International Journal of Current Engineering and Technology
E-ISSN 2277 – 4106, P-ISSN 2347 – 5161

Available at http://inpressco.com/category/ijcet

Research Article

Utilizing Docker Containers for Reproducible Builds and Scalable Web Application Deployments

Vasudhar Sai Thokala[1*]

[1]Independent Researcher

Received 01 Nov 2021, Accepted 10 Dec 2021, Available online 13 Dec 2021, **Vol.11, No.6 (Nov/Dec 2021)**

Abstract

Docker, developed and released in 2013 by Docker Inc. under the Apache 2.0 license, is an open-source container engine. Due to their significance in infrastructure virtualisation, containers have a special position in the annals of computing. This paper provides a complete discussion of Docker technology and how it has been applied in the current software development cycle, specifically in containerisation, to arrive at a highly efficient, repeatable, and portable application deployment method. Docker is an application container platform that packs code and dependencies together to run everywhere at scale. This capability is critical in the DevOps and CI/CD settings to cut down on deployment problems while using up less of the software development life cycle. Docker Essentials–Docker engine, Docker client-server, Docker images, Docker containers, and the role of these derivatives in constructing applications inside isolated containers are explained. Further, Docker utility in the attainable scalabilities is followed by frames such as Docker Swarm and Kubernetes that address the idea of horizontal scaling, Load Balancing, and features like microservices. The paper also revisits basic Docker tools and practices like Docker Compose and CI/CD for easier management of containers. A literature review addresses current research, emphasising gaps in Docker-based systems related to cloud environments and security. Last but not least, this paper provides a future work outlook which aims at improving the use of Docker in large-scale distributed systems.

Keywords: *Docker Container, Deployment, CI/CD pipelines, Load Balancing, Scalability, Kubernetes.*

Introduction

One of the most important parts of developing and deploying software nowadays is using Docker containers. Containerisation is a technology that Docker uses to package apps and their dependencies. Application source code, libraries, and dependencies may be packaged into a standardised executable component called a container. This component allows the app to execute in any environment. This approach also addresses the issue of difficulty that is involved when it comes to fulfilling the application deployment and improves on the duplication of builds of the software across the different computing platforms [1]. Docker is designed as an extension of this, where applications are isolated at the process level through containerisation technology that is lighter than full machine virtualisation than a virtual machine, utilising fewer resources and being faster to launch[2].

This reliability of Docker containers is even more critical from the DevOps' and CI/CD's perceptive[3], docker guarantees that an application performs as required when it is shifted between the development, testing, and production phases.

*Corresponding authors' ORCID ID: 0000-0000-0000-0000
DOI: https://doi.org/10.14741/ijcet/v.11.6.10

This brings the much-needed reproducibility to the mix to help avoid "works on my machine" issues and help to reduce the development life cylce. Furthermore, Docker isolates the software environment, which makes this tool truly valuable for developers striving to deliver code that will work in the same way regardless of the platform.

Scalability is another critical capability helped by Docker mainly through Docker Swarm and Kubernetes to ensure that new containers are created where needed across the large network [4][5]. Containers implemented in microservices architecture take advantage of Docker features concerning scalability. Docker containers can be easily containerised and deployed across multiple host systems efficiently for high-availability applications and microservices-based architecture patterns where there is a huge variability of application usage and, hence, a huge variability of the network load[6].

Additionally, the examples of its use together with various CI/CD tools describe how Docker contributes to shaping scalable web application deployment [7]. It also points out that Docker usage in CI/CD pipelines helps make the deployment and testing as flexible and reliable as possible, as Docker creates environments

that are similar to the production environments[8][9]. This capability not only leads to better quality and faster delivery of application releases but also makes the deployed applications more secure because all modifications are tested in a development environment first. The large-scale adoption of Docker implies that it provides a solution for both reproducibility and scalability challenges of SW development and deployment cycles[10].

Motivation of the Study

This paper is motivated by the need for efficient, reproducible, and scalable deployment in modern software development. Traditional models are resource-consumptive and sensitive to the environment, but Docker rises above this by running applications within disposable and minimally intrusive containers. Due to microservices' continuous integration and continuous delivery, Docker has become critical for agile development. This paper provides a comprehensive overview of Docker technology to support advancements in scalable, containerised solutions for modern deployment needs.

