Patrycja Zajączkowska
ORCID: 0009-0001-9275-6467

University of Siedlce
Faculty of Exact and Natural Sciences
Institute of Computer Science
ul. 3 Maja 54, 08-110 Siedlce, Poland

zajaczkowska.patrycja@outlook.com

Modern Application Lifecycle Management Approaches

DOI: 10.34739/si.2025.32.03

Abstract. Effective application lifecycle management (ALM) relies on modern tools and practices that streamline development while ensuring security and stability. This article examines key components of a modern ALM strategy, including DevOps culture, CI/CD pipelines, Infrastructure as Code, containerization, orchestration, monitoring, observability, and the growing role of AI. Special attention is given to the integration of security into the CI/CD process, particularly through automated testing techniques such as Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST). These practices help identify vulnerabilities early in the development cycle, reducing risk and enhancing application reliability. Adopting these technologies involves initial investments in licenses, training, and team restructuring. However, these efforts are rewarded through automation of repetitive tasks, accelerated deployment cycles, improved scalability, and proactive issue detection. AI-driven tools further enhance development efficiency through intelligent code suggestions, predictive analytics, and automated bug detection. By embracing these modern approaches, organizations can achieve a more secure, efficient, and agile development process, better equipped to adapt to a rapidly evolving technological landscape.

Keywords: DevOps, CI/CD, IaC, Containerization, Orchestration, Monitoring.

1 Introduction

The rapid evolution of technology in software development and deployment has necessitated the use of advanced tools and methodologies to streamline application management in production environments. This paper explores the significant advancements in IT, specifically focusing on the implementation, management, and scaling of applications.

The primary aim is to understand and present various tools and technologies that facilitate efficient deployment and management of applications. Additionally, the goal is to demonstrate the superiority of automated solutions over manual management in every phase of the application lifecycle. Highlighting the indispensable nature of these tools for scaling applications, the article also showcases the benefits of application isolation through containerization technology. By leveraging these tools and technologies, considerable time savings are achieved, ensuring continuous availability and reliability of services. Furthermore, the advantages of versioning application infrastructure via configuration files are discussed.

The core thesis posits that containerization and orchestration tools, along with DevOps elements, significantly enhance the management of the application lifecycle in production environments. It is explained how these tools influence efficient lifecycle management, demonstrating their impact through practical examples that highlight increased deployment efficiency and improved response to user demand spikes and application errors. The aim is to validate the thesis by proving that there are increasingly sophisticated IT tools emerging for managing the application lifecycle.

The motivation behind this research stems from the constant change in today's world, which requires quick adaptation to avoid falling behind. Delays can translate to financial losses. Frequent updates and bug fixes to existing applications, often small in size, require administrators or developers to spend time on repetitive tasks such as building and deploying new packages to the production environment. DevOps practices offer a more sensible solution by automating processes from infrastructure creation to deploying new versions and self-healing and scaling applications.

2 Devops

DevOps is a cultural and technical movement that aims to unify software development (Dev) and IT operations (Ops) to improve collaboration and productivity by automating infrastructure, workflows, and continuously measuring application performance. It emphasizes the integration of developers and operations teams to work together throughout the entire lifecycle of an application, from development and testing to deployment and operations [18].

The DevOps methodology impacts the application lifecycle during the planning, coding, delivery, and operations phases. Each phase is interdependent, and roles are not strictly assigned to specific phases. In a true DevOps culture, every role is involved in each phase to some extent.

During the planning phase, DevOps teams track the progress of application feature development and define and outline future development directions. Scrum tools [9], Kanban boards [24], and appropriately prepared dashboards are used to provide an at-a-glance view of work progress [21].

The coding phase encompasses every aspect – from implementing functionalities, testing, code quality assessment, and merging it with the rest of the project to building artifacts ready for deployment in test or production environments [23]. During this phase, DevOps engineers focus on automating repetitive processes, such as running tests or building new application versions. With the tools used, development teams can concentrate on more creative tasks and deliver application functionalities more quickly.

In the delivery phase, the DevOps team is responsible for deploying and configuring the infrastructure and delivering the application on time. Application delivery includes launching it and uploading new versions in production environments at the right time. To provide reliable solutions, Continuous Delivery and Continuous Deployment concepts are used, and infrastructure is defined as code [21].

