Original Article

Deploying a Mongo DB Container for High Availability Query Processing using Novel Docker Container over Kubernetes Micro Service Architecture

Niranjan S[1], Saravanan M.S[2]

[1]Research Scholar, Department of Computer Science and Engineering, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, Tamilnadu. India, Pincode: 602105.
[2]Project Guide, Corresponding Author, Department of Computer Science and Engineering, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, Tamilnadu. India, Pincode: 602105.

Abstract

Aim: The aim of the study is to estimate the high availability of Query processing using Novel Docker Container over kubernetes micro service architecture. **Materials and Methods:** Sample groups that are considered in the project can be classified into two each has 20 samples, one for Microservice architecture and other for Novel Docker container, which are tested using 0.80 for G-power to determine the sample size and for t-test analysis. 20 queries have been used in each group for estimating high availability. **Results:** The Novel Docker container with efficient accuracy of 91.10%, which by far seems to be better than the kubernetes based micro service architecture which gives around 87.50%. As a result, the study group has a statistically negligible difference. The significance is around 0.039 ($p<0.05$) and therefore there is a statistical insignificant difference among the study group. **Conclusion:** The query processing speed in the Novel Docker container is better compared to kubernetes based micro service architecture.

Keywords: Novel Docker Container, Query processing, Kubernetes, Micro service architecture, Mongodb, High availability.

DOI: 10.47750/pnr.2022.13.S04.188

INTRODUCTION

Container technology developments have reignited interest in employing containers in the cloud (National Academies of Sciences, Engineering, and Medicine et al. 2020). Containerization of cloud infrastructure services, i.e. running infrastructure management software in containers, has recently gained traction (Mouat 2015). Containers, whether used for apps or cloud infrastructures, are appealing because they enable effective separation with low overhead and a quick start-up time, resulting in extremely agile solutions (Guthrie et al. 2014). Microservices are small applications that perform specific business functions and communicate with one another through APIs(Surianarayanan, Ganapathy, and Pethuru 2019). Traditional monolithic architectures, in which the application is a big and complex code base, can be addressed using the microservices architecture paradigm(Zhelev and Rozeva 2019). Containerization has cleared the path for new microservices-based software development approaches. The application is structured as a collection of loosely bound services in a microservice-based architecture. The capabilities of services are structured around them, and they communicate with one another through well-defined interfaces (Sahai and Graupner 2007). This is a departure from the common monolithic software design, in which old or "existing monolithic" programmes and services can be "Containerized" by breaking them down into microservices, each running on a single or several containers. Containers are one of the most important microservices enablers. Multiple replicas of individual services are used to achieve scalability and high availability in containerized systems(Sahai and Graupner 2007; De Santis et al. 2016). Containers are ephemeral, making their life cycles difficult to manage in large deployments(Baier and White 2018a). In a distributed context, it's more difficult to ensure their availability and replica count. Frameworks and tools for container orchestration make it easier to manage containers. Container orchestration frameworks offer features such as automated container deployment, scaling, resiliency, rescheduling failed containers, and container management (Baier and White 2018b; Arundel and Domingus 2019). You can make your application independent of the host environment using Docker. You can now wrap each of your microservices in a Novel Docker container because you have a microservices architecture. Microservice is the liquid that you pour into

Docker, which is a Cup or Container in other words. Different sorts of liquids can be poured into the same cup. However, it will become crowded. You can also run multiple Microservices in the same Novel Docker container (Jangla 2018).

There are around 57 IEEE papers and 61 google scholar papers have been published over the past 5 years. The most cited article is "Container and Microservice Driven Design for Cloud Infrastructure DevOps". In the industry, the microservices architectural style has gotten a lot of attention, and the transition to this design is well underway. This architectural style enables software applications to be built as a collection of loosely linked, independently deployable microservices.Because stateless microservices may be deployed as interchangeable instances, they are simple to reproduce. The same cannot be said for stateful microservices. Because stateful microservice instances have different states, they cannot be swapped.The most widely used NoSQL database is MongoDB. MongoDB stores data as BSON (Binary JSON) documents and is based on the Document Store data paradigm. MongoDB has a schema-less storage format that allows various entries to have varied fields, meaning that there is no set data structure. There are no data types associated with field values, and separate fields may have various data types. Hierarchical data structures are possible using the JSON format, and a field can store several values using an array.Kubernetes Operators are application-specific controllers that allow you to design, configure, and manage stateful applications like databases using the Kubernetes API. Kubernetes users can utilise the MongoDB Enterprise Operator for Kubernetes with MongoDB Ops Manager or Cloud Manager to automate and manage MongoDB clusters on self-managed infrastructure, whether on-premises or in the cloud.MongoDB gives you the freedom to run anywhere you choose, with a database that works in every environment, while containerization and Kubernetes make portability a breeze. Managing your Kubernetes deployments has never been easier, thanks to the ability to control application containers with their underlying database instances.