Organized of this paper

The paper is structured as follows: Section II overviews Docker technology, Section III provides the Docker Container Management Tool For Web Deployment, Section IV covers scalable deployment with Docker, Section V discusses Docker deployment tools, Section VI reviews related research, and the final section concludes with future work.

Docker Technology Overview

Technology known as "containerisation" organises application components, their dependencies, and system libraries into a single, cohesive unit. The developed and structured apps may run and be released in a container. This platform, which ensures that applications function in all environments, is called Docker. Additionally, the apps that will be deployed into containers are automated [11]. Applications are run and virtualised in a container environment with the inclusion of Docker's deployment engine [12]. Docker facilitates the efficient execution of programs by providing a lightweight and fast environment. What makes Docker tick are its four primary components: containers, clients, images, and engine [13]. The sections that follow will provide a detailed explanation of these components. Docker Container Architecture is shown in Figure 1.

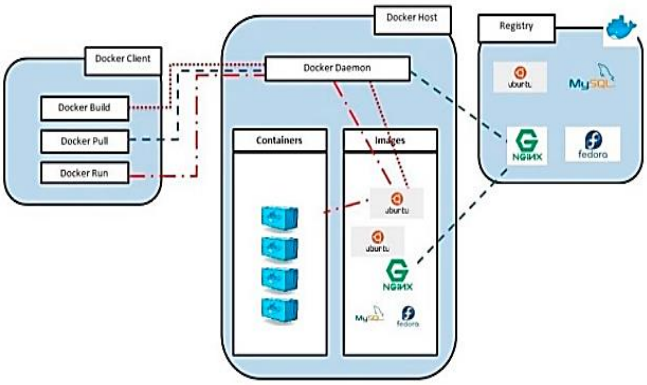
Docker Engine

Docker Engine is the central component of the Docker system. It is a client-server program that requires the following components to be installed on the host computer.

Docker Daemon: an executable software that runs continuously (the dockerd command) that makes it easier to build and run apps.

The Docker daemon is communicated with via a Rest API.

A client requests access to the operations from the docker daemon via the console.



Architecture of Docker Container.

Docker Client-Server

Docker technology relies on a client-server design. Docker daemon, the server process executing on the host computer, communicates with the client. The daemon's primary roles are to run the system, build containers, and distribute them. One computer may host both the container and the daemon for Docker. Figure 1 depicts the Docker architecture.

Docker Images

Two distinct approaches exist for creating Docker images. Using a read-only template to construct a picture is the main approach. The template is built using low-overhead operating system images, which may be CentOS, Ubuntu 16.04, Fedora, or any other lightweight base OS image [14]. Base photos are often the starting point for all photographs. Every time base pictures are made from scratch, a new image must be constructed [15]. "Committing a change" is the term for this kind of establishing a new image. The next step is to create a Docker file that contains all the necessary instructions for creating a Docker image [16]. When executed from the terminal, the docker build command will ensure that the created image has all of the criteria specified in the docker file. The term for this procedure is "automated image building"[17].

Docker Containers

A Docker image is what really creates a Docker container. The container must include all of the necessary components for the program in order to execute it in a constrained manner. Application or software service requirements might inform the creation of container images [18]. Imagine the following situation: an application that runs on the

Nginx server and Ubuntu operating system must be included in the docker file. The "docker run" command builds and starts a container with an Ubuntu OS image that includes the Nginx server.

Advantages of Docker Container

In recent years, Linux containers have grown in sophistication and popularity. The advantages offered by docker container are largely responsible for Docker's meteoric rise in popularity. The key benefits of docker are density, speed, portability, scalability, and quick delivery.

Speed

The speed of containers is one of their most highly praised features. When discussing the advantages of utilising Docker, it would be absurd to exclude mentioning both its speed and its functionality. Due to their compact size, containers may be constructed in a relatively short amount of time. The compact nature of containers allows for more rapid development, testing, and deployment. After construction is complete, containers may be moved to the testing environment for evaluation before being moved to the production environment [19].