The final phase, operations, focuses on maintaining, monitoring, and resolving issues in the application in production environments. DevOps focuses on enhancing security and ensuring high availability [8] and application reliability. Automatic alerts and dashboards visualizing traffic and server resource usage are used to identify problems before they occur [21].

By adopting DevOps practices, organizations can deliver software more rapidly, reliably, and efficiently. This approach also fosters a culture of continuous improvement [25], where feedback loops are shortened, leading to faster identification and resolution of issues. DevOps tools and methodologies, such as CI/CD (Continuous Integration/Continuous Deployment), IaC (Infrastructure as Code), and monitoring solutions, support this collaborative environment, enabling teams to respond swiftly to changes and maintain high-quality service delivery. Through automation and the breaking down of silos between teams, DevOps helps in achieving shorter development cycles, increased deployment frequency, and more dependable releases, thereby meeting the ever-evolving demands of the business landscape [18].

However, DevOps does have its drawbacks. Implementing this methodology is a significant undertaking, often requiring the adoption of a microservices architecture, which increases project complexity. This, in turn, necessitates the maintenance of multiple environments: development, staging, testing, and production, each serving a specific purpose in the development lifecycle, such as isolating code changes, ensuring quality, and deploying stable releases. Ensuring the security of the solutions used is also a critical concern, requiring constant attention to potential vulnerabilities. Additionally, conducting employee training on the necessary tools for effective work in a DevOps culture poses a challenge. Introducing new approaches and technologies can encounter resistance from employees accustomed to older technologies or a work culture that relies less on communication between different teams [31].

3 CI/CD

CI/CD (Continuous Integration/Continuous Deployment) pipelines automate the steps required to bring new code from commit to production, including building, testing, deployment, and infrastructure preparation. Implementing CI/CD minimizes downtime and accelerates code releases by ensuring that code changes are quickly and reliably integrated and deployed, thus enhancing the overall development workflow [18]. This leads to a more agile and responsive development process, capable of meeting rapidly changing business needs.

Continuous Integration is a practice where all changes to the source code are regularly and frequently merged into the main branch of a shared code repository, with each change automatically tested upon commit or merge, and the build process automatically initiated. Through Continuous Integration, errors and security issues can be more easily identified and fixed, even in the early stages of implementation.

In a CI/CD pipeline, testing is typically performed automatically: each code change triggers a series of tests to ensure that the application behaves as expected. This helps identify issues early in the development process and prevents costly fixes in the future. During the continuous integration stage, various types of tests are conducted, including:

- Unit tests, which verify that individual units of code function as expected;
- Integration tests, which check how different modules or services within the application work together;
- Regression tests, which are performed after a bug is fixed to ensure that the same issue does not recur.

Static and Dynamic Application Security Testing (SAST and DAST) are among the most important types of security tests that can be integrated into CI/CD pipelines. As application security and regulatory compliance become increasingly critical, continuous security testing has emerged as a key component of modern software development. These tests help identify vulnerabilities early and ensure that applications remain secure and compliant throughout their lifecycle.

Static Application Security Testing (SAST) is used to identify potential vulnerabilities in the application's source code. As a form of white-box testing, it analyzes the source code itself rather than the behavior of the application during runtime. SAST tools scan code for vulnerabilities such as buffer overflows, SQL injection flaws, and cross-site scripting (XSS) issues. These tools rely on predefined rules and patterns and may be enhanced with semantic analysis to understand the context and implications of the examined code. SAST enables early detection and elimination of threats during the development phase. Upon completion, the tool generates a report outlining detected vulnerabilities categorized by severity, along with remediation recommendations. These reports help developers locate problematic code and understand how to fix the identified issues [30].

Dynamic Application Security Testing (DAST), on the other hand, is a form of runtime testing that simulates real-world attacks to uncover vulnerabilities. Unlike SAST, DAST detects issues

during actual user interactions with the application, such as input validation or authentication weaknesses. It simulates hacker-like attacks and observes the application's responses. This approach is highly effective in identifying flaws and vulnerabilities such as SQL injection, XSS, CSRF (cross-site request forgery), and IDOR (insecure direct object references). DAST tools are also essential for ensuring organizational compliance with regulations such as the Payment Card Industry Data Security Standard (PCI DSS) and the General Data Protection Regulation (GDPR) [29].

