International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(RESEARCH ARTICLE)

Check for updates

Containerization in cloud computing: comparing Docker and Kubernetes for scalable web applications

Bangar Raju Cherukuri *

Department of Information Technology, Andhra University, India.

International Journal of Science and Research Archive, 2024, 13(01), 3302–3315

Publication history: Received on 15 September 2024; revised on 26 October 2024; accepted on 29 October 2024

Article DOI: https://doi.org/10.30574/ijsra.2024.13.1.2035

Abstract

Containerization has taken cloud computing to a new level where application developers can develop an attractive way of deploying portable web apps. This paper compares Docker, one of the most popular containerization technologies, and Kubernetes based on their ability to manage cloud resources, advanced integration, and microservices implementation. Docker, which produces lightweight containers, reduces complexity in the application delivery process by including all the software components and their configurations. Kubernetes, in contrast, stands tall as a manufacturing tool for the automation of containerized application deployment, scaling, and management in clusters.

Here, the study's objective is to give a comparative analysis of Docker and Kubernetes to understand the best course of action. The research thus integrates a quantitative assessment of the performance of different algorithms based on simulated data with several case studies in various sectors, including e-commerce and software as a service (SaaS). Basic parameters of deployment velocity, resource consumption, system growth capabilities, and robustness of the two platforms are considered here as comprehensive measures.

The present study reveals that Docker can provide simple, portable solutions for small, small, and medium-scale applications. Kubernetes offers excellent orchestration solutions that are better for large-scale and complex applications. The study concludes that Docker and Kubernetes are suitable whenever specific project demands are involved. However, Docker stands out in a basic project setup and configuration, while Kubernetes functions better within the multi-cloud/ microservices conditions. The findings of this research are useful in informing organizations on their best practices when using containerization.

Keywords: Application developers; Kubernetes; Docker; Containerization; SaaS

1. Introduction

1.1. Background to the Study

A containerization is a solid characteristic of contemporary cloud computing, as it defines how to place an application and its requirements in a single vessel. This has ultimately changed how web applications are developed, deployed, and controlled in different settings. Pre-containerization, the task of redistributing the applications and services, meant compatibility issues that reduced efficiency and an inconsistent platform (Pahl, 2015). Containers address these problems by packaging the software with the expected contexts and all the libraries and configurations that go with it.

Docker's containerization suggestions led to business solutions by introducing lighter containers that can be easily built, deployed, and managed. Docker containerizes applications by running them in a Docker container on top of the host

* Corresponding author: Bangar Raju Cherukuri

operating system. It has something in common with other containers regarding the kernel, but it doesn't hold the content of the other, and the latter doesn't hold the former either. This approach is cheaper than traditional virtual machines due to the flexibility of the availability of resources espoused by Docker containers (Merkel 2014). Today, developers use Docker widely in enterprises developers use to build and deploy scalable applications.

Since containerization gained popularity, multiple containers must be operated in various states. This resulted in creation of the container orchestration platform, the most stable and popular Kubernetes. Kubernetes is an application that has been developed to manage the operation of applications in a set of computers that are running in containers, and it exists to facilitate the deployment of such applications across the available computers as well as the scaling of the applications in operation. Thus, orchestrating containers makes complex microservices configuration possible with Kubernetes assistance and ensures high availability with load balancing (Hightower et al., 2017).

The need for a flexible approach to application deployment across several cloud platforms has also motivated the selection of containerization. Since the different cloud providers offer various services and infrastructures, organizations need a viable and possible way of porting their applications from on-site-based infrastructure to cloud-based ones. Containerization gives this flexibility since applications may be packed once and deployed at any location, irrespective of the structure of the underlying hardware (Pahl, 2015). This capability has been especially useful for hybrid or multi-cloud consumer organizations because they lower the propensity for vendor lock-in and are less costly.

1.2. Overview

Today, many technologies like docker containerization technologies and Kubernetes have greatly transformed how developers create and deploy web apps. Docker, which arrived in 2013, makes it easier to package applications with its dependencies, and the resulting containers are light and can be run in any environment. This feature helps to overcome the "it works on my machine" issue since the containers' functioning is consistent across the developer's local environment, on-premises servers, and the cloud (Merkel, 2014). Containers are intended to be lightweight, agile, and confined, so Docker containers are small, start quickly, comprise a shared host OS Kernel, and consume fewer resources than a Virtual Machine.

Kubernetes works with Docker as a container orchestrator. Although Docker offers the methodologies for creating the containers and utilizing them in processes, Kubernetes is responsible for overseeing the processes of container distribution, growth, and running across clusters. Kubernetes does this through load balancing for applications, which will maintain and spread the application components across available resources (Hightower et al., 2017). Kubernetes is responsible for keeping track of containers' status, including the possibility of restarting built-in containers when this is needed or increasing their analytical capacity due to the influx of more requests from the obtained load-balanced Internet addresses. Due to this level of automation, the Kubernetes system is well-suited for large systems comprising numerous sub-applications based on the microservices model.

In this paper, both Docker and Kubernetes demonstrate the important attributes that enable the efficient deployment of web applications. Flexibility or portability is one of the key benefits that come with it. Compared to other solutions, containers can be easily moved from one environment to another, including a more suitable multiple-cloud application development, test, and deployment solution. Another main advantage concerns flexibility: containers are reusable and often created so that if the application increases or decreases, they can be made larger or smaller without taxing the system. Kubernetes does exceptionally well in this aspect because it has availability and horizontal scaling characteristics that continually alter resource allocation based on current traffic (Burns et al., 2016).

One particularly significant characteristic of these technologies is the automation of the global manufacturing process. Docker makes creating and assembling containers easy, whereas Kubernetes helps manage them more effortlessly. It is beneficial to a CI/CD system since it makes it easier for developers to deploy updates and Patches through several environments that do not disturb the application. Further, Kubernetes supports system roll updates, which makes it possible to safely deploy versions with just some instances available for a duration of time up to a maximum of zero (Hightower et al., 2017).

