Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Load balancing and service discovery using Docker Swarm for microservice based big data applications

Neelam Singh[1], Yasir Hamid[2], Sapna Juneja[3], Gautam Srivastava[4,5,6], Gaurav Dhiman[1,7,8,9], Thippa Reddy Gadekallu[9,10] and Mohd Asif Shah[11,12*]

Abstract

Big Data applications require extensive resources and environments to store, process and analyze this colossal collection of data in a distributed manner. Containerization with cloud computing provides a pertinent remedy to accommodate big data requirements, however requires a precise and appropriate load-balancing mechanism. The load on servers increases exponentially with increased resource usage thus making load balancing an essential requirement. Moreover, the adjustment of containers accurately and rapidly according to load as per services is one of the crucial aspects in big data applications. This study provides a review relating to containerized environments like Docker for big data applications with load balancing. A novel scheduling mechanism of containers for big data applications established on Docker Swarm and Microservice architecture is proposed. The concept of Docker Swarm is utilized to effectively handle big data applications' workload and service discovery. Results shows that increasing workloads with respect to big data applications can be effectively managed by utilizing microservices in containerized environments and load balancing is efficiently achieved using Docker Swarm. The implementation is done using a case study deployed on a single server and then scaled to four instances. Applications developed using containerized microservices reduces average deployment time and continuous integration.

Keywords Big data, Containerization, Docker, Microservice, Docker Swarm, Load-balancing

*Correspondence:
Mohd Asif Shah
ohaasif@kdu.edu.et
[1] Department of Computer Science and Engineering, Graphic Era Deemed to be University, Dehradun 248002, India
[2] Abu Dhabi Polytechnic, Abu Dhabi Polytechnic, Abu Dhabi, United Arab Emirates
[3] KIET Group of Institutions, Delhi NCR, Ghaziabad, India
[4] Department of Mathematics and Computer Science, Brandon University, Brandon, Canada
[5] Research Centre of Interneural Computing, China Medical University, 40402 Taichung, Taiwan
[6] Dept. of Computer Science and Math, Lebanese American University, 1102 Beirut, Lebanon
[7] Govt. Bikram College of Commerce, Patiala, Punjab, India
[8] University Centre for Research and Development, Department of Computer Science and Engineering, Chandigarh University, Gharuan, Mohali 140413, India
[9] Department of Electrical and Computer Engineering, Lebanese American University, Byblos, Lebanon
[10] School of Information Technology and Engineering, Vellore Institute of Technology, Vellore, India
[11] Department of Economics, Kebri Dehar University, Kebri Dehar, Somali 250, Ethiopia
[12] School of Business, Woxsen University, Hyderabad, Telangana, 502345, India

Introduction

The Big Data era led to the advent of tools, technologies and architectures with improved efficiency, elasticity and resiliency. Big data applications need sophisticated architectures with inherent capabilities to scale and optimize. To enhance the scalability and elasticity of big data application deployment, implemented environments need to be continuously improved and updated. Organizations are using cloud based services to enhance the performance and to lessen overall cost. Containerization, a cloud based technology, is attaining popularity since it is lightweight in nature. Docker is one of the predominant, extensively used container based virtualizations, and since it is an open source project, it can be used to develop, run and deploy an application efficiently.

Big data applications require extensive resources and environments to store, process and analyze this colossal collection of data in a distributed manner. Containerization with cloud computing provides a pertinent remedy to accommodate big data requirements, however requires a precise and appropriate load-balancing mechanism. The load on servers increases exponentially with increased resource usage, thus making load balancing an essential requirement. Moreover, the adjustment of containers accurately and rapidly according to load as per services is one of the crucial aspects in big data applications. This study provides a review relating to containerized environments like Docker for big data applications with load balancing. In this study, a scheduling mechanism of containers for big data applications established on Docker Swarm and Microservice architecture is proposed. In this paper, we utilize the concept of Docker Swarm to effectively handle big data applications workload and service discovery.

Improving the performance of an application is an ongoing struggle with increased demands and usage. Technical innovations are always aiming towards achieving higher degree of efficiency and performance. Employing an environment that gives performance, reliability and fault tolerance is required by all organizations. Cloud Computing has created a niche for itself when performance, resiliency, availability, and a cost effective solution is required. Modern technologies are employing Cloud Computing to gain benefits as the Cloud enables the ubiquitous availability of resources in a cost effective yet competent manner [1–6]. The Cloud has made resources like infrastructure, platform, and software available to the end user without any management efforts. The Cloud offers everything as a service including emerging technologies like the Internet of Things (IoT) or Big Data [3]. Various organizations are offering cloud based solutions for handling Big Data as is shown in [7]. Elasticity in a cloud environment having multi-tier structure is achieved by scaling the quantity of physical resources [8, 9]. Resources can be scaled in two ways, either by horizontal scaling i.e. by adding more (VMs) virtual machines [10], or through vertical scaling by adding more resources to the deployed VMs [11]. Both of the methods require additional time, suffer from latency issues, and may incur additional cost. To expedite the processes and optimizing the cost of application development, different paradigms and architectures have been studied and evaluated.