By frequently merging changes and running automated tests, the potential for code conflicts is minimized - even when many developers are working on the project. When code is continuously merged and changes are pulled daily into local copies of the project, the likelihood of conflicts during merge requests is reduced. If conflicts or errors do occur, they can be resolved quickly since the context is still fresh in the developers' minds.

Continuous Delivery is a software development practice that works in conjunction with Continuous Integration (CI) to automate the process of delivering infrastructure and deploying applications. After the code is tested and built as part of the CI process, CD takes over in the final stages to ensure that the code is packaged with everything needed to deploy the application in any environment, at any time. Deployments can then be triggered manually, or the process can transition to Continuous Deployment, where deployments are fully automated. With CD, software is built in such a way that it can be deployed to production at any moment [12]. The environment where the application runs can be configured manually by an administrator or tuned through the Continuous Deployment process. When it comes to managing infrastructure elements and configuring environments, the current dominant trend is the Infrastructure as Code approach.

Key technologies in CI/CD include version control systems like Git for managing code, build automation tools such as Jenkins, CircleCI, and GitLab CI/CD for automating the compilation and testing processes, and testing frameworks like JUnit and Selenium for automating tests. These tools work together to streamline the integration, testing, and deployment of code, ensuring a smooth and efficient development workflow. Security testing tools such as Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) can also be integrated into CI/CD pipelines, enabling early detection of vulnerabilities both in the source code and in the running application. Popular SAST tools include SonarQube, Checkmarx, and Fortify, while common DAST tools include OWASP ZAP, Burp Suite, and Acunetix. GitLab offers built-in SAST capabilities as part of its DevSecOps platform, allowing teams to incorporate security scans seamlessly into their CI/CD workflows without needing external tools.

In addition, integrating SAST and DAST tools into CI/CD pipelines supports compliance with widely recognized security standards such as ISO/IEC 27001 (in Europe) and SOC 2 (in USA). These standards require organizations to implement secure development practices and continuous risk assessments, which automated security testing directly facilitates. Maintaining compliance with such frameworks helps organizations not only improve their security posture but also build trust with customers and stakeholders.

Despite the numerous benefits of CI/CD, there are several challenges associated with its implementation. Developing a fully automated CI/CD pipeline requires specialized skills, often necessitating the presence of a DevOps engineer, whose expertise comes at a high cost. Additionally, other team members may need training to adapt to the pipeline's processes, leading to significant upfront investments in both time and resources. CI/CD pipelines also have a steep learning curve, making them difficult to introduce during ongoing projects with tight deadlines. Moreover, legacy systems often lack support for CI/CD, requiring thorough diagnostics and potential upgrades before integration. The success of CI/CD is heavily dependent on the team's commitment to maintaining high standards and meeting integration deadlines; otherwise, the benefits of automation may be lost. Additionally, security must be integrated into every stage of the CI/CD pipeline, as any lapse in security practices can lead to vulnerabilities being introduced into production. Insecure coding practices, improper access controls, and exposed secrets can all be exploited by malicious actors. Therefore, implementing robust security measures, such as static and dynamic security testing (SAST/DAST), secret management, and vulnerability scanning, is essential to safeguarding the integrity and safety of the application throughout the development and deployment process. [5, 7].

4 Infrastructure as Code

Infrastructure as Code (IaC) is a practice that manages infrastructure through code, enabling the use of version control for infrastructure configurations. This approach allows for consistent and reproducible infrastructure setups, reduces the potential for human error, and facilitates the deployment of complex environments with ease and precision. IaC also supports collaboration among teams by making infrastructure changes transparent and reviewable [18].

A configuration script for infrastructure can be written, using appropriate technologies, in two ways – imperatively or declaratively. IaC is inherently declarative by nature. The imperative approach involves defining a list of commands that configure the cloud environment to the desired state. These commands are executed in a defined order. Such a list may contain detailed instructions, but if there is a need to change the configuration after executing it on many machines, all steps must be repeated [32].

Declarative automations, on the other hand, involve specifying the desired state of the environment. The user describes the final state of the infrastructure they want, and the chosen IaC technology takes care of realizing and managing this state [32].