Our institution is passionate about high quality evidence based research and has excelled in various fields (Parakh et al. 2020; Pham et al. 2021; Perumal, Antony, and Muthuramalingam 2021; Sathiyamoorthi et al. 2021; Devarajan et al. 2021; Dhanraj and Rajeshkumar 2021; Uganya, Radhika, and Vijayaraj 2021; Tesfaye Jule et al. 2021; Nandhini, Ezhilarasan, and Rajeshkumar 2020; Kamath et al. 2020). In the existing research they didn't identify the high availability query processing speed. The main aim of our project is to deploy a mongodb container for high availability query processing on a Novel Docker container and kubernetes based micro service architecture to calculate the query processing speed in seconds.Increasing the speed of Query makes the application to run or execute faster at same time consideration of the deployment time of the query is also very important.In order to use containers, there will be a lot of research, experimenting, and learning from businesses in the coming years.

Materials and Methods

The research work was performed in the Cloud Computing Laboratory, Department of Computer Science and Engineering, Saveetha School of Engineering, SIMATS (Saveetha Institute of Medical and Technical Sciences). The proposed work contains two groups. Group 1 is taken as a Novel Docker container and group 2 as kubernetes based micro service architecture. The Microservice Architecture Based Kubernetes and Novel Docker container were executed and evaluated a different number of times with a sample size of 20. The minimum power analysis for G-Power calculation is fixed at 0.8 and the maximum accepted error is fixed at 0.05 . Same set of 20 queries are used to calculate the processing speed of the queries in each architecture to get the accuracy of each architecture.

Testing setup for this proposed system used a Vmware workstation. Vmware workstation is used to create a guest OS. Hardware configuration for this proposed system is Intel core i5 8th gen processor and requires 4GB random access memory and 256GB Solid state drive used. The configuration of the system is the Windows 10 operating system.

Procedure for deploying mongodb container on docker

Step-1: Install Docker

Install the Docker community edition

- $sudo apt install docker-ce

Check the status of the installation with the following command

- sudo systemctl status docker

Step-2: Install Docker Compose

Install the current stable release of Docker Compose

Sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

Step-3: Setting up a MongoDB container

This section will show you how to use a Docker Compose file to create a MongoDB container. Let's use the search command to find the official MongoDB container image before producing the compose file.

- $sudo search docker mongodb

Next, we need to create a directory called "mongodb" to hold the docker-compose file.

- mkdir -p mongodb/database

In the compose file, we have created a service called mongodb using the Docker image mongo.

Step-4: Interacting with the MongoDB container

Using the docker exec command, we can access the terminal of the MongoDB container.

- $sudo docker exec -it mongodb bash

In the bash terminal of the container, we call the mongo command to access MongoDB. We will create a database called "college" and a collection called "students records", along with three documents.

Step-5: Calculate query processing speed

In the bash terminal after executing the queries use the below queries to calculate the execution time.

- db. setProfilingLevel(2)
- so after the query you can use the db. system. profile. find() to see the query execution time and other.

Algorithm for deploying mongodb container for high availability query processing using Docker is given below:

Input: Query Q to run

```
M=Mongodb container
Q=Query
D=Docker Compose file
E=Docker exec
Enable Docker(M)              //Enable docker service
Create Dockercomposefile(*D)        //Function to create dockercomposefile
        Create Dockercomposefile(*D)=new Dockercomposefile(*D)
Configure(*CF)
        Configuration of(Dockerfile)
Run Dockerexec(*E)          //Funcion to implement Docker exec
        Run Dockerexec(*E)=new Dockerexec(*E)
Create Query(*Q)
        Create Query(*Q)=new Query(*Q)
if( Enable Docker(M)  == true)
Create Dockercomposefile(D)
Configuration of(Dockercomposefile)
Run Dockerexec(M)
Create Query(Q)
Run Query(Q)
 End if
 Else
      "Error while enabling docker"
 End Else
```

Output:

Mongodb container Deployment
Query processing speed in sec
Availability

Procedure for deploying mongodb container on Kubernetes based micro service architecture

Step-1: Set-up KIND

- **Docker:** In order for "KIND" to work, Docker must be installed. To install docker on Linux, use the package manager that comes with the operating system, such as apt on Ubuntu.
- **Kubectl:**It will need to utilise the kubectl command to interact with the cluster once it is up and running. It can discover kubectl install instructions, including methods to install it, using the Linux distribution's packages manager.