Containerization is one of the cloud based techniques which is gaining popularity because of features like being lightweight, scalability, and availability when compared to virtual machines. Containers are best suited for continuous integration and continuous delivery (CI/CD) workflows. Docker [12], an open source project, is a widely used container-based virtualization tool assisting in the development, execution, and deployment of applications in containerized environments. Docker can manage workloads dynamically in real time due to portability and its lightweight nature. Applications executed in a Docker container remain isolated from the underlying host environment.

Docker can prove beneficial to deploy big data applications. Applications can be deployed in containers to serve massive workloads. It is a challenging task to manage numerous containers for a single application. Docker thankfully comes with a cluster management tool called Docker Swarm to handle multiple clusters. Docker Swarm provides clustering and an orchestration mechanism and thus can deploy several containers across different host machines. Docker Swarm also provides a fault tolerance mechanism not only by detecting failed containers on a host machine, but also redeploying the same container on another host machine [13]. Big data applications suffer from some of the major issues like conventional data analysis methods not adjusting to input data i.e. on-the-fly or real time streaming data. Methods used may also pose computational and speed-up overhead. Moreover, there is no theoretical derivation of parallelization speed-up factors. Machine Learning (ML) programs may exhibit skewed distribution if the load on each machine is not balanced, as they still lack methods to synchronize during waiting times for the exchange of parameter updates. As such, synchronization is another open challenge to handle big data using ML algorithms. Singh et al. [14] proposed a container based microservice architecture to handle monolithic design challenges like scalability, integration, and throughput. However, the focus was not on handling big data based applications which requires massive effort and deployment issues. Another issue which needs to be addressed is how to assign containers in real time accurately to manage service loads. In [15], a container scheduling approach employing ML is proposed by the authors by analyzing data sets using Random Forest (RF).

Most of the current research fails to contribute the cause and effect of decrease in service execution performance due to an increase in load on the nodes. Another area of concern is how to assign service load dynamically at run time in terms of big data applications.

This research aims to distribute big data applications implemented as a microservice inside the Docker Swarm according to resource utilization for respective host machines and service discovery, both of which are important for microservice architectures. The main focus of this research is on memory utilization according to given memory limits. In this paper, we propose a Docker Swarm based mechanism to observe consumption of memory by each host machine and then look to assign the load to a given host machine based on memory usage using microservices for big data applications. Performance of the work is evaluated based on the load assignment according to memory utilization. Contributions of the work focuses on improving performance during higher workloads that may occur due to big data processing and scaling the services to improve the efficiency.

The paper is structured as follows: Section Related Work will give a comprehensive summary of the work done in this area. Section Architectural design of Docker based load balancing and service discovery scheme for microservice based big data application will give an account of the proposed work and methodology and Section Result and Discussion will analyze the result of the proposed work. In Section Conclusion, we conclude our work.

Related work

Big data applications require extensive set of resources like storage, processing power, and communication channels due to their inherent characteristics. To handle this gigantic pile of data, it is common that techniques, frameworks, environments, and methodologies are continuously reviewed, analyzed and developed. This section explores the work done for big data analytics using Cloud computing and the use of Docker as well as Docker Swarm for the purpose of managing and orchestrating clusters for load balancing.

A Microservice based architecture for big data knowledge discovery which aims to acknowledge scalability and efficiency issues in processing is proposed by Singh et al. [16]. Naik et al. [17], have demonstrated the in workings of a model based on big data processing centered on Docker containers in multiple clouds by automatic assignment of big data clusters using Hadoop and Pachyderm. In the development phase, the environment used ensures the accurate working of code but it may fail during the testing or production phase due to environmental changes and/or differences. Containerization comes into play to handle this issue. Hardikar et al. [18] explored several facets of Containerization like automation, deployment, scaling, and load balancing with a focus on Docker as the runtime environment and Kubernetes is deployed for orchestration. The focus is mainly on containerization, but handling the big data microservice is not focused on directly in the study. In microservices, based on neighbourhood divison, a container scheduling approach called CSBND was proposed in [19] to optimize the system performance using response time and load balancing. The research did not handle big data and microservice based applications to be deployed on containers.