Another aspect of Infrastructure as Code is the type of operations performed – provisioning resources and configuration management. Provisioning encompasses all steps necessary to prepare a new resource on the cloud platform. For example, to deliver a server, one must prepare the network to which the server will be connected, create and attach the storage resource, and set up the operating system. Configuration management, in turn, deals with automating the configuration of individual systems, such as multiple servers, firewalls, or routers [4].

A key concept of the IaC approach is idempotence. This means that if the same script is executed multiple times, the result remains the same. Changes are made to the system only when necessary.

The most popular technologies in Infrastructure as Code include Terraform, which allows teams to define and provision infrastructure in a declarative manner, enabling the management of resources across multiple cloud environments. Ansible and Puppet are also widely used for configuration management, automating the setup and maintenance of servers and other infrastructure components.

Despite the advantages of Infrastructure as Code, several challenges need to be considered. One of the primary concerns is configuration drift, which occurs when the actual infrastructure deviates from the IaC configuration due to manual changes or external updates, such as security patches. This drift can lead to inconsistencies, non-compliance, and even service failures over time. Managing Role-Based Access Control (RBAC) is another significant challenge. Since IaC often requires central storage in repositories like GitHub, improper management of RBAC can result in unauthorized access or security vulnerabilities. Ensuring that access is properly restricted and monitored is crucial to maintaining the security and integrity of the infrastructure. Additionally, the complexity of defining infrastructure configurations, adhering to conventions, and keeping up with tooling updates further complicates the adoption and maintenance of IaC practices [17].

5 Containerization

Containerization involves running isolated processes on a hosted machine, ensuring consistent application behaviour across different environments. Containers encapsulate all dependencies an application needs, promoting portability, enhancing server efficiency, and reducing maintenance costs. This technology streamlines the development, testing, and deployment processes, leading to more predictable and reliable software delivery. Containers also simplify the management of application dependencies and environment configurations [11].

Containers are called lightweight because, unlike virtual machines, they do not require their own operating system (guest OS, see Fig. 1). Containers use the host operating system's kernel, saving gigabytes of disk space on the machine they run on. This disk space saving directly impacts the cost of maintaining virtual machines in a cloud environment. An additional advantage of sharing the host operating system is the shorter startup time for the containerized process [15].

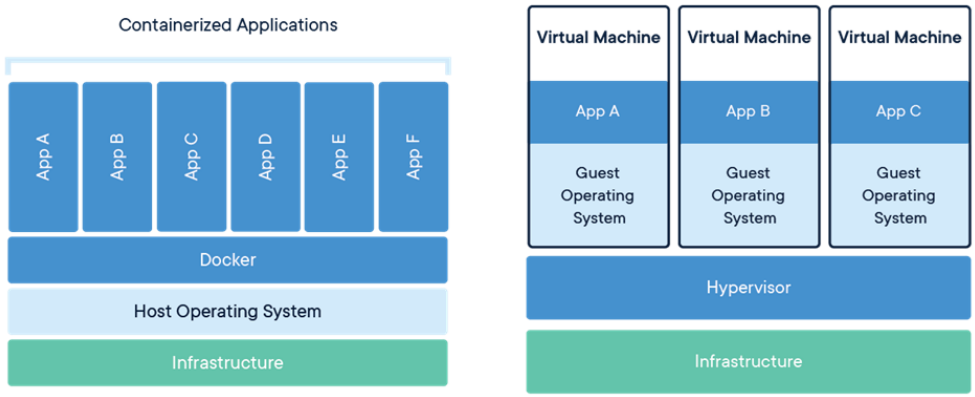

Figure 1. Comparison of Containers and Virtual Machines. Source: https://www.docker.com/resources/what-container/

The architecture of containerization consists of: IT infrastructure as the base layer, the host operating system, the runtime engine (an intermediary between the container and the host operating system), and the application process. To create and run a container, a container image must first be prepared. A container image is a read-only template that consists of a list of instructions defining the container. Each instruction constitutes a separate layer of the image (see Fig. 2) [11, 15].

