IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

# Application Optimization in a Containerized Environment

[1]Karunakara Rai B, [2]Shashank R, [3]Sampreeth A V, [4]Greeshma B E, [5]Madhuri P k

[1]Associate Professor, NMIT, Bangalore,[2,3,4,5] Final year B.E students NMIT, Bangalore

[1,2,3,4,5]Nitte Meenakshi Institute of Technology, Bangalore-India

***Abstract:*** One of the major problems faced while running a third-party application locally is that the application works properly on the developer's system but it may show some error or may not be in the proper format while running on another device this is due to the absence of some dependencies which the developer may have used. This can be avoided by containerizing our application using Docker, Containerization provides us an isolated environment in which our images, which are packet consisting of the source code and other necessities and dependencies, run. To further improve the performance of the application, so as to avoid crashing of the application when a lot of users access it, optimization is done using Kubernetes cluster. In this paper we propose a methodology to improve the performance of a web-application in terms of load handling capacity so that the probability of crashing of the webapp due to over loading is reduced.

***Index Terms*** **- Docker, Kubernetes, minikube, Apache JMeter, kubectl.**

## I. INTRODUCTION

Docker is a set of containers as a service that helps us in creating and managing containers in which an image consisting of the source code for our application along with all the dependencies are present. Even after the containerization of an application, there is a possibility of it crashing when a large number of users try to access the application. Taking this into consideration we have further optimized the application using an open-source container-orchestration system called Kubernetes. In this modern era where time is money, our project provides a solution to avoid time-consuming tasks such as server maintenance and decreasing the probability of server crash or large buffer time by bringing it down to an acceptable margin by adopting cloud-native architecture and streamlining the application and further optimizing it using Docker and Kubernetes.

## II. RELATED WORK

A. Balalaie, A. Heydarnoori, and P. Jamshidi published Migrating to cloud-native architectures using microservices: An experience report giving reason to why even though cloud is popular it is not used to its full potential also discussing migrating a monolithic on-premise software program structure to microservices. Importance of non-stop handing over changed into highlighted [1]. N Kratzke and P C Quint published a systematic mapping study dealing with cloud native application engineering processes and defines what cloud-native software mean. Fehling et al. put forth that a cloud-native software must be IDEAL, so it must have a [I]solated state, is [d]istributed in its nature, is [e]lastic in a horizontal scaling way, operated through an [a]utomated control system and its additives must be [l]oosely coupled. Via this paper, we comprehend that

although the primary concerns on cloud-native programs originate from research, the term "cloud-native" appears to be widespread and used – perhaps even dominated – by the software and cloud enterprise nowadays [2]. M Noshy, A Ibrahim, and H A Ali published a paper discussing Real-time virtual machine (VM) migration is an essential virtualization feature that allows you to migrate virtual machines from one location. To another without stopping the VM. This document briefly introduces the concepts, advantages, and methods of real-time virtual machine migration [3]. A Aouat, E A Deba, A E H Benyamina, and D Benhamamouch published a comparative study analysing that Groups are seeking out answers to install an infrastructure that spans more than one times of public and personal clouds. despite the fact that there are exclusive answers proposed now no longer all issues of each customer will be met. This paper gives us a comparative observe among the maximum used answers solution [4]. Carlos Guerrero, Isaac Lera, Carlos Juiz published a case study in which a method to optimize the provision of microservice-based applications using containers in a multi-cloud architecture is proposed. Here the optimization goals are the cost of cloud services, the network latency between microservices, and the time to start new microservices when the provider is unavailable [5].

## III. OBJECTIVES

The objectives of the proposed work as stated as below

- To construct a cloud native application thereby overcoming constrains of on-premises architecture
- Containerizing the application thereby enabling it to run properly on all devices.
- Deployment of Application to Kubernetes Cluster for resource and memory management, thereby optimizing existing Container technology.
- Comparison of health metrics of Application before and after Optimization.

## IV. RESEARCH METHODOLOGY

The proposed methodology is shown below.