Portability

Software developed in a Docker container is very portable. These mobile apps are lightweight and moveable, so you can take them with you anywhere without sacrificing performance [19].

Scalability

It is possible to install Docker on several physical servers, data servers, and cloud platforms. In addition, it is compatible with every Linux system. It is easy and fast to move containers between different cloud environments, local hosts, and back to the cloud. Simple adjustments allow the user to quickly and easily adjust the scale to their exact specifications [20].

Rapid Delivery

Docker Containers' standardised format eliminates the need for programmers to worry about each other's work. The administrator is in charge of setting up and maintaining the server with containers, while the developer is in charge of the applications that operate within the container. Because containers are tested and have all necessary dependencies integrated into the programs, they can operate in any environment [19].

Density

By eliminating the need for a hypervisor, Docker is able to maximise resource utilisation. That is why it is possible to run more containers on a single host than VMS. The greater density and absence of resource waste in Docker containers make them perform better [20].

Disadvantages of Docker Container

The following are some of the negative aspects of docker containers [21][22]:

The local host is responsible for providing the Linux kernel, so a docker cannot provide complete virtualisation.

Docker is not compatible with older computers at this time. Local computers that are 64 bits only are supported.

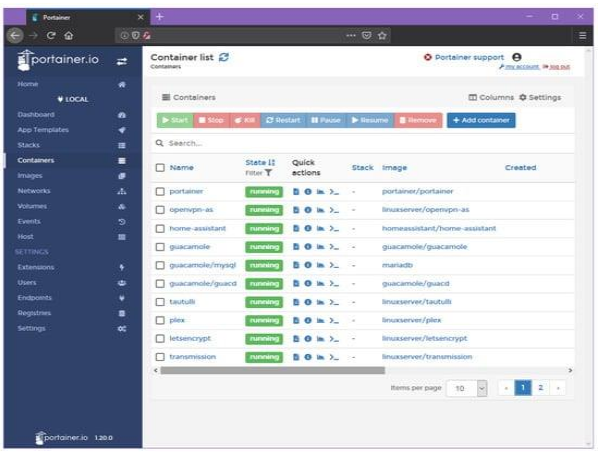
The Docker container is required to provide the whole virtualised environment for Mac and Windows computers. There has to be confirmation that the integration and performance with the hostmachine's OS are adequate and that user adoption of these systems is not hindered, even if the boot2docker software resolves this problem.

Evaluating the likelihood of security vulnerabilities is essential. Digitally signing docker images might make it simpler to build off of trusted binaries in the future.

Determining whether academics and researchers will really consider using Docker is a major challenge.

Docker Container Management Tool For Web Deployment

Programs without a graphical user interface (GUI) often run in the background using container technology. To facilitate user engagement, however, many programs need GUI functionality. The Docker Swarm Visualiser is an open-source project on Github that aims to provide a visualisation tool for Docker Swarm. Anyone using the host container may see this project. Having said that, it is useless for understanding overlay networks [23]. The platform monitoring technique that Docker employs is inadequate. The stat command is Docker's only monitoring option by default. This should only be used when very simple container data is required rather than complex monitoring.



Portainer Dashboard for Monitoring

An improved self-hosted data centre may be built with the help of Portainer, shown in Figure 2, which manages containerised Nginx web servers and is compatible with Kubernetes, Docker, and Docker Swarm, among other services. It is possible to swiftly deploy, monitor, and watch the behaviour of the web server containers and offer adequate and instant security whenever required by giving a command line interface (CLI) instead of the Docker graphical user interface (GUI). Application, cluster, registry/image, identity/access, storage, network, monitoring, and alert management are some of the extra capabilities that Portainer develops for infrastructure teams and developers.

Scalable Web Application Deployment with Docker

Scalability in web application deployment[10] is critical for handling varying loads and ensuring application availability and performance under increasing demands. Docker, with its containerisation technology, plays a pivotal role in enabling this scalability, offering solutions through various tools and methods such as Kubernetes and Docker Compose. This section delves into the core concepts of scalability, Docker's utilities in scalable deployments, the principles of load balancing, and a practical case study involving Docker and Kubernetes[24][25].