1.3. Problem Statement

Web application deployment scaled in clouds has always been a problem, especially when containers have not been used. The main classic methods imply the deployment of virtual machines or separate servers that may cause the ineffective utilization of resources. These approaches are time-consuming, involve a lot of manual work, and have issues with reproducibility because software could work differently in different environments. This incompatibility poses

additional challenges to the development and deployment process. Fixing confidence that other applications will run consistently across the development, deployment, and production stages becomes challenging.

Further, the issues are complex in the current era of web applications that depend on the microservices architecture. Microservices depend on the efficient integration of multiple independently deployable modules, a feat that can be challenging to deal with without in place. Existing approaches can be problematic in scalability, load, distribution, and automatic failover, which causes performance bottlenecks and system breakdowns.

Containerization tools like Docker and Kubernetes have improved this aspect when deploying web applications. Docker helps develop, ship, and deploy applications in environments devoid of variations, while Kubernetes manages to manage the scale at the same time; it is critical to realize and recognize the strengths and weaknesses of each tool for those organizations that strive to get the most out of the cloud infrastructure. The comparison of Docker and Kubernetes facilitates an understanding of which platform fits better when addressing problems such as scalability, resource utilization, and operation efficiency to enable businesses to make the right decisions on how to deploy their web applications.

1.4. Objectives

- To compare Docker and Kubernetes regarding the effectiveness of scaling out web applications,
- To support the subsequent assessment of the above strategies' effectiveness in containerizing cloud resources.
- To evaluate the advantages of using Docker and Kubernetes for CI and Microservices.
- It was imperative to establish the main goal: how does one correctly use Docker and Kubernetes in various clouds?
- Compare modulin terms of movability and the availability of canned support options.

1.5. Scope and Significance

As such, this study focuses on assessing Docker and Kubernetes as two of the state-of-the-art technologies suitable for improving the cloud-like scalability of web applications. It will focus on three key areas: Typically, the domains are cloud resource management, CI/CD methodology, and the microservices architectural pattern. Cloud resource management entails optimizing computing resources such that applications can be scaled up or down without calling for an excess of the same. This work will evaluate how Docker and Kubernetes prioritize and distribute resources, manage the load, and scale up the cloud environment.

For the CI/CD concept, the research will thus examine how those tools enhance the development and delivery process. Therefore Docker guarantees environments are consistent with development/deployment; Kubernetes conversely helps in updating and deploying apps as it offers most tools for automation-i.e., Scaling and monitoring. Knowledgeful about them is necessary if organizations are interested in shortening the deployment time and enhancing its efficiency.

Both of them are hereby presented, as well as the consequent intent of this research is to add to the current body of knowledge that could help in controlling the factors applied in the modern application release. Therefore, after giving the result of the comparison between Docker and Kubernetes, the study will also provide the challenges and opportunities of the two platforms so that the current and potential users of the two technologies would be in a better position to give a conclusion of the relevance of the technology in organizations. Further, the work will elaborate on container orchestration applications in multi-cloud environments where portability and flexibility are critical drivers in avoiding vendor lock-ins and cost optimization. This study aims to equip organizations with the ability to achieve optimal value and efficiency from Clouds while maximizing scalability.

2. Literature review

2.1. Evolution of Containerization

Containerization has been through major changes in how cloud computing platforms build, deploy, and run applications. The first wave enabled by virtual machines or VMs works by emulating the hardware platform and executing different OS's in isolation from the actual computers. As much as VMs provided resource isolation and security, there was always a problem because they were usually a bad match regarding the memory and processing power overhead Soltesz et al., 2007.

Containers were deemed more scalable and high-performance compared to conventional virtual machines. Unlike hardware, containers are on top of a shared operating system kernel, which provides lightweight isolated encapsulation of applications with lesser consumption. This approach allowed the application to start cooler time, memory consumption, and scalability improvements. Technological advancement in the area began with Solaris Containers and LXC, later developing into the currently known container technology (Pahl, 2015).

The beginning of this containerization revolution was in 2013 through the Docker. Docker presented a multipurpose platform simplifying the creation and running of container applications and their packaging. It rationalized container deployment by packaging an application and all that it requires alongside the container and environment (Merkel, 2014). Portable Docker allows the applications to be developed and deployed in different systems ranging from computers, data centers, and cloud service providers.

With container usage increasing, the requirement for handling many containers across different systems has emerged. Google developed Kubernetes and released this platform as an open-source platform in 2014, and it achieved this necessity. Kubernetes is an orchestration tool to maintain and manage applications as a set of containers across a cluster of hosts, including unique features such as load distribution, self-repair, and gradual update of all replicas (Burns et al., 2016). One advantage that has made it the most popular container orchestrator for microservices in cloud computing is its capability to handle large structures.

2.2. Docker: Containerization and Portability

Docker is considered an important key factor in the discourse of containers as a platform through the facilitation of application packaging. Docker was launched in 2013 to provide an innovative structure for developers to package, ship, and run their applications alongside their environment subsections, final versions of libraries, and configurations, all in a single container format. This makes it easy to relocate to various institutions; thus, it can be operated wherever it is placed, and its functionality is consistent in each place (Merkel, 2014). Most significantly, Docker's plain and lightweight architecture makes it one of the most popular tools in modern cloud computing, focusing on flexibility and scalability.

Docker's outsourced value relies on its ability to run applications in individual containers with the applications sharing the same operating system kernel. Traditional virtual machines may have different instances with particular operating systems and, therefore, much less efficient resource usage than Docker containers, which share the host OS (Pahl, 2015). This lightweight approach brings scalability, higher performance, lesser overhead, and faster response times than traditionally tiered applications, especially in environments where applications must grow rapidly and be deployed quickly.

