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

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

Research Article

Survey of Containerization, Orchestration, and CI/CD Integration on DevOps in Modern Software Development

Satyadhar Kumar Chintagunta*

Independent Researcher

Received 01 Dec 2023, Accepted 24 Dec 2023, Available online 26 Dec 2023, **Vol.13, No.6 (Nov/Dec 2023)**

Abstract

Containerization, orchestration, and CI/CD are the key tenets of the contemporary DevOps practices that allow organizations to be agile, scalable, and release software on a regular basis. Containerization packages up applications and dependency packages, and these are small and lightweight packages, which can be guaranteed to work in an environment that is both more heterogeneous and also in supporting microservices-based architecture. Apache Mesos, Kubernetes and Docker Swarm are orchestration systems, which simplify how to structure distributed applications in a multi-cloud and hybrid environment. The systems offer computerized multi-faceted operations such as deployment, scaling, networking and fault tolerance. The combination of these technologies and CI/CD pipelines automates the software development, testing, and deployment and makes the release processes faster and avoids the possibility of human error. Containerization, orchestration, and continuous integration/continuous delivery (CI/CD) ensure improved efficiency, quality, and resilience of cloud-native systems and enable operations and development teams to cooperate and make the delivery process seamless. It is a survey that generalizes what the different scholars have accomplished in the previous few years and opens up a multifaceted outlook of these technologies and the manner in which they help in the development of the modern software engineering practices, and this offers a glimpse of responsible and receptive DevOps ecosystems.

Keywords: *Containerization, Orchestration, CI/CD, DevOps, Cloud-native, Microservices, Automation, Software.*

1. Introduction

DevOps has greatly affected the dynamic software development industry as it has made the software delivery process quicker and more reliable because it bridges the gap between the development and the operations. Containerization, orchestration, and continuous integration/continuous delivery (CI/CD) are some of the critical elements incorporated in the modern DevOps ecosystem that assist organizations in attaining their objectives of scalability and agility [1]. The application development, deployment, and maintenance processes are being shaken by new technology such as containerization, orchestration, and continuous integration/continuous delivery (CI/CD) to make it more efficient, consistent, and flexible in this constantly changing world.

Containerization has brought about change in package applications that are the wrapping software along with its dependencies in lightweight, portable containers, which can be executed in the same manner in any environment.

*Corresponding author's ORCID ID: 0000-0000-0000-0000
DOI: https://doi.org/10.14741/ijcet/v.13.6.14

Containers and their concern are on resource efficiency enabling cloud providers to spin containers and make it a perfect fit fast Microservices based architecture [2]. The containerized method is applied to streamline deployment of applications to maximum levels; it minimizes the possibility of environment-to-environment conflicts, and can be horizontally scaled, which is exactly the essence of the DevOps practices.

Orchestration frameworks such as Kubernetes, Docker Swarm and Apache Mesos are essential in handling the complexity explosion that arises when handling a system of containers [3]. Orchestration frameworks: This one automates such processes as-deployment, scaling, networking, and fault tolerance. Resiliency and workload portability in multi-cloud or hybrid infrastructures can also be achieved through orchestration [4]. Multi-cloud or hybrid infrastructures have relevance to their resiliency and workload portability because applications reliable and scalable, even though the underlying systems are different. Orchestration has since been a part of DevOps pipeline. DevOps also relies heavily on CI/CD pipelines, which automate the processes of application development, testing, and deployment. By automating and continuously validating feedback and validation, CI/CD

pipelines not only decrease human error but also provide software with a consistent degree of quality [5][6]. CI/CD correctly applied in conjunction with containerization and orchestration can provide either software delivery that is both agile and stable. CI/CD and containerization IaC follows them, providing the automatization of infrastructure provisioning, and infrastructure configuration is directly part of development process.

The fact remains that there are also improvements, yet interoperability, security, and performance optimization becomes a prominent problem [7]. To address these problems, long-term research into the potential to apply scalable orchestration methods, devops practice standardization, intelligent automation of CI/CD pipeline is very important to overcome them. This survey has given a integrated image of the concept of containerization, orchestration and CI/CD integration in DevOps, their functions, interactions and how they processed in the future in developing the software of tomorrow.

Structure of the Paper

The outline of the paper is as follows: In the second II, cover the basics of devops, which are essential in today's software development. In Section III, go over devops containerization and orchestration. Cover devops is covered in Section IV. Section V - summarize vital results, problems, and suggested future research. Lastly, in Section VI, give some considerate recommendations.

Fundamentals of Devops in Modern Software Development

DevOps has emerged as a disruptive approach in contemporary software engineering as a result of the increasing complexity of the environment and the necessity for more dependable and quicker software production. Devops method presupposes that the developers and the operations specialists are able to work on the same team throughout the software development life cycle, promoting the teamwork ethos and shared ownership. DevOps are based on the concepts of automation, feedback loops, CI/CD. These ideas are supported with tools and technology that accelerates the code development process to production using automated testing, code deployment and monitoring of performance. Multiple trending technologies such as Ansible, Docker, Kubernetes, and Jenkins facilitate particular processes of automation and assist with dependency management [8]. The following features can be achieved in this DevOps framework:

The DevOps methodology is very compatible with the Agile and Lean methodologies.