Big data analytics

Big Data Analytics deals with discovering knowledge from large datasets popularly known as big data for strategic planning, decision making, and prediction purposes [20]. To analyze these colossal datasets, dynamic environments are required which need to be scalable enough to manage varying workloads as conventional methods often fail to process these large sizes of data. Big Data Analytics is an assortment of tools, technologies, methodologies combined in a system/platform or framework to perform knowledge discovery through processes like data gathering, cleaning, modelling, and visualization [21]. Techniques like machine learning and deep neural networks are utilized to perform the analysis process.

The authors in [20, 22] provide an insight into various machine learning and deep learning algorithms which prove to be beneficial in Big Data Analytics. These processes require sophisticated architecture for storage, processing, and visualization. Cloud computing is considered to be an effective solution for it. The authors illustrated the affinity of Big Data with cloud with respect to its characteristics [23]. A web server load balancing mechanism focused on memory exploitation using Docker Swarm was proposed by Bella et al. [24]. This work focused on web server load balancing, however the service discovery part is not considered in the paper. Big data applications requires extensive use of resources and resource utilization for big data is also not discussed.

Containerization using Docker

To increase the efficiency of methods and optimize development as well as the deployment cost of applications over the cloud, there have been numerous architectures, frameworks, environments, and paradigms examined in the literature extensively. Docker, which is an open source containerization tool, is fast emerging as an alternative for application deployment over any cloud based architecture. Container centric virtualization is a substitute for virtualization done using hypervisor where containers share all resources like hardware, operating system and supporting libraries while maintaining abstraction and isolation [25]. Docker is a well-known lightweight tool providing prompt development and relocation with improved efficiency and flexibility in resource provision [26].

A distinct host can be used to create numerous containers in multiple user spaces, which is unlike VMs [27]. Container-based applications fabricated using Microservice architecture require traffic management and load balancing at high workloads. This issue is handled through container load balancing. A load balancer for a container results in higher availability and scalability of applications for client requests. This ensures seamless performance of Microservice applications running in containers. Tools like Docker Swarm as well as Kuberbnetes provide support to manage and deploy containers. Figure 1 gives an illustration of a distributing application client load to containerized microservices using a load balancer.

Docker Swarm

Management of containers is an important and crucial aspect of containerization. Load Balancing is required to handle requests dynamically. To manage Docker clusters, Docker Swarm, a cluster administration and orchestration tool is used that links and controls all Docker nodes [28]. Docker Swarm offers features like reliability, security, availability, scalability, and maintainability. It helps in the balanced distribution of any load and checks host machines for failed containers. If any failed containers are found, Docker Swarm redeploys it [23]. It is an enhancement of Docker.

Docker Swarm is made up of two types of nodes, manager and worker nodes. All membership and allocation processes are handled by the manager node while worker nodes execute swarm based services in Docker Swarm. The Manager node uses its own IP address and port to expose swarm services to all clients. Requests from clients are channelled to a chosen worker node by the swarm manager's internal load balancing mechanism so that requests are evenly distributed [29]. Although the Docker Swarm load balancing process distributes the load, the ability to monitor resource utilization according to available limits is not provided. This can lead to uneven load distribution making any Big Data Microservice prone to collapse. In this study, we will distribute Microservice based loads in Docker Swarm by checking resource consumption of host machines creating an even load distribution mandated by available limits.

Microservice architecture

Monolithic architectures are the most common conventional architectures used to deploy applications. These architectures work on more or less three basic layers i.e. presentation, business, and data logic in order to handle simple to complex tasks. The architecture is simple and easy to use since everything is under one autonomous deployment unit. However, the architecture may limit the application to scale and make updates a difficult task when a complex task needs to be managed. Microservice architectures aim to minimize the issues that exist in monolithic architectures by dividing the entire application into lightweight and loosely coupled components [30, 31]. Every component has its individual code repository and can be updated independently, making any complex application far more scalable, resilient and efficient. Service Discovery and Load Balancing are two critical as well as fundamental aspects of Microservices. Service Discovery can be defined as a registry of running instances for one or many services. It is required by Microservices to collaborate. Systems' scalability, throughput, execution time, response time, and performance is largely influenced by load balancing [32].

Container-based virtualization and Microservices make a perfect association as containers provide a decentralized environment and are lightweight in nature.