The degree of portability is another advantage of Docker. That way, Docker guarantees that a container is portable and can run anywhere it is needed, from a developer's local workstation to a data center or directly on AWS or Google Cloud. This ability to transfer containers from one environment to another without alteration means that compatibility risks are minimized, making Docker perfect for CI/CD processes (Merkel, 2014). The developers can also create, test, and deploy applications more simply and efficiently using containers.

Additionally, Docker makes it easier to work with microservices, an application suite of services that interact with one another. This is because each microservice can be developed as a separate container. Hence. Thus, if one service has to be altered, upgraded, scaled, or run in a different fashion, it can always be done independently. This flexibility is especially valuable in organizations planning to implement demand or technology changes through agile development (Pahl, 2015).

However, with Docker, there is a lot that one can achieve towards solving the problems of managing containers at scale. With more containers in use, tracking where they are applied, how to scale them up, and how to connect them is challenging. Therefore, This has triggered the emergence of platforms for orchestrating and striving for ways to ensure containers are deployed in the correct systems.

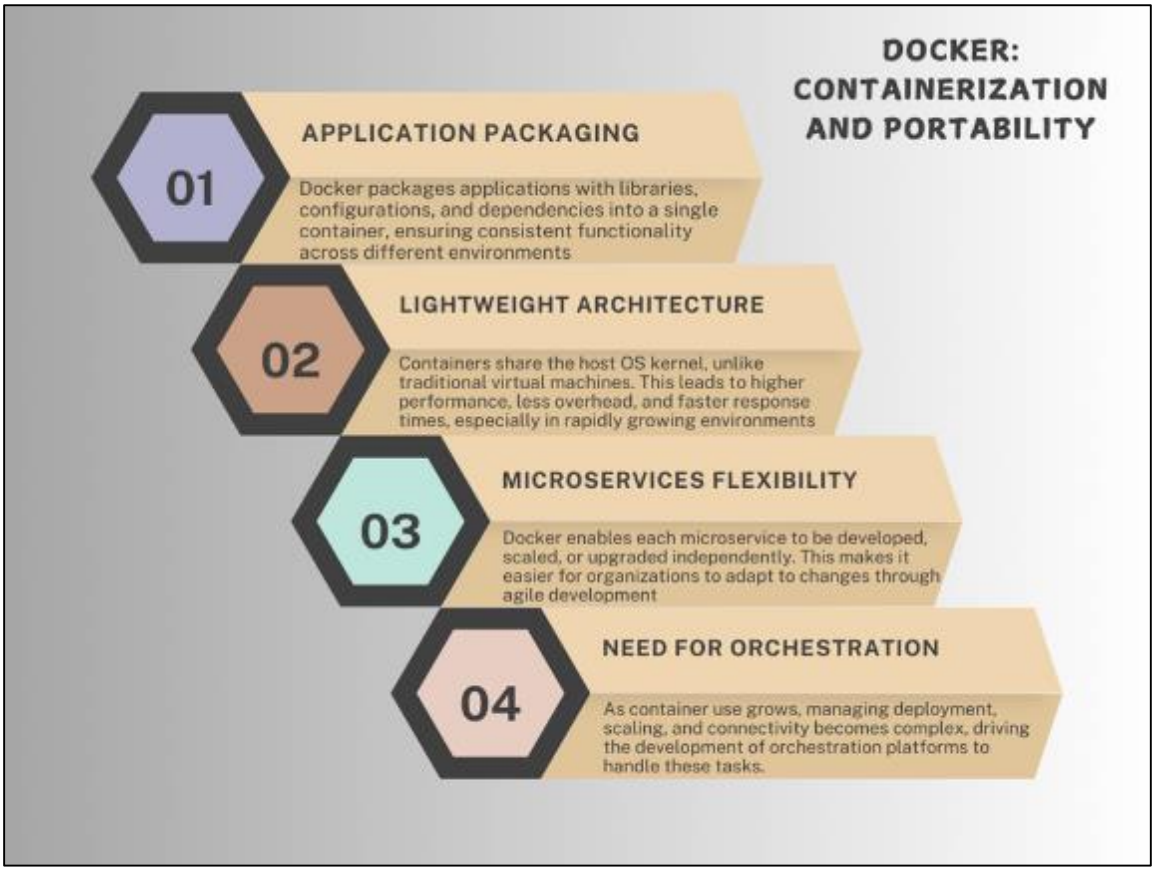

Figure 1 An image illustrating Docker: Containerization and Portability

2.3. Kubernetes: Container Orchestration for Scalability

Kubernetes was developed by Google and open-sourced in 2014 as an application container orchestrator, which has since become the most popular platform for running complex container applications. When organizations shifted from a single large monolith application to many small independent services, the requirement of a strong container was recognized. Kubernetes platform can provision, scale, and manage containers across the clusters in a way that makes applications run, dependable, reachable, and optimized (Hightower et al., 2017).

Number three, an important area where Kubernetes excels is with питань managing automatic deployment. It helps to implement the new versions of an application by organizing rolling updates and canary implementations; thus, updated services can be implemented with maximum availability. As Burns et al. (2016) note, if there's a hitch during the update process, Kubernetes can revert this change, making the application more stable.

The other important characteristic of Kubernetes is known as the scalability of the system. There is a horizontal pod autoscaler so applications can increase or decrease their instance at different utilization rates. This feature tracks the amount of CPU and memory used and scales the amount of uprunning containers or 'pods.' This means that using multitenant architecture, the application resources can be correctly distributed, the optimum response can be provided during high-loaded periods, and the least amount of resources can be used in periods of low usage (Pahl, 2015).

