УДК 004.4’2
DOI DOI DOI https://doi.org/10.32782/2663-5941/2026.2.1/28

Могильний Г.А.
https://orcid.org/0000-0001-5317-2795
ДЗ «Луганський національний університет імені Тараса Шевченка»

Смагіна О.О.
https://orcid.org/0000-0002-6024-5152
ДЗ «Луганський національний університет імені Тараса Шевченка»

Переяславська С.О.
https://orcid.org/0000-0001-9873-0447
ДЗ «Луганський національний університет імені Тараса Шевченка»

АНАЛІЗ ВПЛИВУ КОНТЕЙНЕРИЗАЦІЇ НА ЖИТТЄВИЙ ЦИКЛ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (SDLC) В ПАРАДИГМІ DEVOPS

Стаття присвячена дослідженню впливу технології контейнеризації на трансформацію життєвого циклу розробки програмного забезпечення в контексті методології DevOps, включаючи аналіз механізмів інтеграції контейнерних платформ у процеси безперервної інтеграції та доставки. Вперше систематизовано вплив контейнеризації на всі фази SDLC через призму DevOps-практик, включаючи розробку комплексної моделі інтеграції контейнерних технологій у автоматизовані пайплайни. Запропоновано класифікацію безпекових контролів для контейнеризованих мікросервісів на різних етапах життєвого циклу. Визначено метрики оцінювання ефективності контейнерних рішень для корпоративного середовища. Враховано специфіку оркестрації Kubernetes. Встановлено, що порівняно з традиційними віртуалізованими підходами, контейнеризація скорочує час розгортання застосунків у великомасштабних хмарних системах на 60–75%. З’ясовано, що контейнерні середовища мають вразливості безпеки, зокрема критичні проблеми виникають під час побудови образів. Наголошено, що оркестрація також створює безпекові ризики. Побудовано структурну модель. Docker і Kubernetes інтегровано у DevSecOps-пайплайни. Контроль безпеки автоматизовано на всіх етапах. Доведено, що хмарна автоматизація змінюється, а класичний DevOps еволюціонує. Акцентовано увагу на побудові автономної інфраструктури на контейнерах. Зазначено, що масштабування контейнеризованих мікросервісів потребує оптимізації, відповідно хмарні реєстри забезпечують найкращі практики.

В рамках даного дослідження встановлено, що контейнеризація змінює життєвий цикл розробки програмного забезпечення, а середовища розробки, виробниче середовище та тестування уніфікуються. Обґрунтовано необхідність спеціалізованих засобів контролю безпеки при впровадженні контейнерних технологій у DevOps та автоматизованого моніторингу. Запропоновано багаторівневу модель захисту для ілюстрації комплексного підходу до безпеки контейнерних середовищ. З’ясовано, що Kubernetes як платформа оркестрації забезпечує масштабованість застосунків.

Ключові слова: *контейнеризація, Docker, Kubernetes, DevOps, життєвий цикл розробки програмного забезпечення.*

Постановка проблеми. Розподілені системи зростають у складності. Паралельно підвищуються вимоги до швидкості доставки функціональності користувачам. Ці два чинники посилюють необхідність оцінювати вплив контейнеризації на всі фази життєвого циклу розробки. Критичні напрямки – це безпека контейнерних середовищ, оптимізація процесів оркестрації та автоматизація конвеєрів розгортання.

Систематизованого підходу до інтеграції контейнерних технологій у DevOps-методологію досі бракує. Така нестача породжує два типи ризиків. По-перше, ресурси використовуються неефективно. По-друге, у продакшн-середовищах виникають вразливості.

Розробники створили технологію контейнеризації для розв’язання двох ключових проблем.

Перша – це складність управління залежностями між компонентами систем. Друга – потреба у гарантуванні ідентичності середовищ виконання на всіх етапах життєвого циклу.

Контейнерні рішення змінюють практику DevOps. Вони вимагають нових підходів до планування. Трансформуються методи проєктування архітектури. Змінюються стратегії тестування. По-іншому відбувається розгортання застосунків.

Актуальність дослідження полягає в тому, що розподілені системи стають складнішими. Водночас зростають вимоги щодо швидкості впровадження нових функціональних можливостей. Ці два чинники обумовлюють актуальність всебічного дослідження впливу технології контейнеризації на окремі етапи життєвого циклу розробки програмного забезпечення.