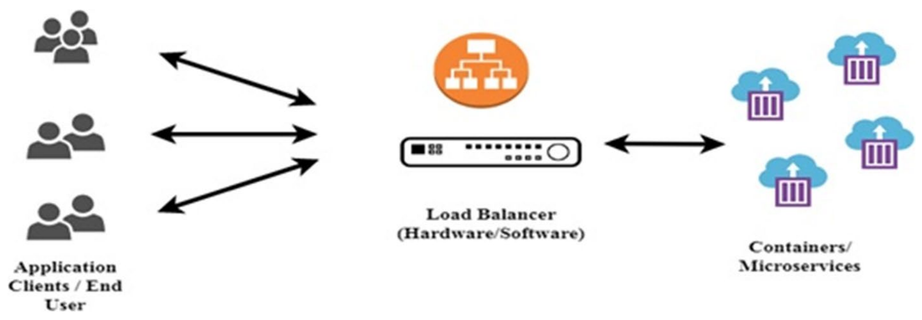

Load Balancer (Hardware/Software)
Containers/ Microservices
Application Clients / End User

Fig. 1 Container load balancing

Today, Docker is used to build modules called Microservices [33], to decentralize packages and distribute jobs into distinct, stand alone applications that collaborate with each other. Microservices can be considered as small applications that must be deployed in their individual VM instances to have discrete environments. But to dedicate an entire VM [34] instance to just a part of an application is not an efficient approach. Docker containers require less computing resources when compared to virtual machines, therefore deploying hundreds and thousands of Microservices on Docker containers will reduce performance overhead and will increase the overall efficiency of the applications [35]. In this study, we will distribute the load of Big Data applications inside a Docker Swarm by utilizing resources of host machines. The main objective is to balance the load by checking memory consumption of all host machines based on known memory limits. This research aims at service discovery and server-side load balancing for Big Data applications based on Microservices using Docker Swarm.

Architectural design of Docker based load balancing and service discovery scheme for microservice based big data application

A fault tolerant and decentralized architecture is provided by Docker Swarm. A set of Docker hosts can be combined into a swarm using swarm mode. Services can be created and scaled with health checks along with built in load balancing and service discovery features. Big Data Applications require an extensive set of resources which are required to be properly load balanced. A Docker based Load Balancing and Service Discovery system is used for Big Data applications. Figure 2 provides a look at the Microservices stack in use for a big data application.

We containerize our application using Docker as Microservices. We used Docker Swarm for orchestration, service discovery, and load balancing.

The Big Data application stack will provide the given functionality in the form of Microservices:

- Extraction of links from the input URL using front end PHP application using Apache server.
- Interaction of the Web application with API server (Python) to manage link extraction and return JSON response.
- An image of Redis cache (used by API server) to check for already scraped pages and evading repeated fetch.

The experimental setup is made up of four Swarm nodes – a master node and three worker nodes as given in Table 1 and Fig. 3, respectively. The master node is implemented using NGINX service where the Swarm commands are run. Swarm itself is responsible for scheduling, Domain Name Service (DNS) service discovery, scaling, and container load balancing on all nodes. This Docker Swarm load balancer will run on every node and will balance load requests as required.

Four services are created within the Docker Swarm. A Master Load Balancer service to enable load balancing and the three other services are Microservices for the implemented scenario for a Big Data application. These three worker nodes are namely the PHP front end Microservice, python API Microservice, and Redis cache Microservice, respectively. The port number

Table 1 Docker Swarm services for big data application

Type	Container	Port
Load Balancer	NGINX	80
Front End PHP service	Apache	8080
API server (Python)	Python	5000
API server (Redis)	Redis	6379

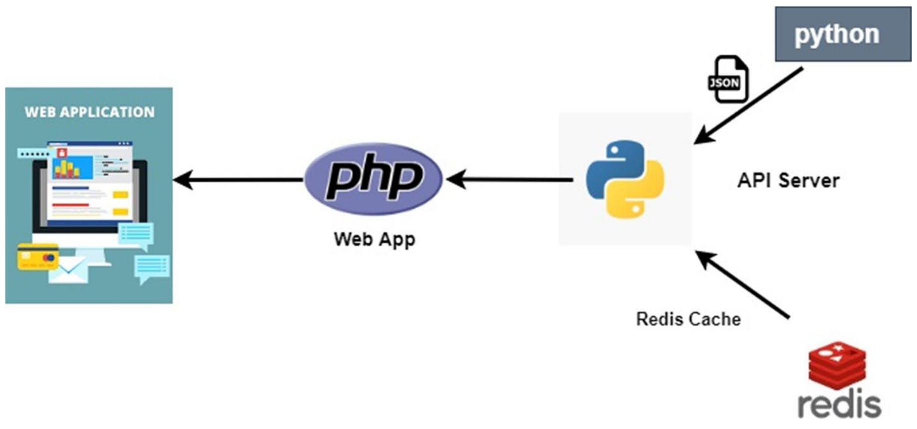
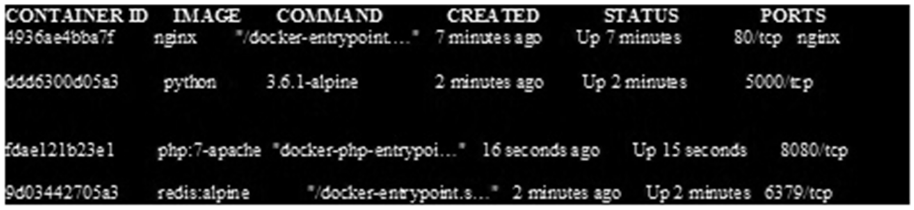