Kubernetes also performs well in monitoring and the possibility of automatic problem detection and problem-solving. It periodically monitors the health of containers and self-heals them by either restarting the container or replacing it with a new one in case of a bad state. This self-healing capability will provide high availability and fault tolerance, and therefore, Kubernetes is appropriate for large-scale and business-critical applications (Brewer et al., 2017).

There are some other noteworthy features of Kubernetes, which also make it suitable for the modern world, where multi-cloud and hybrid-cloud are sailing the seas: Victoria and areata to make a transition and switch between different cloud providers freely without draining into a single provider's cage. Companies must optimize various cloud providers' costs and particular features (Hightower et al., 2017).

2.4. Comparison of Docker Swarm and Kubernetes

The two solutions applied for managing container applications are Docker Swarm and Kubernetes but the difference was found based on the function, scalability, simplicity, and ecosystem support. Docker Swarm is a native orchestration tool from Docker and is tailored for easy integration with the Docker environment. On the other hand, one of Google's products — Kubernetes — provides a richer set of orchestration capabilities and better supports complex big-scale applications.

Another distinct aspect that must be mentioned is scalability – Kubernetes outperforms Docker Swarm here. It is designed for large solutions environments and is very efficient in application orchestration within thousands of containers. It is offered with functionalities such as horizontal scaling, where the containers can be scaled depending on the usage of the resources; it also provides load balancing inherent within the pods to share traffic among nodes (Hightower et al., 2017). Docker Swarm can be set up rather easily; however, it was not designed to manage highly complex applications, as was the case with the Docker. It is extolled for less extensive implementations, simplicity of usage, and compatibility with Docker compared to the ability to utilize sophisticated orchestration (Merkel, 2014).

Turning to the issue of usability, docker swarm has the advantage. This is tightly interwoven in Docker CLI, so it's easier to deploy containers if you already have a Docker background. Docker Swarm is somewhat easier to learn, while Kubernetes is easier to set up and has several preconditions to be met before use. Nonetheless, some elements cannot be achieved in Docker Swarm, like self-healing, rolling back, and monitoring, available in Kubernetes (Pahl, 2015, p. 1529).

The other fundamental difference is that much ecosystem support is present in machines. Kubernetes is very popular and well-documented, with vast resources available from contributors in the community. For this reason, more enterprises will use it for big application deployment. Hybridized Cloud SD-WAN provides extensive compatibility with most cloud providers like Amazon Web Services, Google Cloud, and Azure and can be easily integrated with multiple hybrid clouds (Brewer, 2017). On the other hand, Docker Swarm has a less complete ecosystem and is used less often for managing complex large-scale environments.

2.5. Containerization in Continuous Integration and Continuous Deployment (CI/CD)

Docker and Kubernetes can be game changers in Context Integration and Context Deployment (CI/CD) since they provide a more efficient, reliable, and scalable way to orchestrate and deliver a specific application. Docker manages the packaging process and presents an application, its codebase, and dependencies in portable containers that would run the same regardless of where they are hosted. This causes the applications to behave in a similar manner in the development, test, and production environment, reducing the infamous 'it works on my machine' problem (Sharma et al., 2016). One of the advantages that stem from using containers is that changes are easily made and passed through the development process without having to worry about environments.

Kubernetes goes further by coordinating Docker containers, which means that it ceilingly automates container-based applications' deployment, scaling, and running. Kubernetes is also used in CI/CD pipelines to perform rolling updates and canary deployments, thus reducing the time taken while updating services. Similar to the standard updates, rolling updates do not update all the instances at once; the master selects a subset and updates them while others stay active, leading to minimal disruption (Hightower et al., 2017). This way, with failures, Kubernetes can redo the deployment, which makes the system more reliable.

Docker and Kubernetes are important when using various CI/CD tools such as Jenkins, GitLab CI, and CircleCI. For example, Jenkins pipelines assemble Docker images after every commit of the code and deploy these images on the KS or KS pipeline, which means code integration to the deployment process (Merkel, 2014). This integration enhances quick feedback cycles so developers can detect and correct problems in the development stages.

Furthermore, Kubernetes has this autoscaling feature to enable applications to manage sudden loads required for modern cloud-native applications. This integration also ensures that applications of the CI/CD pipeline are deployed, available, and scalable at a very high rate (Burns et al. 2016). Also, with Kubernetes, we have the self-health check, where containers can be automatically restarted or replaced when the fail in the application, making the application more stable.

2.6. Microservices Architecture and Containerization

This concept is a major driver of microservices application architecture, which involves partitioning an application into multiple relatively autonomous components. This architecture also provides element-developer flexibility for development teams to build and deploy parts of the system individually for better malleability and expansion. Additionally, Docker and Kubernetes have a critical role in working with microservices since these two frameworks offer (Pahl et al., 2019) a standardized way in which to package and manage microservices:

In the first place, Docker and Kubernetes preserve the integrity of microservices since these two frameworks offer a standardized approach to how they can be loosely coupled and handled as individual components.

With the MS architecture, each service can be independent and contained in a Docker where all required dependencies are present. This isolation reduces the interaction of services with each other and makes the implementation process to be easy. This makes microservices very portable since Docker's strength enables microservices to run on different environments, including developers' local computers, physical data centers, and cloud environments (Merkel, 2014). This flexibility is important in organizations that seek to implement microservices because it enables individual services to change independently without having a domino effect on the other parts of the application.

Kubernetes and Docker are integrated, and Kubernetes is an orchestration tool that handles these containerized microservices at scale. Another mechanism that uses Kubernetes to manage microservices is the capability to deploy them across the clusters and scale them based on the racket while detecting failures easily. In every microservice, there is always a provision for service discovery and load balancing to enable microservices to call other services without the hassles of external configuring. However, operational constraints may apply (Burns et al., 2016). For instance, Kubernetes can provide internal DNS by automatically assigning the name of the services to the containers so that containers can easily communicate.