Understanding Scalability in Web Applications

Scalability in web applications means the ability of an application to handle growing loads proportional to the capacities [25]. This can be achieved through two main strategies:

Horizontal Scaling: By the term “scaling out” we mean making more computers or instances available. This method improves the flexibility of the application for request handling as the load is distributed in different hardware components. Horizontal scaling, also called scaling out, is more appropriate for stateless applications, where every server can handle requests separately [26].

Vertical Scaling: It is called scaling up when an increase in the processing power of a machine such as the CPU or RAM is done. Though this approach is easier to implement especially for additional capacity that is small, it sometimes has the limitation of capacity of one machine and is less scalable than the horizontal scaling [27].

Docker’s Role in Scalable Deployments

Docker simplifies both horizontal and vertical scaling through containerisation [28]:

Container Orchestration with Kubernetes: Kubernetes is a framework for orchestration that handles large-scale Docker container management. It automatically handles the scheduling and running of containers on a cluster of machines, with facilities for maintenance, scaling, and deployment patterns such as rolling updates and rollbacks. Kubernetes excels in horizontal scaling, offering robust, production-ready environments that integrate well with cloud services.

Docker Compose for Managing Multi-container Environments: Developers may create and manage Docker applications with many containers using Docker Compose. While it is traditionally used in development environments, Docker Compose can also simplify the management of multi-container setups in smaller production environments, facilitating easy scaling and integration of services.

Load Balancing and Scaling Containers

Load balancing is crucial for effectively distributing incoming network traffic across multiple backend containers to ensure reliable and consistent response times, maximise throughput, minimise response time, and avoid overload of any single resource. Docker, especially when integrated with orchestration tools like Kubernetes, provides built-in load balancing to handle the distribution of user requests efficiently[28]. This can be enhanced by using additional tools or plugins that manage more sophisticated load-balancing strategies, such as application-layer load balancers.

Case Study: Scalable Deployment Architecture with Docker and Kubernetes

Consider a case study involving a microservices-based application deployed using Docker and orchestrated with Kubernetes. This setup typically involves dividing an application into smaller, independent services that communicate over well-defined APIs[29]. These services are containerised, each running in its Docker container. Containers are managed over a cluster of computers by Kubernetes, which transparently handles deployment, scaling, and networking.

Example Deployment: Web application back end, which includes inter alia a users’ authentication service, data service, payment service and others, all of which are hosted in different containers. These services are deployed scalable using Kubernetes so that the resources can be adjusted to those services that are most required based on load[29].

Benefits Realized: This architectural model achieves excellent modularity, distributivity, and service independence, thereby enhancing the total quality of the system service [30].

Tools And Techniques for Docker Deployments

Docker has fostered simplification of complexities that surround the creation, distribution and execution of applications. This section discusses several important technologies that improve Docker distribution, with special attention to the orchestration, automation and integration concepts [13].

Docker Compose

Docker compose is a tool for defining and running Docker applications using many containers. The services, networks, and volumes of your application are configured using a YAML file when using Compose. After that, you may build and launch every service from your configuration with only one command.

Ease of Use: Docker Compose simplifies the deployment of multi-container applications by allowing developers to define their complex stack in a single file and then managing it all with simple commands.

Configuration: The YAML file used by Docker Compose provides detail and demonstrates the reproducibility of configuration, which is especially important during development, testing, and production.

Integration: It works well with Docker environments providing simple configurations for the local development and testing hence guaranteeing that the application operates the same way in different surroundings.

Docker Swarm vs. Kubernetes

Docker Swarm and Kubernetes are both the most popular orchestration platforms for Docker containers. While both tools claim to ease the construction and upkeep of containerised applications, they do so to somewhat different degrees [31][32].

Docker Swarm

Simplicity and Integration: Docker Swarm has been built on the Docker API and, therefore, is less complicated and easier to begin with as compared to Kubernetes. This makes it suitable for applications that are not very large, or even if you're just getting started with container orchestration.

Deployment: Setting up a Swarm is straightforward, where you turn a group of Docker engines into a Swarm cluster with a few Docker commands.