Fig. 2 Microservices stack of a big data application



```
CONTAINER ID   IMAGE        COMMAND                   CREATED          STATUS          PORTS
4936ae4bba7f   nginx        "/docker-entrypoint...."  7 minutes ago    Up 7 minutes    80/tcp   nginx

ddd6300d05a3   python       3.6.1-alpine              2 minutes ago    Up 2 minutes    5000/tcp


fdae121b23e1   php:7-apache "docker-php-entrypoi..."  16 seconds ago   Up 15 seconds   8080/tcp

9d03442705a3   redis:alpine "/docker-entrypoint.s..." 2 minutes ago    Up 2 minutes    6379/tcp
```

Fig. 3 Service Discovery for the Containers

and respective containers of the services are listed in Table 1. For example, to run the load balancer service, it is required to open port number http://192.168.0.23:80 and to access the web server, http://192.168.0.24:8080 is used to acquire the services of the Apache web server. Docker Swarm is responsible to supervise and distribute the containers running these services routinely.

The proposed algorithm for the entire process is discussed in Algorithm 1. Load balancing methodologies are covered in Algorithm 2.

Algorithm 1: algorithmµBigLB (service orchestration)

Step 1: Automate installation of requirements using Docker file and build an isolated image (Container 1).

Step 2: Use Microservice for Big Data Application:

a. Creating full path URLs of extracted path
b. Extracting anchor and link texts
c. Return object, move main logic to function

Step 3:

a. Run Server
b. Map host and container ports
c. Expose link extraction (from step 2) as web service API in second python file (Microservice).

Step 4:

a. Create independent image of all the code
b. Create front end using PHP in different folder
c. Services are integrated using docker-compose.yml

Step 5: Create second container for front end PHP application.

Step 6: Create third container for Redis for caching purpose.

Algorithm 2: load balancing

Step 1: Create NGINX service for load balancing.

Step 2: Run memory monitoring service in each worker node.

Step 3: Check for load and service discovery and redirect load if a worker node fails, to active worker node.

Service discovery is managed by applying Docker Remote API from the services to extract service and instance information. It is a built in service discovery mechanism of the orchestrator. In order to test the load balancing aspect of Docker Swarm, our "linkextractor" Microservice is scaled to run multiple instances using Docker API: docker service scale linkextractor = 4.

This will create 4 replicas of our Microservice and if it is curled a few times, it will get different IP addresses. The calls were done using round-robin over the four instances. This load-balancing mechanism of container orchestrator implemented by Docker Swarm "service" abstraction removes the complexity of client-side based load-balancing. The effect on latency and CPU/memory usage is monitored using Docker API: docker stats, which provides container runtime metrics like CPU usage, Memory usage and limits, and network I/O metrics.

We will consider the following parameters for each scenario (container) implemented:

- CPU usage
- Memory usage and limits
- Network throughput

Result and discussion

Using Remote Docker API, service discovery is first performed as it is one of the crucial elements of any Microservice architecture, so that Microservices can discover and collaborate with each other as shown in Fig. 3. Service discovery helps to allocate and assign nodes with lesser nodes and helps in automatic and continuous integration. Containers are assigned based on workloads once service discovery is performed.

Once all the container images are discovered and loaded, our application is executed to check its working in the containerized environment i.e. to extract links from the given URL. Figure 4 shows the links extracted from a test URL.

Once our application is tested, it is scaled from one to four instances to check the effect on latencies and CPU/memory usage with respect to memory limits.

Results achieved in Table 2 and Fig. 5, respectively, show that all four container instances are comparatively sharing similar workloads. Therefore, based on these results, it is concluded that containerized microservices for big data applications based on the proposed architecture can be effectively managed on Docker Swarm. More instances can be added to scale up and handle the deployment and continuous integration process in a much better way.

Monolithic applications suffer from scalability and integration issues making it challenging to handle big data applications which can be easily managed by the proposed architecture.