Agile software development, from which DevOps sprang, emphasizes teamwork between stakeholders including clients, product managers, developers, and (sometimes) quality assurance in order to identify and address product gaps through fast iteration.

"DevOps" refers to the practice of applying Agile concepts to the whole service, not just "the code."

Practices and Principles of DevOps

Continuous Integration (CI) and Continuous Delivery (CD)

CI is the process of integrating and testing of new changes in the code, which is continually repeated in a shared repository. With the help of CI/CD practices, development teams able to automatize the process of building and executing tests on other branches every time any code is pushed to a common repository. This practice greatly reduces integration errors and improves the software delivery rate. CI/CD also automates manual processes, which allows organizations to release new releases with the fewest defects and much shorter turnaround times.

Infrastructure as Code (IaC)

IaaS simplifies the management and provisioning of infrastructure because it removes the manual configuration and configuration files take its place. This will provide scalability and consistency in testing, production, and development environments which helps to reduce the possibility of human error [9]. IaaS and Devops promote collaboration through the exchange of infrastructure specification. By suggest that IaC practices contribute to consistency in the cloud, particularly large-scale DevOps adoption.

Automated Testing

Automated testing is all about the most successful DevOps. It suggests code that has been carefully reviewed in all levels of the development life cycle to make sure that bugs are not allowed to get to production phase. It saves time spent on the manual testing as well as it guarantees the quality of the product during the SDLC process.

Key benefits of DevOps

These are few of the key benefits of DevOps. There, that being said, it can proceed to the details:

Improved Collaboration and Communication

DevOps leads to formation of decent working relationship and contact and this alters the organizational dynamics. It also makes the work easier since it breaks the silos existing between the operations and the development department and other departments [10]. This synergy also ensures transparency and cooperation between cross-

functional teams and transparency and instant communication.

Accelerated Development Cycle and faster time to market

The broadest definition of DevOps is the capacity to shorten the development cycle. Due to the automation of the development process, promotion of continuous integration, and deployment, it also uses less time than the development, testing, and deployment technique. Along with improving efficiency, releasing items at a faster rate ensures that consumers' requests are met sooner, giving businesses a competitive advantage.

Enhanced Quality and Reliability of Software

CI/CD is one of the most important elements of the DevOps, with software excellence, but not speed as its priority. Automated and continuous testing, as well as the process of continual improvements, are also part of its rigorous quality control procedure. Client loyalty and trust are enhanced, software stability is increased, and the user experience is enhanced as a result.

Automations

The main goal of DevOps is to eliminate manual, repetitive processes. As a result of automations, teams have more time and energy to devote to initiatives that need strategic thinking and innovation. Among other benefits, this DevOps feature allows for the simplification of code deployment, configuration management, and infrastructure provisioning processes, leading to increased efficiency, decreased error rates, and more consistent and dependable outcomes.

Challenges in Adopting DevOps

Although it comes with its benefits, some challenges to its adoption in modern software development are experienced, which the organization must solve to achieve its full potential [11].

Organizational Resistance and Cultural Barrier

DevOps is incompatible with conventional, compartmentalized organizational structures because of its emphasis on teamwork between developers and operations professionals. Resistance occurs when teams are not willing to alter established workflows, lose control or they do not trust shared responsibilities.

Skills Gap and Training Requirements

DevOps implementation needs the knowledge of automation, containerization, cloud computing, CI/CD systems, and security measures. There are numerous companies that cannot find the professionals who have such a range of skills. Teams may not effectively adopt modern tools and practices without appropriate training and the continuous learning programs.

Toolchain Complexity and Integration Issue

The DevOps ecosystem offers a wide variety of tools for automated development, testing, deployment, and monitoring, as well as version control. There is also a still existing problem of the incorrect selection of tools depending on the requirements of the organization.

Security and Compliance Concern

DevOps accelerates software delivery and thus becomes difficult to ensure security and compliance to regulations. The systems may be exposed to dangers because of container images, insecure settings, and absence of good access controls. In addition, the adherence to the industry standards (including GDPR, HIPAA, or PCI-DSS) implies that security checks are to be integrated into CI/CD pipelines, commonly known as DevSecOps, which further complicates the matter.

Containerization and Orchestration in Devops

Containerization is one of the game changer technologies in the DevOps world since it delivers software in an efficient and fast manner. Simply put, containerization involves securing an application, including all its dependencies, libraries, and settings, and wrapping them into one lightweight and portable package known as a container [12]. This ensures that there is a consistent running of applications operating in various environments in one of the most widespread problems in Devops that is environment inconsistency between development and testing and production. When integrated into a DevOps pipeline, containers work well with CI/CD, which stands for CI/CD. Published in 2013, Docker was the earliest tool of containerization to become popular, and the list of containerization tools has since multiplied at a steady rate, with options like Pod man, LXC, and CRI-O.