Kubernetes:

Scalability and Flexibility: Kubernetes is more capable and versatile, providing scalability, rollbacks and much more volume control in comparison to Docker. Herein It is best suitable for large scale enterprises.

Features: It has a number of features which can meet high demand of utilising it for large-scale applications, including load balancing, storage management, and automated deployment and undeployment.

CI/CD Integration with Docker

Modern development techniques cannot be accomplished without CI/CD pipelines, which stand for Continuous Integration and Continuous Deployment. Documentation can be simplified in the development process through automations of the building, testing, and running of Docker containers with help of continuous integration and delivery tools including Jenkins or GitLab CI [33].

Jenkins: An extensible open source CI/CD system that can be utilised to run any of the stages of your pipeline. Jenkins in a strategy of creating Docker images from Docker files, copying to registry as well as deployment to a Docker environment[34].

GitLab CI: Built into GitLab, it offers a Docker-first approach whereby each part of your CI/CD pipeline can be defined in gitlab-ci.yml, using Docker images as the environment for each stage of the pipeline[34].

Literature Work

Previous research in this area primarily utilises statistical methods and basic models to address issues related to Docker containers.

In, Naik (2017) This study explores an additional potential use case for Docker in big data analysis and proposes a solution based on Docker containers for processing massive amounts of data across many clouds. This Docker container-based system provides a low-cost and easy-to-use platform for anybody with a basic understanding of IT. Also, it's easy to create on one computer, many machines, or even several clouds. This article showcases the architecture and virtual development of the suggested large data processing system that uses Docker containers across several clouds. The next step is to illustrate the use of Hadoop's web-based GUI Hue to automate the setup of big data clusters using two popular big data analytics frameworks: Hadoop and Pachyderm (without Hadoop)[35].

In, Chelladhurai, Chelliah and Kumar (2016) Important security concerns with Docker containers have been covered in this paper, along with the associated research being done in this field. Additionally, we have suggested security algorithms and techniques to deal with Docker container technology problems linked to DoS attacks. The security techniques' first testing and trials show promise[36].

In, Mohallel, Bass and Dehghantaha (2017) The focus of this study was the relative security of running services on host-based operating systems vs inside Linux containers. Our paper's base operating system, Debian Jessie, and hosts executing services inside Docker containers were subjected to a battery of tests using Docker v1.10 to compare their attack surfaces. Using Docker containers actually increases a host's attack surface, according to our vulnerability evaluation [37].

In, Abdelbaky et al. (2015) Considering the needs and constraints of both users and resource providers, the paper proposes the C-Ports prototype architecture for managing and deploying Docker containers across various hybrid clouds and traditional clusters. In order to install containers on top of resources, the

framework makes use of Comet Cloud. It selects resources for allocation and dealslocation using a constraint-programming technique. A five-cloud and two-cluster dynamic federation has successfully deployed and managed containers using our prototype[38].

In, Molto et al. (2017) The paper describes a procedure for implementing open-source tool and standard-based coherent application delivery on HDCIs, where programs need to be delivered on both virtual machines and Docker containers. Application needs were described coherently by using and expanding the TOSCA standard and by implementing DevOps principles. This made it possible for the apps to run on a variety of systems. This method was used in the INDIGO-Data Cloud project, which is detailed in the article[39].

In, Chen et al. (2020) This article describes how to build a Docker container log collecting and analysis system utilising the widely used open source log collection system ELK, distributed message queue Kafka, and lightweight log collector Filebeat. Fast deployment made possible by the system's use of Docker container technology allows for real-time log collecting, data filtering and forwarding, presentation and analysis, and a significant improvement in the efficiency of operation and maintenance staff. Experiments validate the system's excellent availability, stability, extensibility, and real-time performance[40].

In, Long et al. (2020) the authors of this research suggest using Docker and Kubernetes in tandem to virtualise and deploy FPGA resources as lightweight containers. Docker containers encapsulate each application's execution environment and abstract FPGA resources. Virtualised FPGA containers are scaled and scheduled automatically by Kubernetes. This allows for the secure and efficient sharing of FPGA resources across several local and distant applications. Results from the experiments demonstrate an improvement in the utilisation of FPGA resources[41].