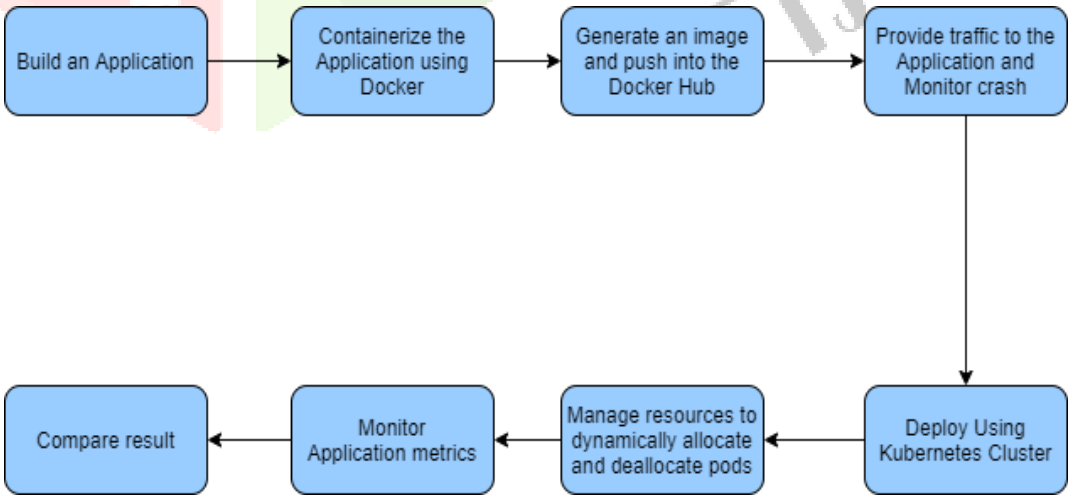



figure 1: proposed methodology

An application is built using HTML, JavaScript, and other software tools, this application is containerized using docker which creates an isolated environment, with all the required libraries and dependencies, for the application to run in. A Docker image is created by writing a docker file and pushed into the docker hub, through which we can access our containerized application to deploying or further development, using docker push command, figure 2 shows the containerized web app we built and deployed in our localhost port

8080.Now traffic is provided to the application and its performance is monitored in terms of load handling capacity using Apache JMeter which is an open-source software designed for load testing and monitoring the performance of a website.