Evolution of Containerization

Containerization A process of building and operating software in which applications and their dependencies are bundle together as self-bounded systems called containers [9] These containers are a lightweight, isolated executable environment which guarantees uniformity and system-to-system mobility. The application code, runtime environment, libraries, and dependencies are safely contained within containers, allowing for their execution on any containerized infrastructure. This approach has a number of strengths, such as:

Containers can operate on a great variety of operating systems and infrastructure platform such as Linux, Windows, and cloud environments. They contain all the necessary elements which give an opportunity

to easily relocate applications between the development environment, testing, and production environment.

Containers are efficient because they share the kernel with the host computer, which reduces resource consumption. They boot quicker, and the footprint is less compared with other traditional virtual machines, meaning that they can be easily used to harness infrastructure resources.

The programs and their dependencies are separated by containers, and no changes in a container can affect the change in the other containers. This isolation enhances security, stability and fault tolerance, as it reduces the conflict and the extent of effect of failure.

Scalability: Containers can be easily horizontally scaled by executing numerous copies of a similar containerized program. Container orchestration technologies are capable of automating through which the scaling can be accomplished allowing the applications to effectively manage the increased traffic and demand.

Containerization encourages the use of agile development because it allows developers to work within a consistent and repeatable environment. Containers shorten development cycles, simplify deployment, and alleviate the process of setting up development environments.

Security and Resource Management Challenges in Containerization

Container security management is an important issue because vulnerabilities can affect usability, performance and service delivery. The issues were cross cutting across different phases of container development, deployment and execution.

Container Image Phase: Suspicious or malicious container images may grant access to malicious code into the system and thus become security threats. In addition, failure to perform or do inadequate vulnerability scans exposes containers to the threat of stealing data in the form of stolen credentials and exploitable security vulnerabilities.

Container Host Phase: Container security is compromised because of insecure host settings and lack of proper resource isolation. Host fatigue, data breaches, buffer overflows, and attacker unauthorized access can all result from hosts that provide escalated access rights.

Intra-Container Phase: The common weaknesses at this level are weak isolation between containers and poorly configured environments. Such vulnerabilities may enable the attackers to access the systems without authorization, which result in DoS attacks or total system breakdowns. Improper scaling further increases the risk by exhausting system resources

Networking and Orchestration Phase: Insecure communication between containers or with external systems is a significant challenge. Misconfigured orchestration settings can create privileged networks, granting attackers control over nodes, while poor segregation of networks can expose sensitive data to interception.

Runtime Phase: Running development and production environments together increases the attack surface during runtime. Misconfigurations at this stage can lead to instability, degraded performance, and service delivery issues, making the container environment more vulnerable to exploitation.

Key Container Technologies (Docker, Podman, LXC)

There are some key contrainer technologies like Docker, Podman and LXC are explains below[13]:

Docker

Popularity: The Docker platform is among the most popular options for containerization.

Ecosystem and Community: Features an extensive community and ecosystem that offers a plethora of resources, including tools and support.

Portability: Docker containers are portable and may operate on any hardware or operating system.

Docker Hub: Anyone may utilize Docker Hub, a public registry, to exchange container images.

Tooling: Docker Swarm, Docker Compose, and other tools are all part of the package.

Learning Curve: Perfect for newcomers and smaller projects because to its low learning curve and ease of usage.

Podman

Daemonless: Podman differs significantly from Docker in that it does not use a daemon to operate containers.

Rootless: Enhances security by enabling the running of containers without root capabilities.

Compatibility: Docker commands can be substituted with Podman commands, as Podman is CLI compatible with Docker.

Scaling and Orchestration: Lacks inherent clustering capabilities; however, it can integrate with Kubernetes.

LXC/LXD

Linux Containers (LXC): Lightweight VMs (LXCs) are more similar to OS virtualization at the system level.

LXD: The LXC container manager that prioritizes ease of usage.

OS Level Virtualization: Provides a complete Linux environment, complete with filesystem, networking, and commands.

Performance: Virtual machines often have lower overhead than containers created using Docker.