Moreover, Kubernetes provides rollout techniques like rolling updates and canary deployments that help teams run updates for microservices without conducting a whole application update. This capability is essential in the microservices where independent services are frequently updated. Kubernetes also ensures minimum disruption occurs thanks to running new versions of a microservice alongside the old versions while slowly routing traffic to the latest version (Hightower et al., 2017).

2.7. Challenges in Containerization for Scalable Applications

Nevertheless, looking at the key advantages of containerization based on Docker and Kubernetes, several problems with scalable application deployment based on these technologies should be mentioned. The greatest difficulty is that of network configuration. When all these thousands of containers are put into large-scale applications, controlling the traffic between them can be quite challenging. Kubernetes has a CNI that supports the formulation of network policies, but setting consistent, secure, and high-performance networks across many clusters is a complex issue (Zhang et al., 2018).

Security is another critical aspect of containerization environments that organizations must be concerned with. Docker containers are contained in their running in the host OS kernel, meaning that the root of the problem might be an attack on the kernel. Kubernetes adds an extra level of complexity because containers in Kubernetes pods are network-enabled, which means they're endpoints across nodes. These risks require significant security measures such as network segmentation, vulnerability scan, and protection at the container runtime (Pahl et al., 2019).

Managing resources when scaling containerized applications is also a challenging task. Facilities like horizontal pod autoscaling and resource quotas are available in the Kubernetes. Still, there are challenges that a container faces, including over usage of CPU and memory in distributed clusters, which demands ongoing monitoring and tuning (Brewer, 2017). If the resources are not appropriately distributed, they cause degradation in application performance, especially during traffic surges or high loads.

In addition, the data management of containers is challenged by their ability to run indefinitely or until stopped. Although containers are not expected to have a state, several programs and procedures need a state for databases and file systems. Kubernetes uses the PVS (Persistent Volume), allocated to Pvc (Persistent Volume claim) for storage solutions. Still, integration of these volumes in multi-cloud and hybrid environments takes a lot of work. Ensuring data is mirrored, the backup is created, and the data is available in the other cloud providers is an important aspect of high availability and disaster recovery in the cloud environment.

Figure 2 An image illustrating the Challenges in Containerization for Scalable Applications

3. Methodology

3.1. Research Design

This research will use quantitative and qualitative methods to assess the feasibility of Docker and Kubernetes for deploying scalable web applications. Quantitative performance comparisons will apply general assessment metrics of deployment time, resource consumption, and scaling as each platform's handling of containerized environments is compared. Further, qualitative case studies will be incorporated into the study to present actual organizational practices of these technologies with their advantages and difficulties. Such an approach will allow the elaboration of detailed comparisons between Docker and Kubernetes to consider their potential and existent weaknesses when used in certain scenarios.

3.2. Data Collection

Data sources for this study will include case studies, questionnaires, and interviews. The case studies will look at organizations that have deployed Docker and Kubernetes successfully, and based on the case, a report will be constructed based on their strategy, performance, and effectiveness. Furthermore, surveys will be conducted with industry members to gain more general opinions on the application of such platforms. To enrich the quantitative data collected through the surveys, face-to-face interviews will be administered to a set of experts in the field based on the perception. Docker and Kubernetes and real-life experiences of the advantages and difficulties arising from the applications of DockerDocker and Kubernetes. It is important to consider sources that will provide the practical implications of containerization technologies, thereby arriving at multiple sources as the best approach.

3.3. Case Studies/Examples

The two solutions of Docker and Kubernetes have stood out as allowing versatile organizations to control scalable web application solutions that improve deployment schemes, assets, and size. This section focuses on particular examples of the e-commerce and Software as a Service (SaaS) sectors to show how these tools can be beneficial in overcoming diverse organizational issues.

3.3.1. E-commerce Platform: Kubernetes Integrated with Shopify

Shopify, an e-commerce platform that serves millions of merchants, had issues managing high traffic rates, especially during the sales rush like Black Friday. Before engaging Kubernetes, the organization faced challenges managing the resources and scaling activity and experienced service variability during peak periods (Burns et al., 2018). At Shopify, Kubernetes deployment, scaling, and managing containerized services enabled full workload distribution across the organizations.

Horizontal pod autoscaling of Kubernetes enabled scaling of services by demand while maintaining guaranteed availability without allocating excess resources. This made it easier to automate the process, meaning less manual approval and, hence, less cost incurred. Furthermore, the self-healing feature that ran on Kubernetes enhanced reliability because the dead ones could be restarted or replaced with others, greatly reducing the time spent due to adversities (Kelsey et al., 2018). Kubernetes generally allowed Shopify to work with increased loads without issues during high sales traffic.

3.3.2. SaaS Platform: Atlassian's Docker Integration for CD

Confluence and Jira maker Atlassian embraced Docker in its CI/CD systems to solve problems like environmental stability and deployment Runtime. At that time, Atlassian developers complained that the applications' behavior was different in development, testing, and production environments, causing a long deployment cycle. On the other hand, Docker overcame this problem by creating application containers with the same specifications and requirements for every climate into which they were built and deployed, making the development and deployment processes relatively easier (Merkel, 2014).

This integration enabled Atlassian to automate Docker from building testing to service deployment. Every microservice was deployed inside its vessel, eliminating overlapping and cutting down on interconnection. This allowed the ability to make and test these containers portable from the developer's environment, through staging, and to the production environment. This new approach made further releases quicker and coordinated the integration of development, testing, and operational teams, thus enhancing the activity efficiency and stability of the developed software (Turnbull, 2016).