Below is Table I summarising the related work for utilising Docker containers for reproducible builds and scalable web application deployments. This table captures the varied applications and innovations in utilising Docker containers for reproducible builds, scalable deployments, and security enhancements.

Table 1 Summarizing the related work for utilising Docker containers for web application deployments

Paper Reference	Focus Area	Proposed Solution/Framework	Technologies Used	Key Features	Results/Findings
[35]	Big data processing in multiple clouds	Docker container-based big data processing system	Docker, Hadoop, Pachyderm, Hue GUI	Inexpensive, user-friendly, deployable on single/multiple machines or clouds	Demonstrates architecture and automated provisioning of Hadoop and Pachyderm for big data clusters.
[36]	Security of Docker containers	Security algorithms to address DoS attacks	Docker, custom security algorithms	Focus on improving container security and mitigating DoS attacks	Promising preliminary experiments for improving Docker security.
[37]	Container security versus host OS security	Vulnerability assessment of Docker v1.10 compared to host OS	Docker v1.10, Debian Jessie	Assessing the attack surface of Docker containers versus base OS	Containers increase the attack surface compared to running services directly on host OS.
[38]	Multi-cloud Docker container deployment and management	C-Ports: A prototype framework for multi-cloud container deployment	Docker, Comet Cloud, constraint-programming model	Dynamic resource allocation, hybrid cloud support, user-objective-driven constraints	Effective deployment and management across five clouds and two clusters.
[39]	Coherent application delivery across VMs and Docker containers	Workflow using TOSCA standard and DevOps practices	TOSCA standard, DevOps tools	Coherent creation of application artifacts, hybrid delivery support	Successful implementation in the INDIGO-Data Cloud project.
[40]	Real-time Docker log collection and analysis	ELK-based log collection and analysis system	Docker, ELK (Elasticsearch, Logstash, Kibana), Filebeat, Kafka	Real-time log collection, filtering, forwarding, and analysis; rapid deployment	Improved work efficiency, good real-time performance, extensibility, and high availability.
[41]	Virtualisation and deployment of FPGA resources using Docker and Kubernetes	FPGA virtualisation as lightweight Docker containers with Kubernetes orchestration	Docker, Kubernetes, FPGA	Isolated runtime for applications, automated scaling, and scheduling of FPGA containers	Enhanced utilisation of FPGA resources with better scalability and resource sharing.

Conclusion and Future Work

Docker has revolutionised Software Development and Deployment by offering a light and efficient approach to containerisation while providing a measure of portability for applications deployed in different surroundings. These are its principal components: Docker Engine, Client-Server, Images, and Containers; it is an efficient approach to building, testing and moving an app, managing it across various infrastructures. Compatibility with orchestration tools such as kubernetes and Docker swarm is also provided

for scalable deployments; the web applications and microservices are especially significant on Docker. Additionally, Docker integration with CI/CD pipeline enhances its development acceleration and allows for the reduction of deployment complications since the environments provided are closely similar to production ones. However, the literature review also shows that more of it is needed not only to enhance Docker's scalability and security but to tackle specific issues it encounters in multi-cloud and hybrid cloud environments too.

The subsequent research should pay more attention on improving Docker security, such as developing signed images' further security or perfecting the methodical check of vulnerabilities for containers on all the platforms. Another specialising consideration while operating Docker is that the use of a hybrid cloud setting that may possibly earn for elastic resources. There should also be studies for deeper management in the implementation of more complex origins such as Kubernetes and Docker Swarm. Making Docker compatible with older or lower-end systems would be a good thing if Docker wanted to be more widely adopted across different infrastructures. Moreover, evaluating Docker's contribution to new technologies, such as deploying AI models and edge computing, provides potential for using Docker's virtualisation to manage cost-effective hardware resources in these areas.

References

[1] S. Kwon and J. H. Lee, "DIVDS: Docker Image Vulnerability Diagnostic System," IEEE Access, 2020, doi: 10.1109/ACCESS.2020.2976874.