1. Bioenergy refers to electricity and gas that is generated from organic matter,
2. known as biomass. This can be anything from plant and timber to agriculture and food
3. waste and even sewage. Bioenergy includes the production of fuel from organic matter as
4. well. Energy from biomass can be used for electricity, heating, and transportation, and

5. can be replenished anywhere. Around seventy-
6. five percent of the world's renewable
7. energy is composed of biomass energy due to its potential and wide use [7]. Also, it is
8. carbon-neutral, meaning that it adds no net carbon dioxide to the atmosphere. In addition,
9. it reduces the level of trash in the ground by as much as 90 percent by burning solid
10. waste. Biomass fuels, on the other hand, are not completely clean and can also cause
11. deforestation. They are also less efficient than fossil fuels. But proper management and
12. planning of its disadvantages will improve its potential.
13. Bioenergy refers to electricity and gas that is generated from organic matter,
14. known as biomass. This can be anything from plant and timber to agriculture and food
15. waste and even sewage. Bioenergy includes the production of fuel from organic matter as
16. well. Energy from biomass can be used for electricity, heating, and transportation, and
17. can be replenished anywhere. Around seventy-
18. five percent of the world's renewable
19. energy is composed of biomass energy due to its potential and wide use [7]. Also, it is
20. carbon-neutral, meaning that it adds no net carbon dioxide to the atmosphere. In addition,
21. it reduces the level of trash in the ground by as much as 90 percent by burning solid
22. waste. Biomass fuels, on the other hand, are not completely clean and can also cause
23. deforestation. They are also less efficient than fossil fuels. But proper management and
24. planning of its disadvantages will improve its potential.

Orchestration in DevOps

Orchestration in DevOps is the practice of managing the entire containerized application life cycle of a distributed system. Orchestration automates the coordination, deployment, management, scaling, etc.[14]. of a number of containers as unit of microservices so that reliability, availability, and performance is abstracted in modern applications. As microservices architectures and cloud-native systems develop, orchestration is a key mechanism for tying CI/CD pipelines together in a DevOps practice. Out of other orchestrators, Kubernetes is the most dominated platform for orchestration in DevOps, allowing capabilities like self-healing, load balancing, rolling updates, and auto-scaling. Other frameworks exist like Docker Swarm, Apache Mesos, and Hashi Corp Nomad, offering their own versions of orchestration that are dependent on simplicity, flexibility, or resource-efficiencies. These tools enable complex workloads to be managed by DevOps teams according to deployment targets spanning hybrid and multi-cloud configurations.

Need for Container Orchestration

Despite cloud service providers managing a growing variety of activities, whether complex, heterogeneous workloads can be orchestrated in large-scale cloud computing systems is still unknown [15]. Heuristic policies implemented by container orchestrators on older cloud systems fail to account for the wide range of workload scenarios or quality of service requirements. These items have the following major drawbacks:

The majority of these rules are offline-configured static heuristic techniques that are scale-limited and tailored to certain workload circumstances. As an example, threshold-based autoscaling techniques are limited to handling a specific set of workloads that have been previously determined. Extremely dynamic workloads, where applications require on-the-fly scaling in response to unique patterns of behaviour, are outside the scope of these principles.

Heuristic methods may not work as well as they formerly did when the system expands. Problems in allocating resources and scheduling jobs are common applications of bin-packing algorithms like best fit and least fit. However, such approaches may not work well in large-scale computing clusters due to high task scheduling delays.

Applications that are co-located sometimes overlook issues like resource congestion and performance interference. Applications running in close proximity to one another increase the risk of resource contention, which in turn increases the likelihood of performance degradation, increased maintenance expenses, and service level agreement (SLA) breaches.

Kubernetes: Features and Architecture

Open-source container orchestration project K8s makes it easier to deploy, manage, and autoscale container-based application operations and services. A K8s cluster has both master and worker nodes, as seen in Figure 1. The smallest unit in K8s that contains one or more containers is called a pod, and applications operate in pods [16]. The master node is responsible for managing the K8s cluster and should be troubleshooting capable. There are four main components that are required: a kube-controller manager, a kube-apiserver, a kube-scheduler, and etcd. Each part of the master node has a specific role, which are detailed below:

etcd: A K8s cluster relies on this for its data storage needs. With the exception of application data, it records every cluster event.

Kube-scheduler: This is in charge of determining which node is optimal for launching a new pod. Pod resource requests, node resource availability, and affinity and anti-affinity policies are some of the factors considered while choosing an appropriate node.

Kube-apiserver: The kubectl command-line tool allows an administrator to interface with a K8s cluster using kube-apiserver. Additionally, kube-apiserver connects to worker nodes so that the kubelet and kube-proxy tools may manage pod operations.
kube-controller-manager: This loop is essential for the cluster's monitoring. In order to keep the cluster in a near-ideal state, kube-controller-manager keeps an eye on its present status and makes adjustments to its resources as needed.