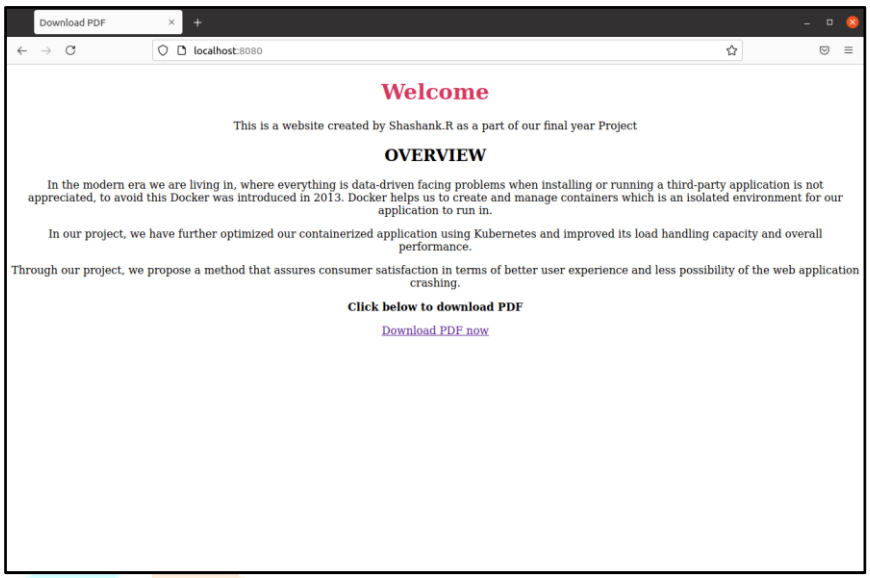


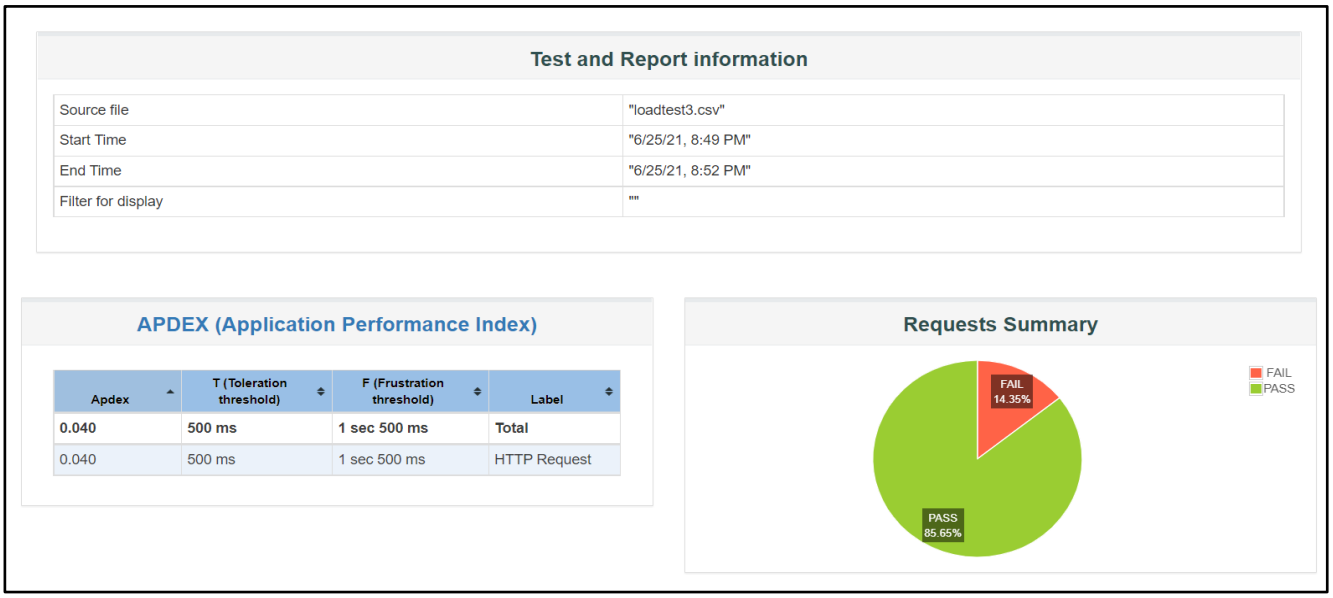

figure 2: app deployed in localhost after containerization

By installing kubectl and minikube and running it locally in our system and writing yaml file for deployment and service, the application is deployed in Kubernetes cluster, which is an open-source container orchestration platform that automates the dynamic allocation and deallocation of resources to pods, which are the copies or iteration, specified in yaml file, of the application deployed. Now the application is again tested using Apache JMeter by providing the same traffic, and the obtained result is compared with the previous one.

## V. RESULTS

The main goal of this proposed work is optimization of our containerized application in terms of load handling capacity and improving the overall performance from the user end perspective, the proposed methodology assures consumer satisfaction and avoids running into problems that may lead to crashing off the application leading to the loss of clients and in turn earning a bad name for the organization.

To provide evidence for the above-mentioned points we have conducted load-testing on the application we have built. To achieve this, we have used Apache JMeter and allocated 20,000 thread users to provide traffic for the application both before and after optimization. As shown in figure 3, before optimization there is a total of 14.35% error i.e., Out of 20,000 users, 2,870 users were unable to access the application due to high traffic which gave the error of localhost:8080 failed to respond, as shown in figure 4.



Test and Report information

| | |
|---|---|
| Source file | "loadtest3.csv" |
| Start Time | "6/25/21, 8:49 PM" |
| End Time | "6/25/21, 8:52 PM" |
| Filter for display | "" |