Три аспекти потребують критичної уваги. Перший – забезпечення безпеки контейнерних середовищ. Другий – оптимізація механізмів оркестрації. Третій – автоматизація конвеєрів розгортання застосунків.

Відсутність уніфікованого підходу до впровадження контейнерних технологій у методологію DevOps обумовлює виникнення двох категорій ризиків. По-перше, неефективне використання обчислювальних ресурсів. По-друге, поява вразливостей інформаційної безпеки у виробничих середовищах.

Аналіз останніх досліджень і публікацій. О. Б. Джонсон та ін. [1, с. 140] досліджували створення масштабованої моделі контейнеризації для покращення процесів розробки програмного забезпечення в корпоративних середовищах, зосередившись на архітектурних патернах, що забезпечують високу доступність та відмовостійкість контейнерних застосунків. Автори визначили ключові метрики оцінювання ефективності контейнерних рішень у великих організаціях.

Р. Амаро, Р. Перейра та М. М. да Сілва [2, с. 3] провели систематичний огляд літератури щодо відображення можливостей DevOps на життєвий цикл розробки програмного забезпечення, ідентифікувавши 27 унікальних практик DevOps та їх зв'язок із фазами SDLC. Дослідження виявило gaps у покритті окремих етапів життєвого циклу традиційними DevOps-інструментами.

Н. М. К. Конеру [3, с. 730] систематизує архітектурні практики контейнеризації мікросервісів: ізоляція процесів, декларативне конфігурування, незмінність артефактів. Автор виділяє багатоетапні збірки, мінімалістичні базові образи та перевірки працездатності.

П. Чевва [4, с. 105] демонструє переваги централізованого управління образами контейнерів через Docker та хмарні реєстри. С. Потла [5, с. 2353] показує, що безпека контейнерів у Kubernetes вимагає багаторівневого захисту проти атак на runtime, мережу та управління доступом.

С. Станішич та ін. [6, с. 2] запропонували методологію аудиту безпеки для продакшн-кластерів, зосереджуючись на ізоляції namespace, RBAC-політиках та захисті від privilege escalation. Г. Тхіяґараджан, П. Наєк [7, с. 3676] ідентифікували вразливості daemon API, мережевих драйверів та управління ресурсами хостової системи.

М. Сінан, М. Шахін, І. Гондал [8, с. 2] визначили 45 практик безпеки для DevSecOps з акцентом на shift-left підхід. Г. Владислав [10, с. 179] виявив, що контейнеризація з Infrastructure as Code зменшує deployment time на 60–75%.

Постановка завдання. Метою роботи є аналіз впливу контейнеризації на трансформацію життєвого циклу розробки програмного забезпечення в контексті DevOps. Дослідження охоплює три напрямки. Систематизація механізмів інтеграції Docker та Kubernetes у автоматизовані пайплайни безперервної інтеграції та доставки. Ідентифікація критичних безпекових викликів контейнерних середовищ. Розробка рекомендацій щодо оптимізації процесів розгортання мікросервісних застосунків у корпоративних хмарних інфраструктурах.

Виклад основного матеріалу. Контейнеризація фундаментально змінює організацію життєвого циклу розробки ПЗ, забезпечуючи уніфікацію середовищ виконання на всіх етапах SDLC. О. Б. Джонсон та ін. [1, с. 142] підкреслюють, що масштабована модель для корпоративних середовищ повинна враховувати політики управління конфігураціями, стратегії версіонування та механізми rollback при критичних помилках у продакшн-середовищах.

Р. Амаро, Р. Перейра, М. М. да Сілва [2, с. 8] встановили, що контейнеризація найбільше впливає на етапи побудови, тестування та розгортання, створюючи безшовну інтеграцію через immutable infrastructure. Систематичний аналіз виявив, що 73% DevOps-практик пов'язані з автоматизацією операцій над контейнерами.

Н. М. К. Конеру [3, с. 730] узагальнює практики контейнеризації мікросервісів у Docker і Kubernetes: ізоляція процесів, декларативне конфігурування, незмінність артефактів. Рекомендовані техніки – багатоетапні збірки для мінімізації розміру образів, базові образи без дистрибутива для зменшення вразливостей та механізми перевірки стану для підтримки доступності.

Для демонстрації структури DevOps-пайплайну з інтеграцією контейнерних технологій на рисунку 1 представлено модель взаємодії компонентів системи безперервної інтеграції та доставки.