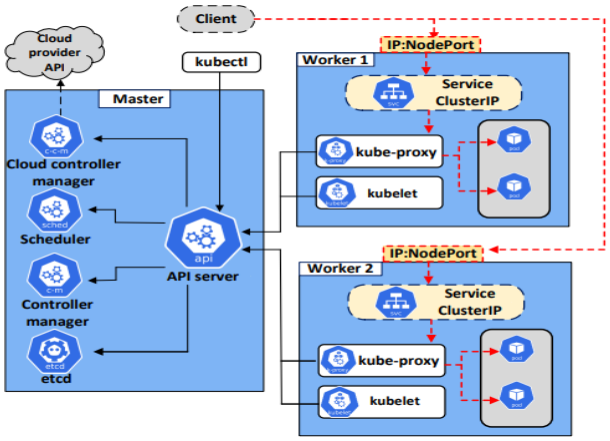

Fig.1 Kubernetes Architecture

Orchestration for Microservices and Cloud-Native Workloads

Multi-cloud and hybrid cloud architectures have taken center stage in modern DevOps practices, enabling enterprises more flexibility and velocity along with better cost efficiency and resilience. Hybrid cloud architectures integrate public and private cloud infrastructure, while multi-cloud techniques leverage several cloud providers' services [17]. Orchestration is an integral part of managing complex and distributed systems by providing automation and abstraction and unified control. Kubernetes, Docker Swarm, and Apache Mesos are orchestration tools that simplify the process of deploying, scaling, and monitoring distributed applications across different infrastructures. By utilizing orchestration, the service complexity unique to each cloud provider can be abstracted, thus providing a level of assurance that workloads are portable and interoperable across multiple environments. In hybrid architectures, orchestration may also aid in establishing a secure link between on-premises and cloud systems, which is crucial for achieving and maintaining application performance goals, meeting regulatory requirements, and remaining compliant.

CI/CD Integration in DEVOPS

The Agile Manifesto [18] lays out the fundamental principles of agile methodology, which include: putting people and their interactions first rather than processes and tools, putting working software first rather than detailed documentation, putting customer collaboration first rather than contract negotiation, and putting adapting to change ahead of planning ahead. "Satisfy the customer through early and continuous delivery of valuable software" is delivered in accordance with the principles of agile development. When CI and CD, which enhanced CI's features, were combined to create the CICD pipeline, agile processes became more efficient and productive.

Impact of CI/CD on Software Development Lifecycle

CI/CD has far-reaching effects on software development since it influences several stages of the process and enhances software quality. These are some of the key impacts to consider [19]:
Improved code quality: CI/CD incorporates testing to a large extent. Any CI/CD system must be reliable in the testing process. To fulfill this goal, CI/CD demands that the testing process must possess the ability to identify the bugs present in the code as early as possible up to the point when it goes to the production phase. Such strict practice provides quality and reliability of codes as opposed to the traditional methods.
Early Issue Identification: Continuous integration is very beneficial in detecting the problem at its early stages. It can do this by testing regularly and combining all the code changes with one another and this makes it easier to trace when and where an error had been made.
Faster deployment cycles: CI/CD shortens development cycles and code release times by automating several steps in the development process, including code creation, testing, and implementation.
Increased Productivity: Developers may spend less time on tedious manual tasks and more time actually writing code thanks to automation, which doesn't require any human interaction.
Better feedback loops: Feedback and dependencies on new features are received faster with changes being automatically promoted to various environments. That allows the developers to resolve such issues as soon as possible.

Current Trends in CI/CD Tooling

CI and CD technology has undergone a dramatic shift to meet the demands of more dependable and speedy software delivery [20]. Since the implementation of the DevOps practices in the organizations is a process, that is still ongoing, the following key tendencies in the sphere of CI/CD tooling have been introduced, such as the contemporary issues of development and the improvement of the automation rates. These tendencies are discussed in the section, and the academic materials published prior to May 2022 favor them.
Shift to Cloud-Native CI/CD: One of the current trends in the CI/CD tooling is the shift to cloud-native. Cloud-

native CI / CD products are designed in such a way that they leverage the features that cloud infrastructure offers in the context of scalability, flexibility and resilience. Tools such as Github actions, GitLab CI and CircleCI are provided as part of a series of cloud based services that eliminate the prospect of having the build servers installed in premises and allow organizations to expand their pipelines with ease.

Emphasis on Security and Compliance: The frequency and sophistication of the cyber-attacks on the CI/CD pipelines have been growing, and this has rendered security to be a vital issue. Advanced CI/CD technologies are incorporating advanced security functions to ensure that the software delivery process is safe. The CI/CD processes currently feature such functionalities as automatic security checks and vulnerability checks.

Integration with DevSecOps Practices: DevSecOps is the rise of security feature in the DevOps process. CI/CD tools are also changing to include DevSecOps through the integration of security checks into all parts of the pipeline. Reduced risk factors and improved software quality are the results of integration's early detection and fixing of security flaws.

Rise of Low-Code and No-Code CI/CD Solutions: Code-light and code-free in recent years, there has been a proliferation of CI/CD solutions aimed at making the process more accessible and democratizing it. These tools offer graphical interface and easy to use tools that enable users to create and orchestrate CI/CD pipelines without the need to write large amounts of code.