3.3.3. Hybrid Approach: Expedia's Combined Use of Docker and Kubernetes

Since Expedia is a global online travel company with infrastructure including microservices, Docker and Kubernetes were used to manage the infrastructure. Docker helped by stating that each microservice could be bundled individually while guaranteeing that all the plates would behave consistently regardless of development, staging, or production. Kubernetes managed these containers, including scaling and load balancing across distributed systems (White et al., 2019).

This was helpful for Expedia, especially Kubernetes' multi-cloud support, where workloads are easily spread across multiple providers to avoid cloud lock-in. The Ooyala's rolling updates and the canary deployments attributes helped to prevent service disruptions and made software releases easy through progressive updates. This approach of using microservices proved beneficial to Expedia. It is as it quickly increases or decreases services depending on customer demand, increases tolerance to faults, and improves operations, which could be valuable in fitting new features and responding to customer needs.

3.4. Evaluation Metrics

The assessment of Docker and Kubernetes will be mostly based on the seven vital measurements for defining the efficiency of one's large-scale web applications. Some of these measurements are deployment speed, which is the period it takes to deploy the applications across the environments, and scalability, which is the platforms' ability which is the platforms' ability to handle congestion by appropriately managing the available containers and resources, especially as they deal with increased workloads.

Resource utilization is another one since it compares how each platform utilizes the system's resources – CPU and memory, among others – to guarantee that applications run smoothly without excess allocated resource consumption. Studying network efficiency will help investigate how Docker and Kubernetes implement how containers communicate with each other and clusters in large-scale applications.

High availability and reliability are achieved by fault tolerance. In this research work, the reaction of each platform to possible failures, such as the possibility of an automatic restart or replacement of failed containers, will be assessed. However, the Ci/CD integrations and supporting multiple clouds are important and will be examined in this work. Comparing these measurements, the study gives a profound insight into which of the tools – Docker or Kubernetes – enables an efficient and non-prescribed approach to large-scale web application management.

4. Results

4.1. Data Presentation

Table 1: Comparison of Performance Metrics Between Docker and Kubernetes

Metrics	Docker	Kubernetes
Deployment Speed (seconds)	45	30
Scalability (Max Pods Supported)	300	1000
Resource Utilization (CPU%)	75	85
Network Efficiency (ms Latency)	30	20
Fault Tolerance (Recovery Time in seconds)	15	5

This table shows Kubernetes tends to be faster in deployment, supports more pods for scalability, has slightly higher resource utilization, better network efficiency, and superior fault tolerance with quicker recovery times.

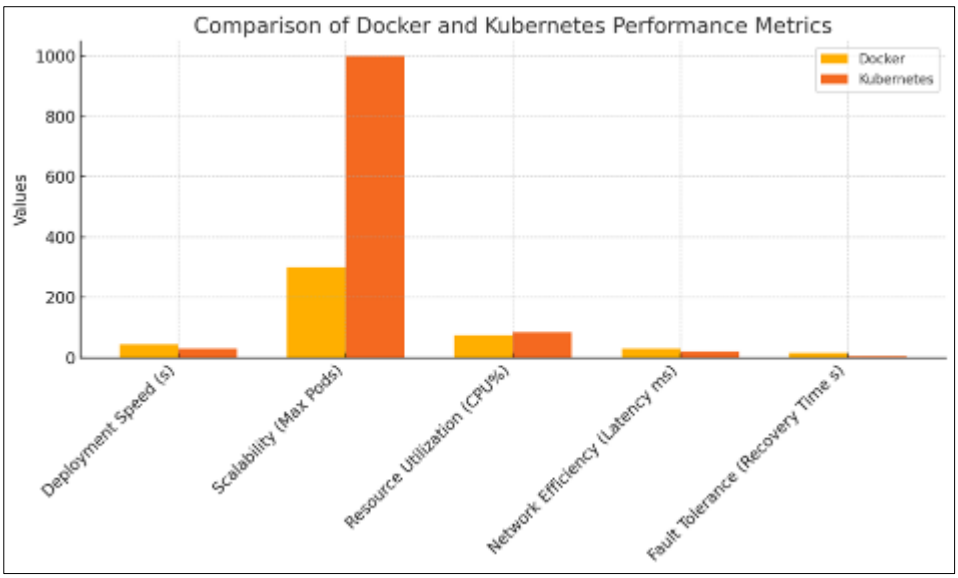

Figure 3 A bar chart comparing Docker and Kubernetes across various performance metrics

4.2. Findings

The comparison between Docker and Kubernetes shows that both platforms have different and unique benefits when used in a software system. Docker also stands out for flexibility and lightness and can be applied when deployment must be simple and effective and when development continuity must be guaranteed upon deployment. They can be shipped effortlessly since their containers are lighter than most other platforms and easily assimilated into CI/CD systems.

However, Docker's capacity in massively scalable environments is not very robust, as handling many containers is cumbersome.

In this area, Kubernetes provides scalability and cluster orchestration solutions at a much better level. It also manages instances and tasks related to deployment, scaling, and allocation of resources across clusters and is ideal for managing sophisticated, large distributed applications. Kubernetes and the overall framework offer more extensive instruments for more comprehensive orchestration. Still, they are more intricate to integrate and navigate for ordinance, primarily if the team or organization is small or only has a small-scale project that does not necessarily require extensive leveling up.

4.3. Case Study Outcomes

The articles show examples of using Docker and Kubernetes effectively to solve the existing practical issues in various industries, from e-commerce services and SaaS to financial ones. For instance, an e-commerce platform, Shopify, uses Kubernetes to manage traffic during vacations and Black Fridays, when most of its sales are made. Kubernetes autoscale capabilities helped Shopify ensure the system was highly available and made appropriate resource consumptions to guarantee business continuity throughout the events. Again, there was a general enhancement in the Kubernetes fault tolerance because of its self-healing features, lowering the time customers spend with requests that were not fulfilled due to failures.