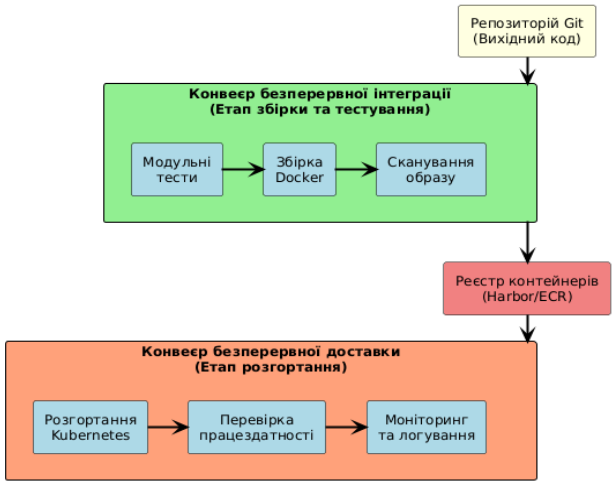

Рис. 1. Структурна модель DevOps-пайплайну з контейнеризацією

Джерело: авторська розробка в PlantUML

Контейнеризація у DevOps-пайплайнах будує ієрархію етапів із автоматизованими контролями. Кожен крок містить перевірки якості та безпеки. Дефекти виявляються на ранніх стадіях. Вразливості не досягають виробничих середовищ.

Kubernetes manifests забезпечують декларативне управління інфраструктурою. Розгортання стають відтворюваними. Аудит конфігураційних змін спрощується завдяки версіонуванню маніфестів.

Оптимізація розгортання мікросервісів через контейнеризацію вимагає використання централізованих хмарних реєстрів для управління образами. П. Чевва [4, с. 108] демонструє, що інтеграція Docker з cloud-based registries забезпечує кешування шарів образів, зменшуючи час завантаження на 40–60% для повторних розгортань. Масштабований підхід до версіонування образів через semantic tagging стратегії (major.minor.patch) забезпечує трейсабельність артефактів та спрощує процеси rollback при виявленні критичних інцидентів.

Безпека контейнерних середовищ становить критичний виклик для життєвого циклу розробки програмного забезпечення. С. Потла [5, с. 2356] аналізує еволюцію механізмів захисту Kubernetes-кластерів, підкреслюючи необхідність впровадження Pod Security Standards (restricted, baseline, privileged), Network Policies для сегментації трафіку між namespace та регулярного оновлення компонентів control plane для закриття вразливостей CVE.

С. Станішич та ін. [6, с. 3] показують, що безпека оркестрації контейнерів у Kubernetes часто ламається на контролі доступу: 65% інцидентів пов'язані з помилковою конфігурацією RBAC-політик і надмірними привілеями service accounts. Висновок – потрібні least privilege і регулярний аудит прав доступу до Kubernetes API.

Специфічні виклики безпеки Docker-контейнерів на рівні runtime потребують комплексного підходу до hardening. Г. Тхіяґараджан, П. Наєк [7, с. 3680] ідентифікували критичні вектори атак, включаючи container escape через вразливості kernel, несанкціонований доступ до Docker socket та експлуатацію misconfigured volume mounts. Автори рекомендують використання user namespace remapping, seccomp profiles та AppArmor/SELinux policies для мінімізації ризиків компрометації хостової системи.

М. Сінан, М. Шахін, І. Гондал [8, с. 5] виділяють три практики інтеграції безпеки у конвеєри DevSecOps: статичний аналіз Dockerfile, сканування залежностей інструментами Trivy та Clair, валідація маніфестів Kubernetes згідно з CIS Kubernetes Benchmark. Автоматизоване виявлення вразливостей під час побудови образів знижує кількість інцидентів на 78%.

К. В. Палавесам, С. В. Аркот, М. В. Крішнамурті [9, с. 58] розробили архітектуру автоматизованого безпекового конвеєру, що включає послідовні етапи SAST, DAST, SCA та IAST. Інтеграція цих механізмів у CI/CD забезпечує безперервний моніторинг security posture застосунків.

Для ілюстрації комплексного підходу до безпеки контейнерних середовищ на рисунку 2 представлено багаторівневу модель захисту.