CI/CD Toolchains

There are some tools of CI/CD are as follows:

Jenkins

Jenkins is an excellent product for continuous integration and delivery that is open-source and built on the Java platform. Jenkins is ideal for individuals just starting out in software development because to its intuitive interface and easy installation procedure [21]. Jenkins eliminates a lot of tedious configuration work by making a number of plugins available for quick and easy addition.

GitLab CI

The Git framework is the foundation upon which the open-source web-based VMS Gitlab is created. In addition, it offers capabilities like continuous integration and delivery pipelines. There is no way to install any more plugins because all of the configuration takes place on the pipeline itself.

GitHub

One method for the continuous deployment of cloud-native apps is the GitOps methodology, which is also called Github Operations. Instead of the more common push method, it relies on a pull method of delivery [22]. The core principle of GitOps is taking advantage of the Git repository, which stores all the details on the production environment's infrastructure.

Literature of Review

The literature reviewed offers insights into containerization, orchestration, and CI/CD integration within DevOps, emphasizing evolving frameworks, performance and security challenges, while outlining future directions to enhance scalability, automation, and effective adoption across diverse development environments.

Zhou et al. (2022) A cloud container is more portable and has a lower volume. Containers would also gain from orchestrators that make large-scale container deployment and maintenance easier. The majority of cloud systems, in contrast to HPC systems, really use advanced container orchestration techniques. However, in the most recent developments, there have been several suggestions for ways to make container orchestration possible on HPC systems. Containerization and associated orchestration solutions on HPC systems are surveyed and taxonomized in this article. Cloud computing and high-performance computing are contrasted. Finally, look ahead to prospective research and engineering opportunities while discussing the obstacles [23].

Garg et al. (2021) Software development and business process acceleration may be achieved through the usage of CI/CD in tandem with DevOps. Applying CI/CD pipelines to an application that uses MLOps components, however, presents a number of important issues. To overcome these challenges, technology pioneers in the field often turn to cloud providers for specialized solutions. Greater insight into the processes of MLOps and DevOps, not to mention the machine learning lifecycle. Identify techniques and tools to operate the CI/CD pipeline of the ML frameworks under MLOps [22].

Brasoveanu et al. (2020) the performance appraisal models, the frequency of the appraisals, the tools, and the degree of details of the received performance information. Based on the findings of the study, the complexity of the performance engineering procedures and technologies underlies the lack of performance analysis application in DevOps, as many participants representing different sectors were included. According to the research, the performance analysis tools should be easily integrated into the DevOps pipeline with a little learning curve [24].

Truyen et al. (2019) the model-based methods used, the frequency of carrying out performance assessment, instruments, and the degree of specificity in the information of performance obtained. In accordance with the findings of the study, which was summarized with a significant sample of the subjects across a vast amount of industries, the complexity of

performance engineering techniques and technologies is guiding the idea that performance analysis is not introduced with large-scale implementations in DevOps. Studies established that it is eminent that the tools of performance analysis can be easily incorporated into the DevOps pipeline, and they have low learning curve [25].

Al Jawarneh et al. (2019) Modern container-based systems need sophisticated orchestration skills due to the large number of microservices services they may contain, each of which may have intricate relationships. The scalability and ease of deployment of microservices, however, were greatly facilitated by container-based technologies. Docker Swarm, Kubernetes, Apache Mesos, and Cattle are just a few of the new container orchestration engines that have emerged, but there isn't yet a comprehensive review of their features and performance that IT managers can use to pick the best one [26].

Buyya et al. (2018) a system for making cloud container orchestration as efficient as possible. Elastic virtual clusters may be built with the help of autoscaling algorithms and policies for scheduling and rescheduling resources. According to the suggested architecture, client applications such as web services, offline analytics activities, and backend pre-processing operations can share a computing environment within containers. Resource management algorithms and policies may be designed to better use the available virtual resources, meeting the resource needs of diverse applications while minimizing the provider's operational expenses [27].

Table I is a summary of important papers on containerization, orchestration, and CI/CD integration in DevOps, which summarize their methods, results, issues, and future prospects, indicating the dynamism of changes and the lack of research.

Table 1 Summary of a Study on Containerization, Orchestration, and CI/CD Integration DevOps in Modern Software Development

