to be served, and the server back the response to the client through the open channel between them. The manipulation unit manipulates (or just viewing) the server's response.

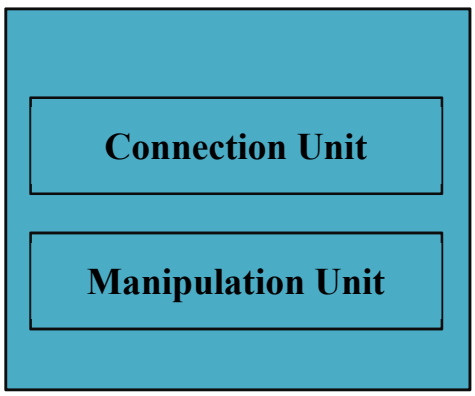


Figure (6) the client container components

Building Dockerfile for a client image follows the steps:

Step1: Using FROM command to define the based image. In this case, the Linux alpine 3.16 is the base image.

Step2: Installing all program dependencies which include the following:

1.Update the base image (alpine image) using the RUN command as follows: *RUN apk update*

2.Installing the Python 3.10 package as follows: *RUN apk add python3 py3-pip*

3.Upgrading the pip tool as follows: *RUN pip3 install - -upgrade pip*

4.Installing the Socket Module as follows: *RUN pip3 install sockets*

Step3: make the image directory like this: *RUN mkdir clientimage*

Step4: copy the client program file to the *clientimage* directory as follows:

COPY client.py clientimage/client.py

Step5: setting *clientimage* directory as a working directory like this: *WORKDIR /clientimage*

Step6: Determine the execution command ENTRYPOINT of execution of the container as follows:

ENTRYPOINT ["python3", "client.py"]

The resulting Dockerfile for building the client image is shown in figure (7):

```
# Get the Latest Base Image for Python...
FROM alpine:3.16
# Install all Program Dependencies
RUN apk update
RUN apk add python3 py3-pip
RUN pip3 install sockets
# Make the Image Directory...
RUN mkdir clientimage
# Copy all Files to the Client Container...
COPY . /clientimage
# Set a Directory for the Program...
WORKDIR /clientimage
# Run the Command...
ENTRYPOINT ["python3", "client.py"]
```

Figure (7) Dockerfile of wiki_client:0.0.1 image

5. Evaluation of resources and cost requirements

5.1 Resources Requirements

This subsection presents the comparison of hardware and software resources requirement between virtualization and containerization of building multithreaded client-server applications using VMware and docker platform respectively as shown in Table 1 [30]. The results showed that the using of containerization as a way to produce portable applications requires less resources than we would need if we used visualization.

Table (1) Resources requirements Comparison (VMware vs. Docker platform)

Requirements	Virtualization requirements for proposed project	Containerization requirements for proposed project
Memory	128 MB for alpine	Docker needs 36 MB
	40 MB VMware	
	74 MB Client and Server Python programs	
	242 MB (total)	
Storage	20 GB alpine	2.4 GB Docker
	1 GB Python	75 MB the client container
	1 GB VMware	89 MB the server container
	22 GB (total)	2.5 GB (total)

5.2. Cost Requirements

The code of proposed project will be estimated using basic COCOMO (Constructive Cost Model model) to find the cost of the time and the effort.

First will calculate the basic COCMO model without using Docker containers the number of Lines of Codes to implement the proposed project on (Windows, macOS, and Linux) is (288 LOC) so (KLOC = 0.228). so, thus:

Organic:

Effort = a (KLOC) ^{b}

Effort = 2.4 (0.228) $^{1.05}$

Effort ≈ 0.65 Person-Month

Time = c (Effort) ^{d}

Time = 2.5 (0.65) $^{0.38}$

Time ≈ 2.1225 Months

Person required = Effort/Time

Person required = 0.65/2.1225

Person required ≈ 0.31 Persons

Second will calculate the basic COCOMO model with using Docker containers the number of Lines of Codes to implement the proposed project is (76 LOC) so (KLOC = 0.076) [31]-[32].