The given case study illustrates the requirement of containerization for applications working on Big Data. The following section illustrates both the merits and demerits of the strategy as proposed:

Merits

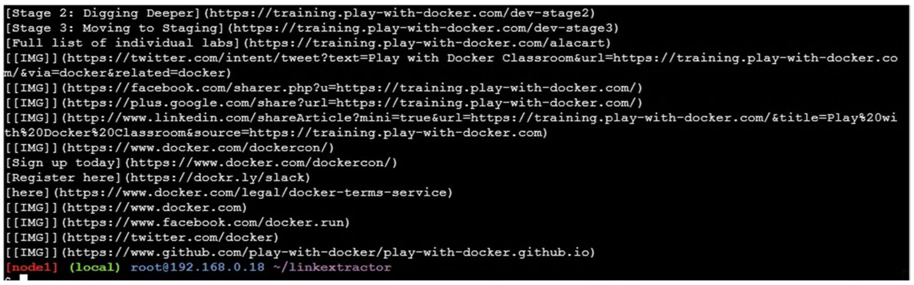
- Containers are well suited for complex applications deployed as microservices and thus can help the efficient balancing of loads across servers as compared to VMs (Virtual Machines)
- According to the requirements of a given application, functionality can be scaled by deploying more containers which can be managed effectively using Docker Swarm. This process is difficult to address using virtualized environments
- Containers can be very easily duplicated or deleted according to requirements and the Swarm can handle this aspect in an efficient manner.

Demerits

- Containers provide scalability, however portability can be affected by placing dependencies on containers.
- Containers are susceptible to attacks as they share the OS kernel. This can affect service discovery and load balancing across servers in case of an attack or any malicious activities.
- Though Containers can be duplicated at an amazing speed, they consume a huge amount of resources making them a costlier strategy as compared to other techniques like virtualization.

Conclusion

It is often a difficult and time consuming process to manage Big Data applications because of their predominant characteristics. Microservices are considered to be a better option to provide a scalable and fault tolerant approach to Big Data application management. Service discovery and load balancing are both important aspects of Microservices that need to be addressed in modern systems. In this study, the benefits of containerization on Microservice based Big Data applications was illustrated. The load balancing and service discovery facets of Microservices are properly handled by a Docker



```
[Stage 2: Digging Deeper](https://training.play-with-docker.com/dev-stage2)
[Stage 3: Moving to Staging](https://training.play-with-docker.com/dev-stage3)
[Full list of individual labs](https://training.play-with-docker.com/alacart)
[[IMG]](https://twitter.com/intent/tweet?text=Play with Docker Classroom&url=https://training.play-with-docker.co
m/&via=docker&related=docker)
[[IMG]](https://facebook.com/sharer.php?u=https://training.play-with-docker.com/)
[[IMG]](https://plus.google.com/share?url=https://training.play-with-docker.com/)
[[IMG]](http://www.linkedin.com/shareArticle?mini=true&url=https://training.play-with-docker.com/&title=Play%20wi
th%20Docker%20Classroom&source=https://training.play-with-docker.com)
[[IMG]](https://www.docker.com/dockercon/)
[Sign up today](https://www.docker.com/dockercon/)
[Register here](https://dockr.ly/slack)
[here](https://www.docker.com/legal/docker-terms-service)
[[IMG]](https://www.docker.com)
[[IMG]](https://www.facebook.com/docker.run)
[[IMG]](https://twitter.com/docker)
[[IMG]](https://www.github.com/play-with-docker/play-with-docker.github.io)
[node1] (local) root@192.168.0.18 ~/linkextractor
$
```

Fig. 4 Links extracted from a given URL

Table 2 Resource utilization by container instances (4)

Container	CPU %	Memory Usage (MB) / Limit (GB)	Net I/O
linkextractor.3.8j1ink3r57nfi1q6a2xf5gkc	10.59%	9.306 / 1.955	1.209 KB/ 578 B
linkextractor.2.ap3adb28c79c8ax3age8ah1qp	11.08%	9.114 / 1.955	1.312 KB/ 630 B
inkextractor.4.2kilgzglenbb0wew9hdme4z7t	11.32%	9.288 / 1.955	1.101 KB/ 528 B
linkextractor.1.y3yojmtxcvva3wa1q9nrh9asb	10.17%	21.26 / 1.955	1.266 KB/ 600 B