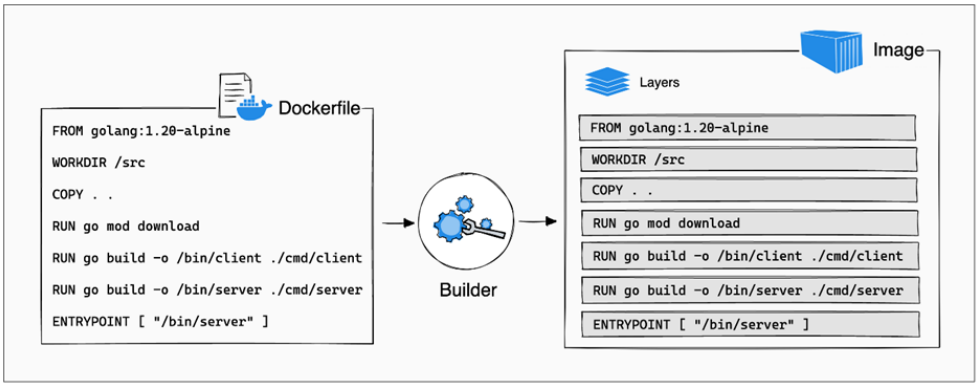

Figure 2. How a Dockerfile translates into a stack of layers in a container image. Source: https://docs.docker.com/build/guide/layers/

Containerization enables applications to adhere to the principle of "write once, run anywhere" – regardless of the operating system, whether it's a local machine, on-premises data center, hybrid cloud, or multicloud environments [15].

To wrap up the discussion on containerization, it's essential to highlight key technologies in this field. Docker is a widely-used platform that simplifies the creation, deployment, and management of containerized applications. Podman offers similar functionality to Docker but is daemonless and rootless, providing enhanced security features. Notably, containerd is a core container runtime used by several container platforms, including Docker. It handles critical aspects of container lifecycle management, such as image transfer, container execution, and storage.

Despite the many advantages of containerization, several challenges remain. Security concerns persist due to the shared kernel model, which can expose vulnerabilities. Management and orchestration complexities, particularly in large-scale deployments, continue to be a challenge. Persistent storage management is also a concern, especially for stateful applications. Additionally, networking complexities in multi-cloud and hybrid environments pose significant challenges. Finally, runtime compatibility issues require careful attention to ensure smooth interoperability across platforms [34].

6 Orchestration of container clusters

Orchestration of container clusters refers to the automated management of containerized applications across multiple hosts. Tools like Kubernetes enable the scaling, networking, and lifecycle management of containers, ensuring that applications are efficiently distributed and resilient. Orchestration simplifies the handling of containerized environments, allowing for automated scaling, self-healing, and optimized resource utilization [26]. This results in highly available and scalable applications that can adapt to varying workload demands.

Container orchestration automates all aspects of container management. It focuses on managing the container lifecycle in a dynamically changing environment. It is used for tasks such as [1, 2]:

- Configuring and scheduling containers
- Provisioning and deploying containers
- Managing container availability (including automatic recovery in case of failure)
- Scaling containers to distribute application load across multiple instances
- Allocating resources among containers
- Monitoring container health
- Securing interactions between containers
- Running many different containerized applications
- Running complex applications composed of a large number of different microservices

The container orchestrator manages containers within groups of server instances (called nodes). A group of nodes that run related containers forms a cluster. For orchestration to be possible, the container orchestrator requires containerization technology to be running on each node in the cluster (e.g., Docker). Nodes are managed by a designated master node. The administrator uses the master node to manage and monitor the container orchestration tool [2].

To create and manage the state of containers, the orchestration tool reads a declarative configuration file. Using information about the desired state of the system, the tool retrieves the necessary container images from the container registry, prepares the containers, and determines the networking requirements between them. The tool then schedules and deploys the multi-container application across the cluster. The specific node to run a particular container is chosen based on the container's requirements and the node's resource constraints (CPU, memory, etc.) [2].

Managing the overall system health and maintaining the specified performance state is achieved by allocating resources among containers and balancing the application traffic load. Additionally, the orchestrator deploys containers to new nodes and removes unnecessary ones. This ensures high application availability and fault tolerance in very large, complex, multi-container systems [2].