Organic:

Effort = a (KLOC) ^{b}

Effort = 2.4 (0.076) $^{1.05}$

Effort ≈ 0.16 Person-Month

Time = c (Effort) ^{d}

Time = 2.5 (0.16) $^{0.38}$

Time ≈ 1.25 Months

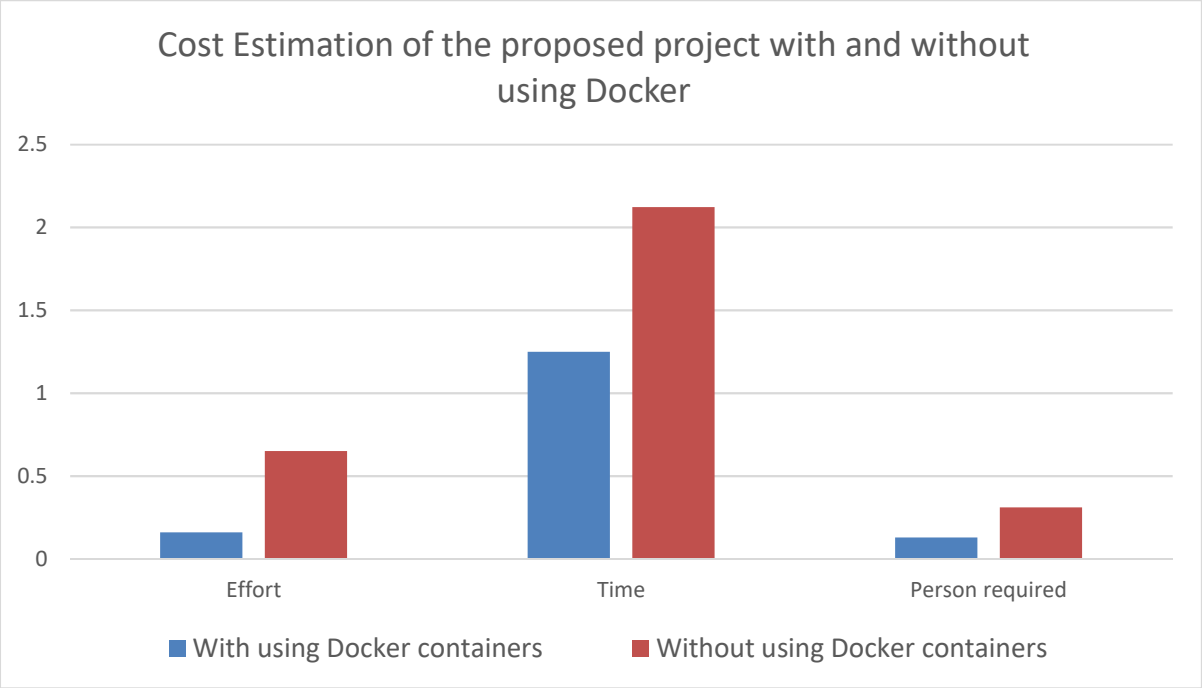
Person required = Effort/Time

Person required = 0.16/1.25

Person required ≈ 0.13 Persons

Table (2) COCOMO model based on Organic system type with and without using Docker containers

System types	With using Docker containers	Without using Docker containers
Effort	0.16	0.65
Time	1.25	2.1225
Person required	0.13	0.31



6. Conclusions and Future Work

The issues raised and discussed in the current study are summarized in this section. Additionally, recommendations for future research that could improve the proposed project are provided in this section.

The current study presents a portable Multithreaded Client-Server Application as containers that can be executed in any execution environment. The use of containers as a way to produce a portable application gives more benefits if we used Virtualization, as becomes possible to run such an application on a computer with limited capabilities.

The proposed project has been developed using the Docker platform with Python programming language. The API of the Python socket module is to create a connection between the nodes. Also, the API of the Python threading module is used in building a multithreaded server that can handle many clients simultaneously.

The current project has generated many ideas that have opened up some interesting avenues for further research in the area of client-server applications. Such that building a general Docker image of the client-server model that can be configured and adapted for use for a particular application will give software companies a great benefit in reducing the cost as well as the time in producing their software product.

References:

[1] "Docker (software) - Wikipedia." https://en.wikipedia.org/wiki/Docker_(software) (accessed Aug. 10, 2022).

[2] "Docker overview | Docker Documentation." https://docs.docker.com/get-started/overview/ (accessed Aug. 05, 2022).

[3] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: a Performance Comparison."

[4] F. Rodríguez-Haro et al., "A summary of virtualization techniques," Procedia Technology, vol. 3, pp. 267–272, 2012, doi: 10.1016/j.protcy.2012.03.029.

[5] S. Nanda and T.-C. Chiueh, "A Survey on Virtualization Technologies."

[6] A. Bhardwaj and C. R. Krishna, "Virtualization in Cloud Computing: Moving from Hypervisor to Containerization—A Survey," Arab J Sci Eng, vol. 46, no. 9, pp. 8585–8601, Sep. 2021, doi: 10.1007/s13369-021-05553-3.

[7] Y. Sohda, H. Nakada, and S. Matsuoka, "Implementation of a Portable Software DSM in Java," 2001.

[8] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis, "Virtualization for High-Performance Computing."