- **Kind:** Finally KIND can be installed using the package manager of the linux operating system.

Step-2: Creation of cluster

It's time to start building the local Kubernetes cluster after installing all of these components. Kind deploys a Kubernetes instance using a Novel Docker container. Because any other containers operating on the system may clash with the ports, it is advisable to halt them.

Step-3: MongoDB Deployment

A deployment specifies a desired condition that the Deployment Controller will enforce. When a deployment's status deviates from the expected state, the Deployment Controller takes steps to restore the desired state.
Create a new file named "mongodb-deployment.yaml" and add the following contents
The example above will deploy a single replica of a MongoDB instance.

- $kubectl apply -f mongodb-deployment.yaml

Step-4: Create Persistent Storage

Containers are made to be ephemeral. When the container is stopped, any changes to its state are lost. If you're using MongoDB as a database server, this implies your entire database will be erased. Persistent Volumes can be mounted to a Pod, allowing data to live on after the container has been decommissioned. Create a Persistent Volume Claim and then mount the volume to a Deployment to add persistent volumes to a container in Kubernetes.
Create a file named mongodb-pvc.yaml
The volume claim will create a writable storage volume, provisioned by your cloud provider, with a 1GB size.
Apply it to the cluster using the kubectl apply command

- $kubectl apply -f mongodb-pvc.yaml

Step-5: Exposing Service

So far, we've only installed a single MongoDB Pod. Outside of testing and development, it is strongly discouraged to expose this single pod as a service. Pods are ephemeral, and their state, including the allocated IP address, is lost when they cease.
Algorithm for deploying mongodb container on Kubernetes based micro service architecture is given below:

Input: Query Q to execute

```
    Q=query
    C=Cluster(*C)
    S=Service(*S)
    M=mongodb container
    E=expose()
    S=Storage
    P=Port
Kindsetup(M)   //Setup kind Service for query processing
          Enable Docker()
    Create cluster(*C)       //Function to create cluster
          Create cluster(*C)=new cluster(*C)
    Create service(*S)      //Funcion to create Service
          Create service(*S)=new service(*S)
    Create Query(*Q)
          Create Query(*Q)=new Query(*Q)
   Create storage(*S)
          Create Storage(*S)=new Storage(*S)
    configure(*CF)
          Configuration of(Cluster,Service,Port)
configure(*CF)
          Configuration(Q,S)
    expose(*S)
          Expose service(*S)=Service deployment.Kubectl()
    if( kindsetup(A) == true)
    Create cluster(C)     //Create Cluster
    Configure(P)          //Configure ports
    Create Service(S)    //Create Service
    Create Query(Q)     //Create Query
    Create Storage(S)   //Create storage
    Expose(Service (S)       //Expose Service
    G=Availability(Q)
```

return(G)
End if
Else
"Error while setting up Kind"
End Else

Output:

Mongodb deployment
Query processing speed
Availability

Statistical Analysis

Statistical software used in the study is IBM SPSS version 26. The independent sample T-test calculation for analysing equal variance, standard error, and levene's test are evaluated. Attributes like platform, Deployment Time, accuracy, availability are dependent variables. Independent sample T-test has been carried out for evaluating the accuracy (*VM2Docker: Automating the Conversion from Virtual Machine to Docker Container* 2015).

Results

In this proposed system it was observed that the Novel Docker container appears to be better than the Kubernetes based micro service Architecture. Novel Docker container enables the continuous delivery and processing of quey's. Table 1 represents query processing time and accuracy. Table 2 shows the statistical calculation such as mean, standard deviation and standard error mean for Novel Docker container and Kubernetes based micro service Architecture respectively. The mean, standard deviation and standard error mean for the Novel Docker container are 64.40,5.814,2.600 respectively. The mean, standard deviation and standard error mean for Kubernetes based micro service Architecture are 58.00,5.148,2.302 respectively.