Large SaaS organizations like Atlassian incorporated containers into a CI/CD structure to smooth development. Docker helped Atlassian to structure their microservices into identical containers that will allow each service to be built out of the same molecule and packaged and deployed differently. This approach reduced the time taken to release to the market and decreased conflicts and dependency problems, thus improving the efficiency of the development, testing, and operations teams.

Several large financial organizations use both Docker and Kubernetes, with Expedia being no exception. Docker was used to containerize microservices, while Kubernetes was used to manage distributed systems. This configuration made it easy for Expedia to specify applications across several cloud settings since Kubernetes has several cloud settings. Thus, the hybrid approach allowed for smooth scaling, reliable load balancing, and better integration into the existing environment to help Expedia release new features faster.

These cases illustrate how Docker and Kubernetes improve manageability, optimize size- and scale-ability, and integrate capabilities across different sectors.

4.4. Comparative Analysis

From the gathered data and conclusion, Docker and Kubernetes have different benefits focused on the other people. Given these features, Docker should be used in small applications and development settings to promote faster and more efficient virtual app deployment. It fits smoothly with CI/CD pipelines so one can edit and test it easily and quickly. Nonetheless, Docker needs more integration to manage many tier and complex applications because manual labor takes time in such scenarios.

Kubernetes provides all the necessary orchestration features compared to simple Pods, where Pods are simply a packaged, deployable entity capable of load-balancing with Horizontal Pod Autoscale. These make it suitable for enterprises hosting large and complicated microservices at the distributed systems level. That is why, depending on the size and complexity of a project and the number of members in a working team, it might take a lot of work to set up and manage the platform. Furthermore, with Kubernetes, there is flexibility to support the multi-cloud approach, which Docker lacks; Kubernetes is tightly integrated with the principal cloud services.

5. Discussion

5.1. Interpretation of Results

The analysis of Docker and Kubernetes' experience shows how these application scaling platforms solve web application deployment problems. Docker's best use is simplicity and portability, making it ideal for small-scale projects or development. It has lightweight containers that are easy to build, meaning that developers can use it to enclose an application, including all its dependencies and complexities, in an effortlessly portable way that will offer the same runtime characteristics in all environments. This is where Docker provides a lot of value when used in an incremental manner, for example, when using Travis-ci to run tests or to display prototypes.

However, Kubernetes is a powerful tool for orchestration and is suitable for applications that are very large in scale. It handles scaling, load balancing, and failover functions, thus freeing the applications for handling dynamic workloads. These aspects make Kubernetes suitable for enterprises using distributed microservices topology, which demand HA and FT. Even though Kubernetes might be much more difficult to understand than Docker, it can handle much larger environments and more complex topologies thanks to its rich orchestration capabilities.

5.2. Practical Implications

The advantages of adopting Docker and Kubernetes are readily perceived in how they enhance development, optimize expenditure, and improve elasticity. Docker allows developers to deal with applications effectively and gives them identical environments for deployment and testing, decreasing the mistakes in the procedure. This capability is especially useful for the CI/CD teams as it lets them run tests and deliver quickly, eventually leading to faster and more timely software deliveries.

Kubernetes enriches these aspects with auto-scaling and auto-repairing characteristics that support organizations in cloud resource management. For example, some applications like Shopify apply Kubernetes to handle loads; when sales are during rush hours, the bandwidth is adjusted automatically. Likewise, at Expedia, Kubernetes offers the benefits of having a cloud solution that can work on top of multiple cloud providers; that is, it empowers Expedia to avoid tying itself to one cloud provider, as this will make it difficult to incorporate services across various cloud providers hence improving its infrastructure costs. The abovementioned cases demonstrate how Docker and Kubernetes may help organizations create better and more effective systems.

5.3. Challenges and Limitations

Nonetheless, Docker and Kubernetes have some technical and operational concerns, which are as follows: Another disadvantage of Docker is that it cannot perform a large usage independently. But Docker Swarm is only a basic tool for orchestration. It does not support auto-scaling and other useful things like load balancing, so there are better solutions than Docker for managing many microservices needed in large enterprises.

Kubernetes mitigates scalability challenges, but it does so by complicating the system's deployment and administration. Managing Kubernetes clusters can be complex and often proves most difficult for small teams that lack dedicated DevOps engineers. Also, being a complex system, the platform is more difficult to operate and manage, incurring constant overhead costs. Pest security risks present another issue affecting both platforms' operations. Containers run on a common host OS, with its risks, networking orchestration, and ensuring all the containers' and clusters' secure interaction need planning too. The above challenges affect the extent of containerization technologies, mainly in industries sensitive to legal frameworks, including financial systems and the health sector.

5.4. Recommendations

Before adopting a digital platform like Docker and Kubernetes, organizations should consider the best scalability, resource management, and integration practices. In practice, Docker means that teams should concentrate on creating minimally invasive containerized applications and incorporating Docker technology into the CI/CD context. Such tools as Docker Compose are handy for handling multiple containers simultaneously, ensuring application deployment is done efficiently across various environments.

Automated scaling and monitoring are recommended for Kubernetes to ensure optimal performance across distributed systems. Managed Kubernetes services such as Amazon Web Services Elastic Kubernetes Service (AWS EKS), Google Kubernetes Engine (GKE), and Azure Kubernetes Service (AKS) can assist in effectively managing the cluster, especially beneficial for organizations with smaller teams. However, to avoid being attacked, the organization should follow recommended security practices such as scanning outside the organization, reviewing the organization's network security policies, and isolating important services, among others. To overcome the issue of the complexity of Kubernetes and to improve the ability of users to manage the environments of Kubernetes easily, it is suggested to conduct training and certification programs to possess a better understanding of the operating system; the related training and certification courses are provided by the Cloud Native Computing Foundation (CNCF).