[9] F. R. Yu, J. Liu, Y. He, P. Si, and Y. Zhang, "Virtualization for Distributed Ledger Technology (vDLT)," IEEE Access, vol. 6, pp. 25019–25028, Apr. 2018, doi: 10.1109/ACCESS.2018.2829141.

[10] "VMware Docs All Products." https://docs.vmware.com/allproducts.html (accessed Oct. 06, 2022).

[11] "VirtualBox for Windows - Download it from Uptodown for free." https://virtualbox.en.uptodown.com/windows (accessed Oct. 06, 2022).

[12] A. Abuabdo and Z. A. Al-Sharif, "Virtualization vs. containerization: Towards a multithreaded performance evaluation approach," in Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA, Nov. 2019, vol. 2019-November. doi: 10.1109/AICCSA47632.2019.9035233.

[13] C. Boettiger and D. Eddelbuettel, "An introduction to Rocker: Docker containers for R," R Journal, vol. 9, no. 2, pp. 527–536, Dec. 2017, doi: 10.32614/RJ-2017-065.

[14] D. Nüst et al., "The Rockerverse: Packages and Applications for Containerization with R," Jan. 2020, doi: 10.32614/RJ-2020-007.

[15] N. Odell and C. A. Shue, "Developing Single Use Server Containers A Major Qualifying Project," 2020.

[16] S. P. Mullinix, E. Konomi, R. D. Townsend, and R. M. Parizi, "On Security Measures for Containerized Applications Imaged with Docker."

[17] A. O. Dayo, "A Multi-Containerized Application using Docker Containers and Kubernetes Clusters," Int J Comput Appl, vol. 183, no. 44, pp. 55–60, Dec. 2021, doi: 10.5120/ijca2021921843.

[18] T. in and D. Basheer Abdullah Albazaz, "Improving Parallel Schedulers in Heterogeneous Distributed Computing for Containerized Big Data Balqees Talal Hasan Agha."

[19] T. Scheepers, "Virtualization and Containerization of Application Infrastructure: A Comparison," 2014.

[20] Rad, B. B., Bhatti, H. J., & Ahmadi, M. (2017a). An Introduction to Docker and Analysis of its Performance Metamorphic Malware CLassification using MLP Neural Network View project An Introduction to Docker and Analysis of its Performance. In IJCSNS International Journal of Computer Science and Network Security (Vol. 17, Issue 3). https://www.researchgate.net/publication/318816158

[21] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux j*, *239*(2), 2.

[22] Turnbull, J. (2017). The Docker Book: Containerization is the new virtualization. James Turnbull.

[23] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An Introduction to Docker and Analysis of its Performance Metamorphic Malware CLassification using MLP Neural Network View project An Introduction to Docker and Analysis of its Performance," 2017. [Online]. Available: https://www.researchgate.net/publication/318816158

[24] Docker overview | Docker Documentation. (n.d.-b). Retrieved August 14, 2022, from https://docs.docker.com/get-started/overview/#docker-architecture

[24] S. B. Shankar and S. N. Shankar, "DOCKER CONTAINER 1 2 3," 2015.

[25] Chung, M. T., Quang-Hung, N., Nguyen, M. T., & Thoai, N. (2016b). Using Docker in high performance computing applications. 2016 IEEE 6th International Conference on Communications and Electronics, IEEE ICCE 2016, 52–57. https://doi.org/10.1109/CCE.2016.7562612

[26] Get Docker | Docker Documentation. (n.d.). Retrieved August 14, 2022, from https://docs.docker.com/get-docker/

[27] Bhat, S. (2018b). Practical Docker with Python. In Practical Docker with Python. Apress. https://doi.org/10.1007/978-1-4842-3784-7

[28] Sabharwal, N. (n.d.). Hands on Docker. Packt Publishing.

[29] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An Introduction to Docker and Analysis of its Performance Metamorphic Malware CLassification using MLP Neural Network View Project An Introduction to Docker and Analysis of its Performance," 2017. [Online]. Available: https://www.researchgate.net/publication/318816158

[30] "Minimal Hardware Requirements | Wikipedia"

https://wiki.alpinelinux.org/wiki/Requirements / (accessed Nov. 01, 2022).

[31] Goyal, Somya, and Anubha Parashar. "Machine learning application to improve COCOMO model using neural networks." International Journal of Information Technology and Computer Science (IJITCS) 3 (2018): 35-51.

[32] R. K. Sachan and D. S. Kushwaha, "Anti-Predatory NIA Based Approach for Optimizing Basic COCOMO Model," 2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence), 2020, pp. 710-715, doi: 10.1109/Confluence47617.2020.9058033.