In the context of container orchestration, several key technologies shape the landscape. Kubernetes is the leading platform, offering extensive automation for deploying, scaling, and managing containerized applications across clusters. Docker Swarm provides a simpler, integrated solution with Docker for straightforward orchestration needs. Additionally, Apache Mesos supports container orchestration alongside other workloads, catering to large-scale distributed systems. Each of these technologies addresses different requirements, enhancing the efficiency and scalability of containerized environments.

While container orchestration offers significant benefits, it also comes with several challenges. Complexity is a major issue, as managing these systems requires specialized skills and expertise, resulting in a steep learning curve. Resource requirements can be high, with substantial demands for both hardware and human resources, which may be burdensome for smaller organizations. Security challenges arise due to the complexities of container isolation and network security, necessitating strong protective measures. Continuous maintenance is crucial, as orchestration platforms need regular updates and patches, which can be demanding. Networking complexity increases in multi-cluster and hybrid cloud environments, making service communication challenging. Application compatibility may also be problematic, especially for legacy systems that need significant modifications to be containerized [16].

7 Monitoring and observability

Monitoring is a crucial component in modern IT environments, ensuring the health, performance, and security of applications and infrastructure. It involves the continuous observation of systems, networks, and applications to detect anomalies, measure performance, and gather insights.

Monitoring is the process of collecting, analyzing, and using information to track a program's progress in achieving its goals and to guide management decisions. This process focuses on tracking specific metrics and visualizing them in the form of dashboards [19].

Observability is the ability to understand the internal state of a system by analyzing the data it generates (logs, metrics, traces). Observability helps teams analyze what is happening in various multi-cloud environments, allowing them to detect and resolve root causes of issues [19]. Observability tools utilize algorithms based on mathematical control theory to understand the relationships between systems in a multi-layered IT infrastructure. When the tool detects an anomaly, it alerts the team and provides the data needed for quick problem resolution [20].

Effective monitoring helps teams proactively identify and resolve issues before they impact end users. Tools like Prometheus, Grafana, and New Relic collect and visualize metrics, logs, and traces, providing a comprehensive view of the system's state. These tools enable real-time alerts and notifications, allowing for immediate action when predefined thresholds are breached. Additionally, monitoring supports capacity planning and optimization by providing data on resource utilization and performance trends. In a DevOps context, integrated monitoring solutions contribute to a feedback loop that enhances the continuous improvement process, ensuring that applications remain reliable, performant, and secure throughout their lifecycle [6].

Implementing effective monitoring and observability in modern IT environments presents several challenges. The complexity of infrastructure across diverse technologies, such as multi-cloud environments and various service models, complicates tool compatibility and data aggregation, making it difficult to maintain a unified performance view. Scalability and performance monitoring pose challenges in dynamic cloud environments, where predicting resource needs and ensuring consistent performance across fluctuating workloads is critical. Integration with legacy systems is another hurdle, requiring careful management of compatibility and security risks as older systems are integrated with modern monitoring solutions. Security and compliance concerns also arise, necessitating robust measures to protect data and meet stringent regulatory requirements. Additionally, data unification and alert management are critical yet challenging, as diverse monitoring tools generate vast amounts of data, leading to potential alert fatigue and difficulties in extracting actionable insights. Finally, the sheer volume of data generated by complex systems can overwhelm observability tools, while data silos can fragment the view of the system, hindering the identification of root causes. Balancing these challenges with cost considerations is essential to align monitoring initiatives with budget constraints [3, 14].

8 AI solutions

AI is increasingly integral to the software development lifecycle, providing sophisticated solutions that enhance efficiency and accuracy. Its ability to automate repetitive tasks allows developers to focus on more complex and creative aspects of development.

One of the pivotal areas where AI makes a significant impact is in code quality assurance. By integrating AI-powered code review tools, developers can receive instant feedback on potential

bugs, security vulnerabilities, and adherence to coding standards. These tools not only expedite the review process but also ensure a higher level of code reliability and security [13,27].

Furthermore, AI-driven predictive analytics can forecast potential system failures or performance bottlenecks by analysing historical data and usage patterns. To detect anomalies, data such as account trends, seasonal day-of-week, and time-of-day patterns are compared. It is suited for metrics with strong trends and recurring patterns that are hard to monitor with threshold-based alerting [10, 28].