[2] R. Goyal, "The Role Of Business Analysts In Information Management Projects," Int. J. Core Eng. Manag., vol. 6, no. 9, pp. 76–86, 2020.

[3] C. Boettiger, "An introduction to Docker for reproducible research," 2015. doi: 10.1145/2723872.2723882.

[4] C. Pahl and P. Jamshidi, "Microservices: A systematic mapping study," 2016. doi: 10.5220/0005785501370146.

[5] V. V Kumar, "An interactive product development model in remanufacturing environment : a chaos-based artificial bee colony approach," MASTER Sci. Manuf. Eng., 2014.

[6] V. V Kumar, M. Tripathi, S. K. Tyagi, S. K. Shukla, and M. K. Tiwari, "An integrated real time optimization approach (IRTO) for physical programming based redundancy allocation problem," Proc. 3rd Int. Conf. Reliab. Saf. …, no. November 2014, 2007.

[7] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," IEEE Access. 2017. doi: 10.1109/ACCESS.2017.2685629.

[8] V. V. Kumar, A. Sahoo, and F. W. Liou, "Cyber-enabled product lifecycle management: A multi-agent framework," 2019. doi: 10.1016/j.promfg.2020.01.247.

[9] K. K. SKR Anumandla, VK Yarlagadda, SCR Vennapusa, "Unveiling the Influence of Artificial Intelligence on Resource Management and Sustainable Development: A Comprehensive Investigation," Int. J. Creat. Res. Thoughts, vol. 9, no. 12, pp. f573–f578, 2020.

[10] V. S. Thokala, "A Comparative Study of Data Integrity and Redundancy in Distributed Databases for Web Applications," IJRAR, vol. 8, no. 4, pp. 383–389, 2021.

[11] N. Richardson, R. Pydipalli, S. S. Maddula, S. K. R. Anumandla, and V. K. Yarlagadda, "Role-Based Access Control in SAS Programming: Enhancing Security and Authorization," Int. J. Reciprocal Symmetry Theor. Phys., 2019.

[12] M. Z. Hasan, R. Fink, M. R. Suyambu, M. K. Baskaran, D. James, and J. Gamboa, "Performance evaluation of energy efficient intelligent elevator controllers," 2015. doi: 10.1109/EIT.2015.7293320.

[13] A. M. Potdar, D. G. Narayan, S. Kengond, and M. M. Mulla, "Performance Evaluation of Docker Container and Virtual Machine," Procedia Comput. Sci., vol. 171, no. 2019, pp. 1419–1428, 2020, doi: 10.1016/j.procs.2020.04.152.

[14] V. K. Yarlagadda and R. Pydipalli, "Secure Programming with SAS: Mitigating Risks and Protecting Data Integrity," Eng. Int., vol. 6, no. 2, pp. 211–222, Dec. 2018, doi: 10.18034/ei.v6i2.709.

[15] V. K. Yarlagadda, S. S. Maddula, D. K. Sachani, K. Mullangi, S. K. R. Anumandla, and B. Patel, "Unlocking Business Insights with XBRL: Leveraging Digital Tools for Financial Transparency and Efficiency," Asian Account. Audit. Adv., vol. 11, no. 1, pp. 101–116, 2020.

[16] M. Z. Hasan, R. Fink, M. R. Suyambu, and M. K. Baskaran, "Assessment and improvement of elevator controllers for energy efficiency," 2012. doi: 10.1109/ISCE.2012.6241747.

[17] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An Introduction to Docker and Analysis of its Performance," IJCSNS Int. J. Comput. Sci. Netw. Secur., 2017.

[18] M. Z. Hasan, R. Fink, M. R. Suyambu, and M. K. Baskaran, "Assessment and improvement of intelligent controllers for elevator energy efficiency," 2012. doi: 10.1109/EIT.2012.6220727.

[19] A. A. Folarin, R. J. Dobson, and S. J. Newhouse, "NGSeasy: a next generation sequencing pipeline in Docker containers," F1000Research, 2015, doi: 10.12688/f1000research.7104.1.

[20] A. M. Joy, "Performance comparison between Linux containers and virtual machines," 2015. doi: 10.1109/ICACEA.2015.7164727.