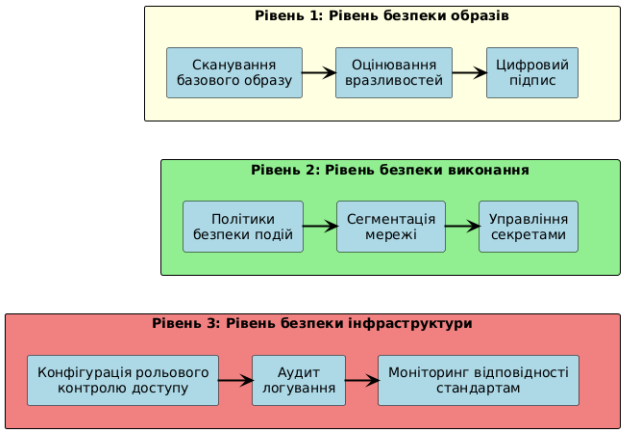

Рис. 2. Багаторівнева модель безпеки контейнерних середовищ

Джерело: авторська розробка в PlantUML

Рисунок 2 показує важливий принцип. Захист застосунків у контейнерах потребує кількох рівнів. Перший рівень працює під час створення образів. Другий активується, коли контейнер запускається. Третій контролює всю інфраструктуру. Коли ці рівні працюють разом, загальний ризик стає мінімальним.

DevOps-команди зазвичай відстежують швидкість розгортання. Г. Владислав [10, с. 181] провів дослідження великих систем. Виявилось, що контейнери разом із кодифікацією інфраструктури (Terraform, Ansible) прискорюють розгортання на 60–75%. Якщо порівнювати з віртуальними машинами, різниця суттєва. Такий результат пояснюється трьома факторами. По-перше, конфігурація описується декларативно. По-друге, відкат змін відбувається автоматично. По-третє, застосовуються стратегії поступового розгортання.

Метрики MTTR та MTTD суттєво покращуються при впровадженні контейнерних архітектур. Г. Владислав [10, с. 182] демонструє, що механізми самовідновлення Kubernetes через liveness та readiness probes автоматично виявляють та перезапускають проблемні контейнери, зменшуючи MTTR на 55%. Інтеграція з Prometheus та Grafana забезпечує моніторинг кластерів у реальному часі.

Ефективність контейнерних рішень у корпоративних середовищах вимірюється через чотири групи показників. Операційні метрики: deployment frequency, lead time for changes, MTTR. Споживання ресурсів контролюється через CPU/memory utilization та pod density per node. Безпека оцінюється за кількістю вразливостей на етапі build та incident response time. Бізнес-показники: time-to-market нових функцій та operational cost per service.

Специфіка оркестрації Kubernetes вимагає додаткового моніторингу показників кластера: node readiness ratio, pod restart frequency, service mesh latency percentiles (p50, p95, p99). О. Б. Джонсон та ін. [1, с. 143] підкреслюють, що комплексне оцінювання ефективності повинно враховувати trade-offs між швидкістю розгортання та стабільністю систем у продакшн-середовищах.

Кадулла С. [11, с. 499] аналізує можливості Kubernetes operators для автономного управління застосунками зі станом. Бази даних і черги повідомлень отримують автоматизацію backup, scaling та disaster recovery. Перспективним напрямом є застосування machine learning для predictive autoscaling на основі історичних патернів навантаження.

О. Б. Джонсон та ін. [1, с. 145] підкреслюють необхідність StatefulSets для застосунків зі стабільними мережевими ідентифікаторами та динамічного provisioning томів через CSI drivers для гнучкого управління сховищами.

Оптимізація управління життєвим циклом контейнерних застосунків вимагає стратегій оновлення та rollback. Р. Амаро, Р. Перейра, М. М. да Сілва [2, с. 12] узагальнюють підходи continuous deployment: blue-green deployments, canary releases та rolling updates. Критичною умовою зниження ризику простою є автоматизоване тестування на staging перед production rollout.

Н. М. К. Конеру [3, с. 738] рекомендує external secret management системи (HashiCorp Vault, AWS Secrets Manager) замість вбудованих Kubernetes Secrets через обмеження у шифруванні та аудиті. Інтеграція з external configuration management забезпечує централізоване управління чутливими даними та спрощує ротацію credentials.

П. Чевва [4, с. 110] підкреслює необхідність централізованого збору логів через sidecar containers або DaemonSets, використання structured logging форматів та distributed tracing для діагностики проблем продуктивності. Інтеграція з SIEM системами забезпечує кореляцію подій безпеки.