It is inferred that the mean accuracy for T-test is 64 which is greater than the mean accuracy of comparison architecture which is 58.00. Moreover, the mean accuracy value of the Novel Docker container is around 64.00 which seems to be superior to the Kubernetes based micro service Architecture. In Table 3, it was observed that the Levens test for equality of variance and its significance for the Novel Docker container is 0.204 and 0.039 respectively and standard error difference and confidence interval are lower than Kubernetes based micro service architecture. Fig. 1 represents the architecture of the proposed system. Mean accuracy and mean loss graph is depicted in Fig. 2. Novel Docker container seems to appear better for Processing the large number of queries.

Discussion

The proposed Novel Docker Container provides better processing speed in less time compared to Kubernetes based micro service Architecture.Docker established the container industry standard, allowing containers to be transported anywhere. Containers share the machine's OS system kernel, thus they don't need their own OS, resulting in improved server efficiency and lower server and licensing costs(Arundel and Domingus 2019). Docker made it possible to have the same development experience across all operating systems, however it can cause performance concerns on Windows in some cases(Singleton 2017). WSL2 improves performance, but it necessitates some thought as to where the file should be saved and how volumes should be created.

Access to OS-level facilities, such as kernel modules, in-memory state, and physical devices, is required for infrastructure services. Containerizing these types of services without sacrificing mobility and security is difficult due to these criteria.To run a MongoDB instance in a Novel Docker container, we utilize a Docker image for MongoDB. We set up a database and added collections to it. A MongoDB database was also backed up, and the database was then restored from the backup(Membrey, Plugge, and Hawkins 2011).Containers are still a relatively new technology for businesses. Expertise in developing cloud-native and microservices apps is even more recent. While cloud companies can provide managed services for most of the technology, skills, procedures, and people are far more difficult to modify.

Containers also provide the technological foundation for serverless and functions as a service, as well as a greater level of abstraction and a different computation model that can benefit specific types of applications(Membrey, Plugge, and Hawkins 2011; Ibryam and Huß 2019). Containers are ushering in a sea change that cannot be overstated. Enterprises will be able to act more like cloud hyperscalers, building and iterating apps quickly and operating them at a big scale, thanks to containers and the public cloud(Adkins et al. 2020).

Conclusion

The Novel Docker container processes the queries with high processing speed and less time compared to Kubernetes based micro service Architecture. Therefore the Novel Docker container seems to be better for query processing.

Declarations
Conflict of Interests
No conflict of interest in this manuscript.

Author Contribution
Author SN was involved in data collection, data analysis, manuscript writing. Author SMS was involved in conceptualization, guidance and critical review of manuscript.

Acknowledgments
The authors would like to express their gratitude towards Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences (Formerly known as Saveetha University) for providing the necessary infrastructure to carry out this work successfully.

Funding
We thank the following organizations for providing financial support that enabled us to complete the study.
1. Renaissance Technologies, Chennai.
2. Saveetha University
2. Saveetha Institute of Medical and Technical Sciences.
3. Saveetha School of Engineering.