[21] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," 2015. doi: 10.1109/ISPASS.2015.7095802.

[22] A. S. Harji, P. A. Buhr, and T. Brecht, "Our troubles with Linux Kernel upgrades and why you should care," ACM SIGOPS Oper. Syst. Rev., 2013, doi: 10.1145/2506164.2506175.

[23] M. Brouwers, "Security considerations in Docker Swarm networking," System and Network Engineering, University of …. 2017.

[24] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: A state-of-the-art review," IEEE Trans. Cloud Comput., 2019, doi: 10.1109/TCC.2017.2702586.

[25] M. List, "Using Docker Compose for the Simple Deployment of an Integrated Drug Target Screening Platform," J. Integr. Bioinform., 2017, doi: 10.1515/jib-2017-0016.

[26] H. Rajavaram, V. Rajula, and B. Thangaraju, "Automation of Microservices Application Deployment Made Easy By Rundeck and Kubernetes," 2019. doi: 10.1109/CONECCT47791.2019.9012811.

[27] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation," IEEE Cloud Comput., 2017, doi: 10.1109/MCC.2017.4250931.

[28] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a

Cloud-Native Architecture," IEEE Software. 2016. doi: 10.1109/MS.2016.64.
[29] M. H. Fourati, S. Marzouk, M. Jmaiel, and T. Guérout, "Docker-C2A: Cost-aware autoscaler of docker containers for microservices-based applications," Adv. Sci. Technol. Eng. Syst., 2020, doi: 10.25046/aj0506116.
[30] D. Kim, H. Muhammad, E. Kim, S. Helal, and C. Lee, "TOSCA-based and federation-aware cloud orchestration for Kubernetes container platform," Appl. Sci., 2019, doi: 10.3390/app9010191.
[31] N. Marathe, A. Gandhi, and J. M. Shah, "Docker swarm and kubernetes in cloud computing environment," 2019. doi: 10.1109/icoei.2019.8862654.
[32] A. Modak, S. D. Chaudhary, P. S. Paygude, and S. R. Ldate, "Techniques to Secure Data on Cloud: Docker Swarm or Kubernetes?," 2018. doi: 10.1109/ICICCT.2018.8473104.
[33] K. Brady, S. Moon, T. Nguyen, and J. Coffman, "Docker Container Security in Cloud Computing," 2020. doi: 10.1109/CCWC47524.2020.9031195.
[34] Malathi. S | Ganeshan. M, "Building and Deploying a Static Application using Jenkins and Docker in AWS," Int. J. Trend Sci. Res. Dev., 2020, doi: 10.1007/978-3.
[35] N. Naik, "Docker container-based big data processing system in multiple clouds for everyone," 2017. doi: 10.1109/SysEng.2017.8088294.
[36] J. Chelladhurai, P. R. Chelliah, and S. A. Kumar, "Securing docker containers from Denial of Service (DoS) attacks," 2016. doi: 10.1109/SCC.2016.123.
[37] A. A. Mohallel, J. M. Bass, and A. Dehghantaha, "Experimenting with docker: Linux container and baseos attack surfaces," 2017. doi: 10.1109/i-Society.2016.7854163.
[38] M. Abdelbaky, J. Diaz-Montes, M. Parashar, M. Unuvar, and M. Steinder, "Docker Containers across Multiple Clouds and Data Centers," 2015. doi: 10.1109/UCC.2015.58.
[39] G. Molto, M. Caballer, A. Perez, C. De Alfonso, and I. Blanquer, "Coherent Application Delivery on Hybrid Distributed Computing Infrastructures of Virtual Machines and Docker Containers," 2017. doi: 10.1109/PDP.2017.29.
[40] L. Chen, J. Liu, M. Xian, and H. Wang, "Docker container log collection and analysis system based on ELK," 2020. doi: 10.1109/CIBDA50819.2020.00078.
[41] X. Long, B. Liu, F. Jiang, Q. Zhang, and X. Zhi, "FPGA virtualization deployment based on Docker container technology," 2020. doi: 10.1109/ICMCCE51767.2020.00109.