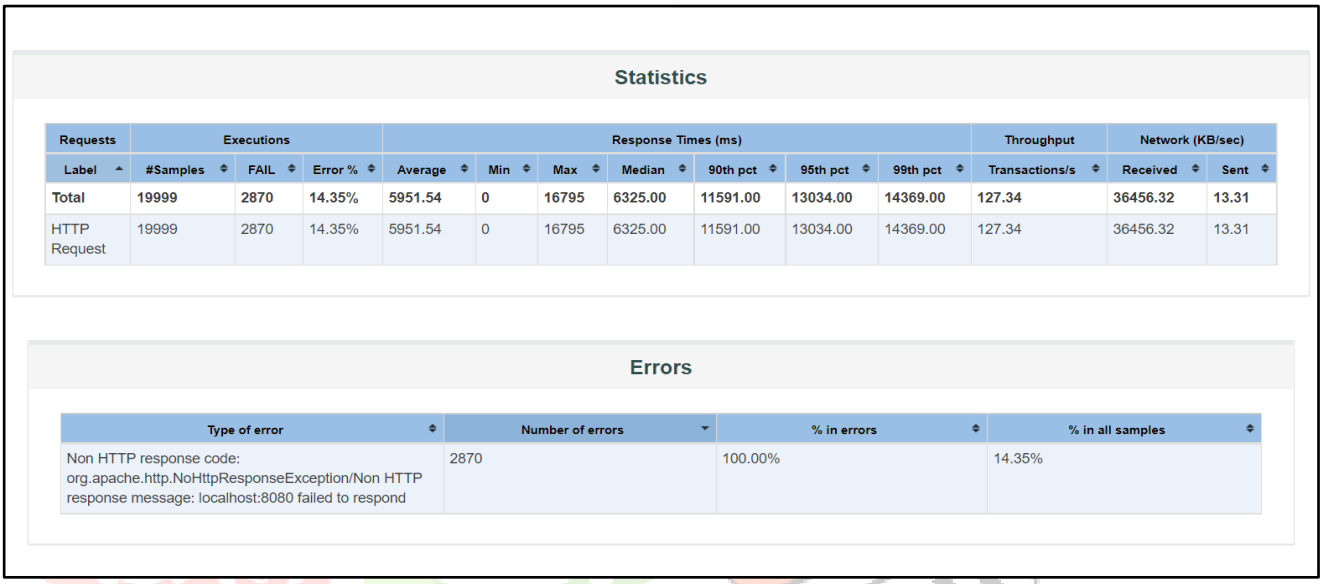
APDEX (Application Performance Index)

| Apdex | T (Toleration threshold) | F (Frustration threshold) | Label |
|---|---|---|---|
| 0.040 | 500 ms | 1 sec 500 ms | Total |
| 0.040 | 500 ms | 1 sec 500 ms | HTTP Request |