Author	Study On	Approach	Key Findings	Challenges	Future Directions
Zhou et al., (2022)	Containerization & Orchestration in Cloud vs. HPC	Survey & taxonomy of containerization and orchestration strategies in HPC and Cloud	Containers are lightweight and portable; orchestration enhances scalability; clear differences between Cloud and HPC practices	Limited orchestration support in HPC environments; integration complexity	Develop efficient orchestration models for HPC; bridge gap between Cloud and HPC container use
Garg et al., (2021)	CI/CD with DevOps and MLOps	Analytical study of CI/CD integration in ML lifecycle	CI/CD accelerates DevOps workflows; MLOps adds unique challenges requiring specialized tools	Difficulties in managing ML pipelines with CI/CD; dependency on cloud providers	Improve tools for seamless CI/CD in MLOps; enhance automation and standardization
Brasoveanu et al., (2020)	Performance Engineering in DevOps	Industry survey on performance evaluation practices	Performance analysis is valuable but rarely adopted due to complexity; need for lightweight tools	High learning curve of tools; difficulty in integration with pipelines	Simplify performance analysis tools; integrate user-friendly solutions into DevOps
Truyen et al., (2019)	Container Orchestration Frameworks (Docker Swarm, Kubernetes, Mesos)	Comparative feature study of orchestration frameworks	Frameworks continuously evolve; each offers unique strengths for distributed systems	Rapid evolution complicates adoption; lack of clear guidance for practitioners	Provide standardized benchmarks; long-term evaluations to guide framework selection
Al Jawarneh et al., (2019)	Functional & Performance Assessment of Orchestration Engines	Comparative study of orchestration tools (Swarm, K8s, Mesos, Cattle)	Containers support scalable microservices; orchestration engines differ in performance	Managing large-scale microservices with interdependencies is complex	Develop advanced orchestration strategies; improve tool selection guidance
Buyya et al., (2018)	Orchestration Framework for Cloud Containers	Proposed architecture for resource scheduling and autoscaling	Resource management improves utilization, reduces cost, and enhances elasticity	Balancing efficiency with QoS requirements; multi-tenant resource conflicts	Refine scheduling and autoscaling algorithms; expand support for diverse workload

Conclusion and Future Work

DevOps containers, orchestration and CI/CD have greatly transformed the modern software development by bringing about the agility, scalability and operational efficiency. Containerization is also transportable and predictable since it bundles applications with their dependencies as lightweight environments in which they can run in a graceful way across a range of platforms. The models that roll out, automate and control resources are the orchestration models, as a result of which microservice-based structures can be operated in complex multi-cloud and hybrid structures. CI/CD pipelines are also conducive to the work of DevOps as the processes of building, testing and deployments are mechanized thereby shortening the release process, enhancing the quality of the code and is made possible to enable continuous

feedback loops. The combination of these technologies will give an ecosystem, which enhances the communication and collaboration between the operations and development team resulting in more reliable and faster delivery of the software. Despite the broad scale of the effects of the problems, the areas that should be enhanced include interoperability problems, maturity of automation, and integration of security. Overall, the overlap between these practices indicates a major shift in the direction to cloud-native and automated practices as the basis of sustainable, resilient, and future-intensive software engineering.

The additional directions are expected to be on AI-based orchestration, predictive scaling and adaptive CI/CD pipelines to have more automation. DevSecOps integration, cross-cloud standardization, and simplified tooling are also factors that should be researched so as to enable scaled, secure and sustainable adoption of DevOps in an increasingly heterogeneous environment.

References

[1] A. Hemon, B. Lyonnet, F. Rowe, and B. Fitzgerald, "From Agile to DevOps: Smart Skills and Collaborations," *Inf. Syst. Front.*, vol. 22, no. 4, pp. 927–945, 2020, doi: 10.1007/s10796-019-09905-1.

[2] R. Pérez de Prado, S. García-Galán, J. E. Muñoz-Expósito, A. Marchewka, and N. Ruiz-Reyes, "Smart Containers Schedulers for Microservices Provision in Cloud-Fog-IoT Networks. Challenges and Opportunities," *Sensors*, vol. 20, no. 6, p. 1714, Mar. 2020, doi: 10.3390/s20061714.

[3] A. Köhler, "Evaluation of MLOps Tools for Kubernetes : A Rudimentary Comparison Between Open Source Kubeflow, Pachyderm and Polyaxon," *Diva-Portal.Org*, no. October, pp. 1–60, 2022.

[4] V. S. Thokala, "Utilizing Docker Containers for Reproducible Builds and Scalable Web Application Deployments," *Int. J. Curr. Eng. Technol.*, vol. 11, no. 6, pp. 661–668, 2021, doi: 10.14741/ijcet/v.11.6.10.

[5] F. Zampetti, S. Geremia, G. Bavota, and M. Di Penta, "CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, Sep. 2021, pp. 471–482. doi: 10.1109/ICSME52107.2021.00048.

[6] G. Modalavalasa, "The Role of DevOps in Streamlining Software Delivery: Key Practices for Seamless CI/CD," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 1, no. 12, pp. 258–267, Jan. 2021, doi: 10.48175/IJARSCT-8978C.

[7] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of DevOps concepts and challenges," *ACM Comput. Surv.*, vol. 52, no. 6, 2020, doi: 10.1145/3359981.

[8] Y. Demchenko *et al.*, "Teaching devops and cloud based software engineering in university curricula," in *Proceedings - IEEE 15th International Conference on eScience, eScience 2019*, 2019. doi: 10.1109/eScience.2019.00075.