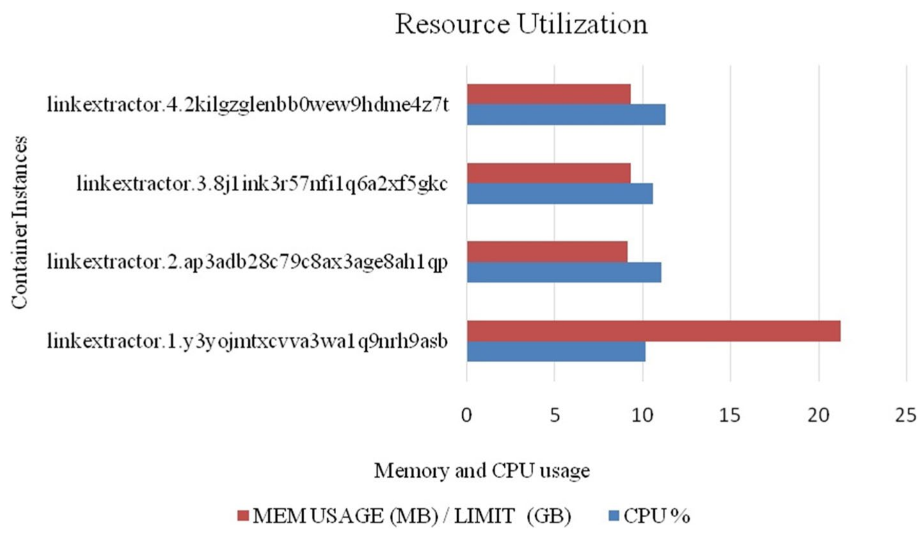


Fig. 5 Memory and CPU usage of four containers in Swarm

container and its attached orchestration tool called Docker Swarm.

This proposed concept shows the usefulness of the Docker tools suite in orchestrating a multi-service stack like is needed for Big Data Applications. This technique can be utilized to avoid a single point of failure in Big Data applications, as such making applications more scalable, resilient, and portable. In the future, the computational complexity and cost efficiency of the proposed work needs to be examined and addressed. The given techniques as presented can also be developed and implemented for Big Data applications in multi-cloud scenarios.

Acknowledgments
Not applicable.

Code availability
Not Applicable.

Authors' contributions
Conceptualization by Neelam Singh; Methodology by Sapna Juneja; Software and formal analysis by Yasir Hamid; Investigation and Writing by Gautam Srivastava; Resources and data collection by Writing by: Gaurav Dhiman; Validation by: Thippa Reddy Gadekallu and Mohd Asif Shah. The authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
Not Applicable.

Declarations

Competing interests
Not Applicable.

Received: 15 September 2022 Accepted: 2 November 2022
Published online: 07 January 2023

References

1. Fox A, Griffith R, Joseph A, Katz R, Konwinski A, Lee G et al (2009) Above the clouds: a berkeley view of cloud computing. Rep UCBIEECS 28
2. Armbrust M et al (2010) A view of cloud computing. Commun ACM 53(4):50–58
3. Rimal BP, Jukan A, Katsaros an Goeleven D (2011) Architectural requirements for cloud computing systems: an Enterprise cloud approach. J Grid Comput 9(1):3–26
4. Buyya R, Yeo CS, Venugopal S (2008) Marketoriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities. In: Proceedings of the 10th IEEE international conference on high performance computing and communications
5. Vouk MA (2008) Cloud computing issues, research and implementations. In: 30th international conference on information technology interfaces (ITI 2008), Cavtat/Dubrovnik, pp 31–40
6. P. Mell and T. Grance, "Draft nist working definition of cloud computing",2009. Available: http://csrc.nist.gov/groups/SNS/cloud-computing/index.html
7. Wan J, Cai H, Zhou K (2015) Industrie 4.0: enabling technologies. In: Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things, pp 135–140. https://doi.org/10.1109/ICAIOT.2015.7111555
8. Liu Z, Zhang Q, Zhani MF, Boutaba R, Liu Y, Gong Z (2015) DREAMS: dynamic resource allocation for MapReduce with data skew. In: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp 18–26. https://doi.org/10.1109/INM.2015.7140272
9. Wei G, Vasilakos AV, Zheng Y, Xiong N (2010) A game-theoretic method of fair resource allocation for cloud computing services. J Supercomput 54(2):252–269
10. Jiang J, Lu J, Zhang G, Long G (2013) Optimal Cloud Resource Auto-Scaling for Web Applications. In: 2013 13th IEEE/ACM international symposium on cluster, Cloud, and Grid Computing, pp 58–65. https://doi.org/10.1109/CCGrid.2013.73
11. Shi X, Dong J, Djouadi S, Feng Y, Ma X, Wang Y (2016) PAPMSC: power-aware performance management approach for virtualized web servers via stochastic control. J Grid Comput 14(1):171–191