REFERENCES

1. Adkins, Heather, Betsy Beyer, Paul Blankinship, Piotr Lewandowski, Ana Oprea, and Adam Stubblefield. 2020. Building Secure and Reliable Systems: Best Practices for Designing, Implementing, and Maintaining Systems. O'Reilly Media.
2. Arundel, John, and Justin Domingus. 2019. Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud. "O'Reilly Media, Inc."
3. Baier, Jonathan, and Jesse White. 2018a. Getting Started with Kubernetes: Extend Your Containerization Strategy by Orchestrating and Managing Large-Scale Container Deployments, 3rd Edition. Packt Publishing Ltd.
4. ———. 2018b. Getting Started with Kubernetes: Extend Your Containerization Strategy by Orchestrating and Managing Large-Scale Container Deployments, 3rd Edition. Packt Publishing Ltd.
5. De Santis, Sandro, Luis Florez, Duy V. Nguyen, Eduardo Rosa, and I. B. M. Redbooks. 2016. Evolve the Monolith to Microservices with Java and Node. IBM Redbooks.
6. Devarajan, Yuvarajan, Beemkumar Nagappan, Gautam Choubey, Suresh Vellaiyan, and Kulmani Mehar. 2021. "Renewable Pathway and Twin Fueling Approach on Ignition Analysis of a Dual-Fuelled Compression Ignition Engine." Energy & Fuels: An American Chemical Society Journal 35 (12): 9930–36.
7. Dhanraj, Ganapathy, and Shanmugam Rajeshkumar. 2021. "Anticariogenic Effect of Selenium Nanoparticles Synthesized Using Brassica Oleracea." Journal of Nanomaterials 2021 (July). https://doi.org/10.1155/2021/8115585.
8. Guthrie, Scott, Mark Simms, Tom Dykstra, Rick Anderson, and Mike Wasson. 2014. Building Cloud Apps with Microsoft Azure: Best Practices for DevOps, Data Storage, High Availability, and More. Microsoft Press.
9. Ibryam, Bilgin, and Roland Huß. 2019. Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications. O'Reilly Media.
10. Jangla, Kinnary. 2018. Accelerating Development Velocity Using Docker: Docker Across Microservices. Apress.
11. Kamath, S. Manjunath, K. Sridhar, D. Jaison, V. Gopinath, B. K. Mohamed Ibrahim, Nilkantha Gupta, A. Sundaram, P. Sivaperumal, S. Padmapriya, and S. Shantanu Patil. 2020. "Fabrication of Tri-Layered Electrospun Polycaprolactone Mats with Improved Sustained Drug Release Profile." Scientific Reports 10 (1): 18179.
12. Membrey, Peter, Eelco Plugge, and Duptim Hawkins. 2011. The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing. Apress.
13. Mouat, Adrian. 2015. Using Docker: Developing and Deploying Software with Containers. "O'Reilly Media, Inc."
14. Nandhini, Joseph T., Devaraj Ezhilarasan, and Shanmugam Rajeshkumar. 2020. "An Ecofriendly Synthesized Gold Nanoparticles Induces Cytotoxicity via Apoptosis in HepG2 Cells." Environmental Toxicology, August. https://doi.org/10.1002/tox.23007.
15. National Academies of Sciences, Engineering, and Medicine, Division on Engineering and Physical Sciences, Computer Science and Telecommunications Board, Panel on Artificial Intelligence, and Committee on Depicting Innovation in Information Technology. 2020. Information Technology Innovation: Resurgence, Confluence, and Continuing Impact. National Academies Press.
16. Parakh, Mayank K., Shriraam Ulaganambi, Nisha Ashifa, Reshma Premkumar, and Amit L. Jain. 2020. "Oral Potentially Malignant Disorders: Clinical Diagnosis and Current Screening Aids: A Narrative Review." European Journal of Cancer Prevention: The Official Journal of the European Cancer Prevention Organisation 29 (1): 65–72.
17. Perumal, Karthikeyan, Joseph Antony, and Subagunasekar Muthuramalingam. 2021. "Heavy Metal Pollutants and Their Spatial Distribution in Surface Sediments from Thondi Coast, Palk Bay, South India." Environmental Sciences Europe 33 (1). https://doi.org/10.1186/s12302-021-00501-2.
18. Pham, Quoc Hoa, Supat Chupradit, Gunawan Widjaja, Muataz S. Alhassan, Rustem Magizov, Yasser Fakri Mustafa, Aravindhan Surendar, Amirzhan Kassenov, Zeinab Arzehgar, and Wanich Suksatan. 2021. "The Effects of Ni or Nb Additions on the Relaxation

Behavior of Zr55Cu35Al10 Metallic Glass." Materials Today Communications 29 (December): 102909.
19. Sahai, Akhil, and Sven Graupner. 2007. Web Services in the Enterprise: Concepts, Standards, Solutions, and Management. Springer Science & Business Media.
20. Sathiyamoorthi, Ramalingam, Gomathinayakam Sankaranarayanan, Dinesh Babu Munuswamy, and Yuvarajan Devarajan. 2021. "Experimental Study of Spray Analysis for Palmarosa Biodiesel-diesel Blends in a Constant Volume Chamber." Environmental Progress & Sustainable Energy 40 (6). https://doi.org/10.1002/ep.13696.
21. Singleton, James. 2017. ASP.NET Core 2 High Performance: Learn the Secrets of Developing High Performance Web Applications Using C# and ASP.NET Core 2 on Windows, Mac, and Linux. Packt Publishing Ltd.
22. Surianarayanan, Chellammal, Gopinath Ganapathy, and Raj Pethuru. 2019. Essentials of Microservices Architecture: Paradigms, Applications, and Techniques. Taylor & Francis.
23. Tesfaye Jule, Leta, Krishnaraj Ramaswamy, Nagaraj Nagaprasad, Vigneshwaran Shanmugam, and Venkataraman Vignesh. 2021. "Design and Analysis of Serial Drilled Hole in Composite Material." Materials Today: Proceedings 45 (January): 5759–63.
24. Uganya, G., Radhika, and N. Vijayaraj. 2021. "A Survey on Internet of Things: Applications, Recent Issues, Attacks, and Security Mechanisms." Journal of Circuits Systems and Computers 30 (05): 2130006.
25. VM2Docker: Automating the Conversion from Virtual Machine to Docker Container. 2015.
26. Zhelev, Svetoslav, and Anna Rozeva. 2019. "Using Microservices and Event Driven Architecture for Big Data Stream Processing." PROCEEDINGS OF THE 45TH INTERNATIONAL CONFERENCE ON APPLICATION OF MATHEMATICS IN ENGINEERING AND ECONOMICS (AMEE'19). https://doi.org/10.1063/1.5133587.