[9] Sarishma and Abhishek, "A Systematic Review of the Impact of Containerization on Software Development and Deployment Practices," *Comput. J. Appl. Comput. Sci. Intell. Technol.*, vol. 1, no. 1, pp. 40–51, Jun. 2021, doi: 10.17492/computology.v1i1.2105.

[10] A. Martin, S. Raponi, T. Combe, and R. Di Pietro, "Docker ecosystem – Vulnerability Analysis," *Comput. Commun.*, vol. 122, pp. 30–43, 2018, doi: 10.1016/j.comcom.2018.03.011.

[11] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *IEEE Access*, vol. 7, no. c, pp. 52976–52996, 2019, doi: 10.1109/ACCESS.2019.2911732.

[12] E. Hitimana, G. Bajpai, R. Musabe, L. Sibomana, and K. Jayavel, "Containerized Architecture Performance Analysis for IoT Framework Based on Enhanced Fire Prevention Case Study: Rwanda," *Sensors*, vol. 22, no. 17, 2022, doi: 10.3390/s22176462.

[13] V. S. Thokala, "A Comparative Study of Data Integrity and Redundancy in Distributed Databases for Web Applications," *Int. J. Res. Anal. Rev.*, vol. 8, no. 4, pp. 383–389, 2021.

[14] Y. Wang, C. Lee, S. Ren, E. Kim, and S. Chung, "Enabling role-based orchestration for cloud applications," *Appl. Sci.*, vol. 11, no. 14, pp. 1–23, 2021, doi: 10.3390/app11146656.

[15] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, "Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions," *ACM Comput. Surv.*, vol. 54, no. 10, 2022, doi: 10.1145/3510415.

[16] Q. M. Nguyen, L. A. Phan, and T. Kim, "Load-Balancing of Kubernetes-Based Edge Computing Infrastructure Using Resource Adaptive Proxy," *Sensors*, 2022, doi: 10.3390/s22082869.

[17] S. Shekhar and P. Goel, "Comparative Analysis Of Optimizing Hybrid Cloud Environments Using AWS, Azure, And GCP," *Int. J. Creat. Res. Thoughts*, vol. 10, no. 8, pp. 2320–2882, 2022.

[18] S. A. I. B. S. Arachchi and I. Perera, "Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management," in *2018 Moratuwa Engineering Research Conference (MERCon)*, IEEE, May 2018, pp. 156–161. doi: 10.1109/MERCon.2018.8421965.

[19] K. M. Pitchikala, "Enhancing Software Development with CI/CD: Best Practices and Strategies," *J. Sci. Eng. Res.*, vol. 2022, no. 12, pp. 172–176, 2022.

[20] S. Jangampeta, J. P. Morgan, and S. T. Makani, "The Evolution of CICD Tools In Devops from Jenkins to Github Actions," *Int. J. Comput. Eng. Technol.*, vol. 13, no. 2, pp. 166–174, 2022.

[21] C. Singh, N. S. Gaba, M. Kaur, and B. Kaur, "Comparison of Different CI/CD Tools Integrated with Cloud Platform," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, IEEE, Jan. 2019, pp. 7–12. doi: 10.1109/CONFLUENCE.2019.8776985.

[22] S. Garg, P. Pundir, G. Rathee, P. K. Gupta, S. Garg, and S. Ahlawat, "On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps," in *Proceedings - 2021 IEEE 4th International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2021*, 2021. doi: 10.1109/AIKE52691.2021.00010.

[23] N. Zhou, H. Zhou, and D. Hoppe, "Containerization for High Performance Computing Systems: Survey and Prospects," *IEEE Trans. Softw. Eng.*, vol. 49, no. 4, pp. 2722–2740, 2022, doi: 10.1109/TSE.2022.3229221.

[24] A. Brasoveanu, M. Moodie, and R. Agrawal, "Textual evidence for the perfunctoriness of independent medical reviews," *CEUR Workshop Proc.*, vol. 2657, no. Ci, pp. 1–9, 2020, doi: 10.1145/nnnnnnn.nnnnnnn.

[25] E. Truyen, D. Van Landuyt, D. Preuveneers, B. Lagaisse, and W. Joosen, "A Comprehensive Feature Comparison Study of Open-Source Container Orchestration Frameworks," *Appl. Sci.*, vol. 9, no. 5, 2019, doi: 10.3390/app9050931.

[26] I. M. Al Jawarneh *et al.*, "Container Orchestration Engines: A Thorough Functional and Performance Comparison," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, IEEE, May 2019, pp. 1–6. doi: 10.1109/ICC.2019.8762053.

[27] R. Buyya, M. A. Rodriguez, A. N. Toosi, and J. Park, "Cost-Efficient Orchestration of Containers in Clouds: A Vision, Architectural Elements, and Future Directions," *J. Phys. Conf. Ser.*, vol. 1108, no. 1, 2018, doi: 10.1088/1742-6596/1108/1/012001.