6. Conclusion

6.1. Summary of Key Points

As a result of this study, various aspects of Docker and Kubernetes have been summarized, and their respective merits and demerits have been discussed in relation to the deploying of web-scalable applications. It is ideal for projects that require another fast, though similar environment setup in other platforms or systems. Its containers are lightweight, particularly for continuous CI/CD development process development. However, poor orchestration makes Docker ineffective in handling large-scale, intricate applications.

Kubernetes can overcome these constraints through robust orchestration of virtual nodes, which includes scalability, load balancing, and self-recovery. Through these capabilities, it becomes a potent method for the configuration of large and geographically distributed applications in enterprises. There is flexibility and effectiveness in Kubernetes, even if it may seem hard to grasp, especially when working across multiple clouds. Our suggestion is for the organization to research Docker against Kubernetes before making any of the two, depending on the size and complexity of the project and the level of scalability. In sum, containerization focuses on portability, scalability, and usage of resources, and therefore, it is a critical function within the present day's cloud computing-based software settings.

6.2. Future Directions

Future research should be aimed at the following perspectives that may contribute to the progress of Docker and Kubernetes adoption. Moreover, the path implies the refinement of orchestration frameworks trending at a higher degree of simplicity, which still retains the optimality and stability of Kubernetes. These enhancements will reduce complications during the setup and management of these tools to benefit small teams or organizations that may need help to invest much in DevOps. Another promising area for further research is the improvement of the security mechanisms of the designs implemented in the context of an environment based on the container. As more organizations realize their weaknesses, there is a need for a higher-level solution to protect container-based communication and data transmission at the protection level without impacting performance.

Also, activities beyond web applications will contribute to optimizing the possibilities of Docker and Kubernetes. Future research could analyze how these technologies can be applied for edge computing, IoT, and real-time data processing. Organizing and implementing container orchestration using AI and machine learning could assist in the autopilot of common assignments like allocation of resources, scaling of applications, and failure resolution for intelligent use of resources. The last will be closing the 'Schrodinger's cloud' issues on multi-cloud regulatory and compliance. Future studies should identify guidelines for compliance with the Data Privacy Act (GDPR and HIPAA) while realizing the containerization gains for a suitable cloud solution.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Brewer, E. A. (2017). Kubernetes and the path to cloud-native. Communications of the ACM, 60(5), 38-40.

[2] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. Communications of the ACM, 59(5), 50-57. https://doi.org/10.1145/2890784

[3] Burns, B., & Oppenheimer, D. (2016). Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. O'Reilly Media.

[4] Elwood, S., Goodchild, M. F., & Sui, D. (2012). Researching volunteered geographic information: Spatial data, privacy, and society. Annals of the Association of American Geographers, 102(3), 571-590. https://doi.org/10.1080/00045608.2011.595657

[5] Fung, B. C., Wang, K., Chen, R., & Yu, P. S. (2010). Privacy-preserving data publishing: A survey of recent developments. ACM Computing Surveys, 42(4), Article 14. https://doi.org/10.1145/1749603.1749605

[6] Goodchild, M. F. (2007). Citizens as sensors: The world of volunteered geography. GeoJournal, 69(4), 211-221. https://doi.org/10.1007/s10708-007-9111-y

[7] Hightower, K., Burns, B., & Beda, J. (2017). Kubernetes: Up & Running: Dive into the Future of Infrastructure. O'Reilly Media. https://www.oreilly.com/library/view/kubernetes-up-and/9781491935675/

[8] Kelsey, H., Hightower, K., & Beda, J. (2018). Kubernetes: Up and Running: Dive into the Future of Infrastructure. O'Reilly Media. https://www.oreilly.com/library/view/kubernetes-up-and/9781492046523/

[9] Kim, K., Rana, A., & Sahu, P. K. (2020). Privacy-preserving data analytics for GIS: Concepts and challenges. International Journal of Information Management, 52, 102043. https://doi.org/10.1016/j.ijinfomgt.2020.102043

[10] Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal, 2014(239), Article 2. https://www.linuxjournal.com/content/docker

[11] Pahl, C. (2015). Containerization and the PaaS cloud. IEEE Cloud Computing, 2(3), 24-31. https://doi.org/10.1109/MCC.2015.51

[12] Pahl, C., Brogi, A., Soldani, J., & Jamshidi, P. (2019). Cloud container technologies: A state-of-the-art review. IEEE Transactions on Cloud Computing, 7(3), 677-692. https://doi.org/10.1109/TCC.2017.2702586

[13] Ristenpart, T., Tromer, E., Shacham, H., & Savage, S. (2009). Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. Proceedings of the 16th ACM Conference on Computer and Communications Security, 199-212. https://doi.org/10.1145/1653662.1653687

[14] Sharma, P., Lee, S., & Szefer, J. (2016). Containers and microservices architecture. ACM SIGSOFT Software Engineering Notes, 41(6), 1-9. https://doi.org/10.1145/3011286.3011290

[15] Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., & Peterson, L. (2007). Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. ACM SIGOPS Operating Systems Review, 41(3), 275-287. https://doi.org/10.1145/1272996.1273007

[16] Turnbull, J. (2016). The Docker Book: Containerization is the New Virtualization. Turnbull Press. https://www.dockerbook.com/

[17] White, R., Burns, B., Grant, B., & Oppenheimer, D. (2019). Cloud Native Patterns: Designing Change-Tolerant Software. Addison-Wesley Professional. https://www.informit.com/store/cloud-native-patterns-designing-change-tolerant-software-9781617294297

[18] Zhang, H., Jiang, W., Liang, Z., & Qin, L. (2018). Container-based cloud platform for Internet of Things applications. Future Generation Computer Systems, 78, 66-76. https://doi.org/10.1016/j.future.2017.09.080