Table 1. Query processing time and accuracy for Novel Docker container and Kubernetes Based micro service architecture.

ITERATION NO (n)	Novel Docker container		Kubernetes based micro service architecture	
	Time (in Sec)	Accuracy(%)	Time (in Sec)	Accuracy(%)
1	6	73	8	66
2	3	63	6	58
3	7	58	9	53
4	11	61	14	54
5	5	67	9	59

Table 2. Group statistical analysis of Novel Docker container with mean value of 64.40 and KUBERNETES BASED MICRO SERVICE ARCHITECTURE with mean value of 58.00 and similarly the results of Standard Deviation and Standard Error Mean are given.

T-Test:

Group Statistics

GROUP		N	Mean	STD Deviation	STD Error mean
ACCURACY	Novel Docker container	5	64.40	5.814	2.600

	Kubernetes based micro service architecture	5	58.00	5.148	2.302

Table 3. Independent Sample T-test Results with confidence interval of 95% and level of significance greater than 0.05 (Novel Docker container seems to appear better for the processing of queries in less time).

		Levene's Test for Equality of Variance		**Levene's Test for Equality of Variance**						
	Equal Variances	**F**	**Sig.**	**t**	**df**	**Sig.(2-tailed)**	**Mean Difference**	**Std. Error Differ -ence**	**95% Confidence Interval of the Difference**	
									Lower	**Upper**
ACCURACY	**Assumed**	.204	.039	1.843	8	.103	6.400	3.473	-1.608	14.408
	Not Assumed			1.843	7.884	.103	6.400	3.473	-1.629	14.429

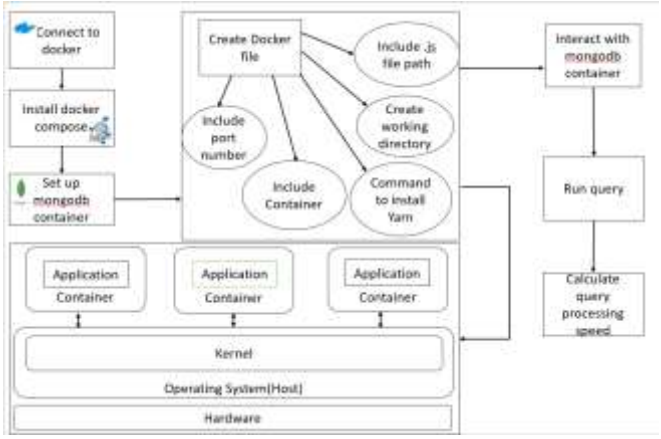

Fig. 1. Architecture for Processing a query in a Novel Docker container using different platforms. Docker,Docker compose, Docker file are the important components in the architecture.

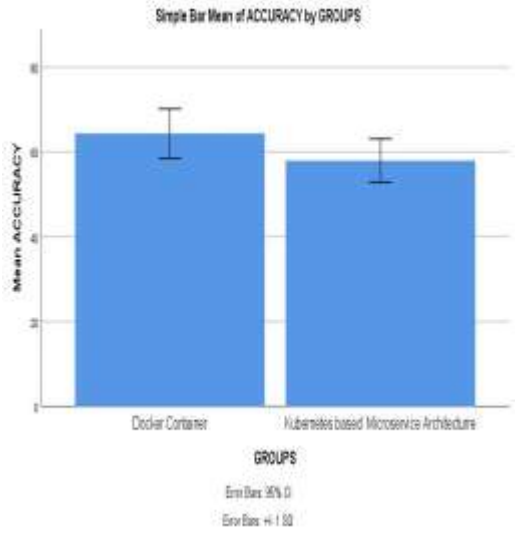

Fig. 2. Bar graph analysis of Novel Docker container and kubernetes based micro service architecture. Graphical representation shows the mean Accuracy of 64.40% and 58.00% for the proposed platform (Novel Docker container) and Kubernetes respectively. X-axis : DC vs KB, Y-axis : Mean Accuracy ± 1 SD.