Additionally, AI can assist in generating code, automating the creation of boilerplate code and even complex algorithms, thereby accelerating the development process. AI-assisted software development relies on advanced language models (LLMs) and ML algorithms to generate code. By using AI responsibly and thoughtfully, developers can discover new solutions faster and create software more efficiently [27].

Beyond code generation and analytics, artificial intelligence also enhances container orchestration and resource management. Machine learning models can predict workload patterns, dynamically allocate resources, and optimize deployment strategies across distributed systems. Studies show that AI-based orchestration significantly improves scalability, fault tolerance, and energy efficiency by learning from system telemetry and autonomously adapting to changing workloads [33].

A more advanced approach to this concept is represented by the CODECO framework (COgnitive Decentralized Container Orchestration). CODECO introduces a distributed cognitive orchestration architecture that applies artificial intelligence to enhance the adaptability and resilience of containerized environments. Unlike traditional centralized orchestrators such as Kubernetes, CODECO decentralizes decision-making across multiple orchestration nodes. Each node uses local machine learning models to predict workload fluctuations, manage container placement, and recover from faults autonomously. This decentralized AI-based model eliminates the single point of failure, reduces latency in orchestration decisions, and enables intelligent self-management of large-scale heterogeneous cloud edge systems. As a result, frameworks like CODECO mark a significant step toward the realization of fully autonomous self-optimizing container ecosystems [22].

This proactive approach allows teams to address issues before they escalate, maintaining optimal application performance and user satisfaction. It also fosters a culture of continuous improvement, as AI tools can provide ongoing insights and recommendations not only in software design and testing but also in infrastructure management and orchestration. With emerging frameworks such as CODECO, artificial intelligence is evolving from a supportive tool into an autonomous decision-making component of modern software ecosystems. Ultimately, the integration of AI across all stages of the development and deployment lifecycle leads to more resilient, scalable, and adaptive software solutions.

9 Conclusion

There are many different tools on the market that help teams manage the application lifecycle. This article presents a few of them. Thanks to these tools, it is possible to implement changes more efficiently and detect vulnerabilities in software more quickly. Adopting these in an organization requires financial investment in purchasing licenses for technologies, learning the discussed tools, and reorganizing the team to introduce a DevOps culture. The money spent on implementing the tools and approaches discussed in the article will pay off multiple times by freeing specialists from having to perform repetitive and monotonous tasks and preventing failures in production IT systems.

References

1. Avi Networks, : Container Orchestration Definition, https://avinetworks.com/glossary/container-orchestration/. Last accessed: 12 Jul 2024.
2. Amazon Web Services, Inc., : What is Container Orchestration?, https://aws.amazon.com/what-is/container-orchestration/. Last accessed: 12 Jul 2024.
3. Ballav S., : Top five cloud monitoring challenges, https://www.site24x7.com/blog/cloud-monitoring-challenges. Last accessed: 25 Aug 2024.
4. Barney D., : Infrastructure Provisioning vs. Configuration Management vs. Configuration Orchestration: How IaC Makes Them All Better, https://www.chef.io/blog/infrastructure-provisioning-vs-configuration-management-vs-configuration-orchestration-how-iac-makes-them-all-better. Last accessed: 11 Jul 2024.
5. Belfiore R., : Pros and Cons of CI/CD Pipelines, https://www.bairesdev.com/blog/pros-and-cons-of-ci-cd-pipelines/ Last accessed: 25 Aug 2024.
6. Bigelow S. J., : The definitive guide to enterprise IT monitoring, https://www.techtarget.com/searchitoperations/The-definitive-guide-to-enterprise-IT-monitoring. Published: 25 Jan 2024.
7. Check Point Software Technologies Ltd., : What is CI/CD Security?, https://www.checkpoint.com/cyber-hub/cloud-security/what-is-ci-cd-security. Last accessed: 27 Apr 2025.
8. Cisco Systems, Inc., : What Is High Availability?, https://www.cisco.com/c/en/us/solutions/hybrid-work/what-is-high-availability.html. Last accessed: 10 Jul 2024.
9. Claire D., : What is scrum and how to get started, https://www.atlassian.com/agile/scrum. Last accessed: 10 Jul 2024.
10. Datadog, : Docs: Anomaly Monitor, https://docs.datadoghq.com/monitors/types/anomaly. Last accessed: 12 Jun 2024.
11. Docker Inc., : Use containers to Build, Share and Run your applications, https://www.docker.com/resources/what-container/. Last accessed: 12 Jun 2024.
12. GitLab B.V., : What is CI/CD? https://about.gitlab.com/topics/ci-cd/. Last accessed: 10 Jul 2024.