12. Preeth EN, Mulerickal FJ, Mulerickal BP, Sastri Y (2015) Evaluation of Docker containers based on hardware utilization. In: 2015 International Conference on Control Communication & Computing India (ICCC), pp 697–700. https://doi.org/10.1109/ICCC.2015.7432984
13. Ismail BI et al (2015) Evaluation of Docker as edge computing platform. In: 2015 IEEE Conference on Open Systems (ICOS), pp 130–135. https://doi.org/10.1109/ICOS.2015.7377291
14. Singh V, Peddoju SK (2017) Container-based microservice architecture for cloud applications. In: 2017 International Conference on Computing, Communication and Automation (ICCCA), pp 847–852. https://doi.org/10.1109/CCAA.2017.8229914
15. Lv J, Wei M, Yu Y (2019) A container scheduling strategy based on machine learning in microservice architecture. In: 2019 IEEE International Conference on Services Computing (SCC), pp 65–71. https://doi.org/10.1109/SCC.2019.00023
16. Singh N, Singh DP, Pant B, Tiwari UK (2021) μBIGMSA-microservice-based model for big Data knowledge discovery: thinking beyond the monoliths. Wirel Pers Commun 116(4):2819–2833
17. Naik N, Jenkins P, Savage N, Katos V (2016) Big data security analysis approach using computational intelligence techniques in R for desktop users. IEEE Symposium Series on Computational Intelligence (SSCI) 2016:1–8. https://doi.org/10.1109/SSCI.2016.7849907
18. Hardikar S, Ahirwar P, Rajan S Containerization: cloud computing based inspiration Technology for Adoption through Docker and Kubernetes. In: 2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC), vol 2021, pp 1996–2003. https://doi.org/10.1109/ICESC51422.2021.9532917
19. Guo Y, Yao W (2018) A container scheduling strategy based on neighborhood division in micro service. In: NOMS 2018–2018 IEEE/IFIP Network Operations and Management Symposium, pp 1–6. https://doi.org/10.1109/NOMS.2018.8406285
20. Singh N, Singh DP, Pant B (2017) A comprehensive study of big data machine learning approaches and challenges. In: 2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS), pp 80–85. https://doi.org/10.1109/ICNGCIS.2017.14
21. Trnka A (2014) Big data analysis. Eur J Sci Theol 10(1):143–148
22. Najafabadi MM, Villanustre F, Khoshgoftaar TM, Seliya N, Wald R, Muharemagic E (2015) Deep learning applications and challenges in big data analytics. J Big Data 2(1):1–21
23. Hashem IAT, Yaqoob I, Anuar NB, Mokhtar S, Gani A, Khan SU (2015) The rise of 'big data' on cloud computing: review and open research issues. Inf Syst 47:98–115
24. Bella MRM, Data M, Yahya W (2018) Web server load balancing based on memory utilization using Docker swarm. In: 2018 International Conference on Sustainable Information Engineering and Technology (SIET), pp 220–223. https://doi.org/10.1109/SIET.2018.8693212
25. Soltesz S, Pötzl H, Fiuczynski ME, Bavier A, Peterson L (2007) Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. SIGOPS Oper Syst Rev 41(3):275–287 (Pubitemid 47281589)
26. Felter W, Ferreira A, Rajamony R, Rubio J (2015) An updated performance comparison of virtual machines and Linux containers. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp 171–172. https://doi.org/10.1109/ISPASS.2015.7095802
27. J. Turnbull, The Docker Book, 2014, Available: www.dockerbook.com
28. Docker.com./Docker Swarm. https://docs.docker.com/engine/swarm/. Accessed 24 Aug 2020]
29. Docker Swarm mode key concepts. Available: https://docs.docker.com/engine/swarm/key-concepts/. Accessed 24 Aug 2020
30. Al-Masri E (2018) Enhancing the microservices architecture for the internet of things. In: 2018 IEEE International Conference on Big Data (Big Data), pp 5119–5125. https://doi.org/10.1109/BigData.2018.8622557
31. Imran S (2021) Ahmad, and do Hyeun Kim, "a task orchestration approach for Efficient Mountain fire detection based on microservice and predictive analysis in IoT environment". J Intell Fuzzy Syst 40(3):5681–5696
32. Dhiman G et al (2022) Federated learning approach to protect healthcare data over big data scenario. Sustainability 14(5):2500
33. Singh P et al (2022) A fog-cluster based load-balancing technique. Sustainability 14(13):7961
34. Kanwal S et al (2022) Mitigating the coexistence technique in wireless body area networks by using superframe interleaving. IETE J Res 2022:1–15
35. Kour K et al (2022) Smart-hydroponic-based framework for saffron cultivation: a precision smart agriculture perspective. Sustainability 14(3):1120

Publisher's Note