Requests Summary



figure 3: graphical test results before optimization



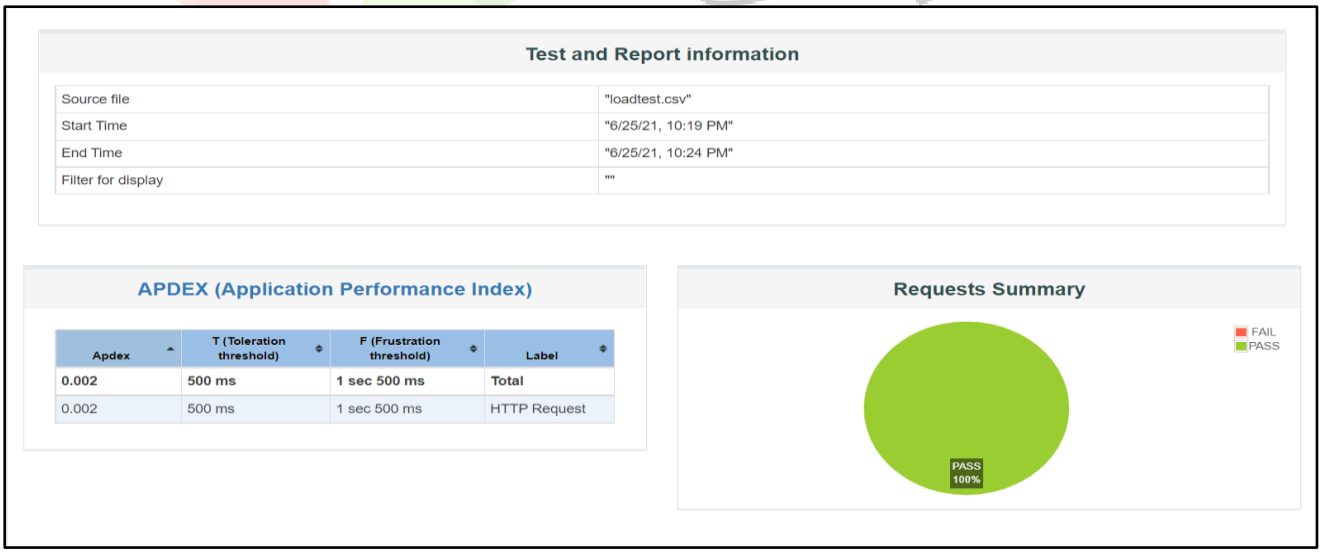
Statistics

| Requests | Executions | | | Response Times (ms) | | | | | | | Throughput | Network (KB/sec) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | #Samples | FAIL | Error % | Average | Min | Max | Median | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
| Total | 19999 | 2870 | 14.35% | 5951.54 | 0 | 16795 | 6325.00 | 11591.00 | 13034.00 | 14369.00 | 127.34 | 36456.32 | 13.31 |
| HTTP Request | 19999 | 2870 | 14.35% | 5951.54 | 0 | 16795 | 6325.00 | 11591.00 | 13034.00 | 14369.00 | 127.34 | 36456.32 | 13.31 |

Errors

| Type of error | Number of errors | % in errors | % in all samples |
|---|---|---|---|
| Non HTTP response code: org.apache.http.NoHttpResponseException/Non HTTP response message: localhost:8080 failed to respond | 2870 | 100.00% | 14.35% |

figure 4: error report



Test and Report information

| | |
|---|---|
| Source file | "loadtest.csv" |
| Start Time | "6/25/21, 10:19 PM" |
| End Time | "6/25/21, 10:24 PM" |
| Filter for display | "" |

APDEX (Application Performance Index)

| Apdex | T (Toleration threshold) | F (Frustration threshold) | Label |
|---|---|---|---|
| 0.002 | 500 ms | 1 sec 500 ms | Total |
| 0.002 | 500 ms | 1 sec 500 ms | HTTP Request |

Requests Summary



figure 5: graphical test results after optimization

After optimization where we used the Kubernetes cluster to deploy the application with 5 pods and after load-testing it by providing the same traffic as before, we can see that drastic is no error as shown in figure 5, this

is because as there are 5 replicas of the application the load is distributed evenly among all the 5 pods, thereby increasing the load handling capacity of the application.

## VI. CONCLUSION AND FUTURE SCOPE

Application containerization is an evolving technology which is dynamically changing the way developers wants to test and run the application instances within cloud. With less resource-intensive alternative for running application on a virtual machine because the application containers can only share computational resources and memory without requiring a full OS to underpin each application.

This project is as successful implementation of containerization of a web application and optimization of the same to improve the overall performance of the app in terms of load and stress handling capacity. HTML and CSS are used to create a web application and docker for containerizing and an open-orchestration system called Kubernetes for optimization. The load testing is carried out both before and after optimization using Apache JMeter. The results successfully show a significant improvement is the error rate i.e. after optimization using Kubernetes thereby improving the load handling capacity. This system can be further modified to support higher user traffic.

## REFERENCES

[1] R. C. Gonzalez and R. E. Woods, Digital Image Processing, Pearson Education Limited, 2014.

[2] M. S. A. Alexandro, L. R. Eduardo and S. Bampi, "Dynamically reconfigurable architecture for image processor applications," in *Proceedings of 36th Design Automation Conference*, USA, 1999.

[3] S. G. Fowers, D.-J. Lee, D. A. Ventura and J. K. Archibald, "The Nature-Inspired BASIS Feature Descriptor for UAV Imagery and Its Hardware Implementation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 23, no. 5, pp. 756 - 768, 2012.

[4] M. Fularz, M. Kraft, A. Schmidt and A. Kasiński, "A High-Performance FPGA-Based Image Feature Detector and Matcher Based on the FAST and BRIEF Algorithms," *International Journal of Advanced Robotic Systems*, vol. 12, no. 10, pp. 1-15, 2015.

[5] Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications by Carlos Guerrero, Isaac Lera, Carlos Juiz