13. GitHub, Inc., : What is CI/CD?, https://github.com/marketplace/actions/ai-code-review-action. Last accessed: 12 Jun 2024.
14. Gowdy M., : Common observability challenges and steps to overcome them, https://blog.quest.com/common-observability-challenges-and-steps-to-overcome-them/. Published: 4 Jan 2024.
15. Smalley I., Susnjara S., : What is containerization?, https://www.ibm.com/topics/containerization. Last accessed: 11 Jul 2024.
16. Kazlouskaya K., : Kubernetes Advantages and Disadvantages, https://ostridelabs.com/kubernetes-advantages-and-disadvantages/. Updated: 13 May 2024.
17. Kimachia K., : Benefits and Drawbacks of Infrastructure as Code (IaC), https://www.enterprisenetworkingplanet.com/data-center/infrastructure-as-code/ Published: 6 Jun 2022.
18. Krief M.: Learning DevOps: A comprehensive guide to accelerating DevOps culture adoption with Terraform, Azure DevOps, Kubernetes, and Jenkins, 2nd edn., Packt Publishing (2023).
19. Livens J., : Observability vs. monitoring: What's the difference?, https://www.dynatrace.com/news/blog/observability-vs-monitoring/. Last accessed: 12 Jul 2024.
20. Magnusson A., : Observability vs. Monitoring: Understanding the Difference, https://www.strongdm.com/blog/observability-vs-monitoring. Last accessed: 12 Jul 2024.
21. Microsoft, : What is DevOps?, https://azure.microsoft.com/resources/cloud-computing-dictionary/what-is-devops. Last accessed: 10 Jul 2024.
22. R. Mijuskovic, S. Borthakur, T. O. Morsh, S. Tschiatschek, and C. Gkantsidis, : A Framework for Cognitive, Decentralized Container Orchestration (CODECO), *IEEE Access*, vol. 12, pp. 112345–112360, 2024.
23. Pająk T., : Co to jest DevOps?, https://blog.conlea.pl/co-to-jest-devops. Last accessed: 10 Jul 2024.
24. Radigan D., : What is kanban?, https://www.atlassian.com/agile/kanban. Last accessed: 10 Jul 2024.
25. Rehkopf M., : What is continuous improvement?, https://www.atlassian.com/agile/project-management/continuous-improvement. Last accessed: 12 Jul 2024.
26. Red Hat Inc., : What is container orchestration?, https://www.redhat.com/en/topics/containers/what-is-container-orchestration. Last accessed: 12 Jun 2024.
27. SonarSource SA., : A developer's guide to AI-assisted software development, https://www.sonarsource.com/learn/ai-assisted-software-development. Last accessed: 12 Jun 2024.
28. Splunk Inc., https://www.splunk.com/en_us/solutions/splunk-artificial-intelligence.html. Last accessed: 12 Jun 2024.
29. Schmitt J., : DAST: A guide to dynamic application security testing https://circleci.com/blog/dynamic-application-security-testing-dast. Last accessed: 27 Apr 2025.
30. Schmitt J., : SAST: A guide to static application security testing https://circleci.com/blog/static-application-security-testing-sast/#c-consent-modal. Last accessed: 27 Apr 2025.

31. Subramanian P. S., : 10 Major DevOps Challenges And Issues, https://www.ideas2it.com/blogs/devops-challenges. Last accessed: 21 Jul 2024.
32. Trend Micro Incorporated, : What Is Infrastructure as Code? https://www.trendmicro.com/en_us/what-is/cloud-security/infrastructure-as-code.html. Last accessed: 11 Jul 2024.
33. Zhong Z., Wang J., Xiang T., Xu C., : Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions, arXiv preprint arXiv:2106.12739 (2021).
34. [x]cube LABS, : The Advantages and Disadvantages of Containers, https://www.xcubelabs.com/blog/the-advantages-and-disadvantages-of-containers/. Published: 23 Feb 2023.