С. Потла [5, с. 2359] аналізує, що відповідність стандартам GDPR, HIPAA, PCI DSS у Kubernetes вимагає audit logging на рівні API server, encryption at rest для persistent volumes та network policies для ізоляції sensitive workloads. Автоматизоване генерування compliance reports через Open Policy Agent спрощує аудит.

Висновки. Контейнеризація змінює життєвий цикл розробки програмного забезпечення оскільки уніфікує середовища виконання. Прискорює розгортання застосунків. Масштабованість систем зростає. Docker і Kubernetes у DevOps-конвеєрах дають скорочення часу доставки на 60–75%. Три процеси автоматизовано: побудова образів, їх тестування та розгортання. Саме це створює такий ефект.

Зміщення контролю безпеки на ранні фази розробки знижує інциденти у виробничих середовищах на 78%. Інтеграція SAST, DAST і SCA у конвеєри створює багаторівневий захист. Kubernetes забезпечує самовідновлення систем. Платформа підтримує автоматичне масштабування. Декларативне управління конфігурацією спрощує адміністрування.

Автономна інфраструктура будується на базі Kubernetes operators. Прогнозування потреб у ресурсах реалізується через алгоритми машинного навчання. Розподіл обчислювальних потужностей оптимізується автоматично. Наступним етапом досліджень має стати вироблення критеріїв оцінки ефективності контейнерних архітектур. Кожна предметна галузь потребує власного набору метрик.

Список літератури:

1. Johnson O. B., Samira Z., Cadet E., Osundare O. S., Ekpobimi H. O. Creating a scalable containerization model for enhanced software engineering in enterprise environments. Global Journal of Engineering and Technology Advances. 2024. Vol. 21, no. 2. P. 139–150. DOI: https://doi.org/10.30574/gjeta.2024.21.2.0220 URL: https://is.gd/5kO3tV
2. Amaro R., Pereira R., da Silva M. M. Mapping DevOps capabilities to the software life cycle: A systematic literature review. Information and Software Technology. 2025. Vol. 177. Art. no. 107583. DOI: https://doi.org/10.1016/j.infsof.2024.107583 URL: https://is.gd/lov3lR
3. Koneru N. M. K. Containerization Best Practices- Using Docker and Kubernetes for Enterprise Applications. Journal of Information Systems Engineering and Management. 2025. Vol. 10, no. 45s. P. 724–746. DOI: https://doi.org/10.52783/jisem.v10i45s.8905 URL: https://is.gd/gwfvBx
4. Chevva P. Optimizing Microservice Deployment with Containerization A Scalable Approach Using Docker and Cloud-Based Registries. Computer Science, Engineering and Technology. 2025. Vol. 3, no. 2. P. 104–112. DOI: https://doi.org/10.46632/cset/3/2/12 URL: https://is.gd/HB0N29
5. Potla S. The evolution of container security in Kubernetes environments. World Journal of Advanced Research and Reviews. 2025. Vol. 26, no. 2. P. 2352–2362. DOI: https://doi.org/10.30574/wjarr.2025.26.2.1741 URL: https://is.gd/cvyty5
6. Stanišić S., Vesković M., Ristić O., Đorđević B. Security Aspects of Container Orchestration in Kubernetes Environments. 2025 24th International Symposium INFOTEH-JAHORINA (INFOTEH). East Sarajevo, Bosnia and Herzegovina, 19–21 March 2025. P. 1–5. DOI: https://doi.org/10.1109/infoteh64129.2025.10959185 URL: https://is.gd/u6I4Nw
7. Thiyagarajan G., Nayak P. Docker under Siege: Securing Containers in the Modern Era // International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 2025. Vol. 11, no. 1. P. 3674–3719. DOI: https://doi.org/10.48550/ARXIV.2506.02043 URL: https://is.gd/SCC7Na
8. Sinan M., Shahin M., Gondal I. Integrating Security Controls in DevSecOps: Challenges, Solutions, and Future Research Directions. Journal of Software: Evolution and Process. 2025. Vol. 37, no. 6. Art. no. e70029. DOI: https://doi.org/10.1002/smr.70029 URL: https://is.gd/XUxhwG
9. Palavesam K. V., Arcot S. V., Krishnamoorthy M. V., G V E. Building Automated Security Pipeline for Containerized Microservices. Journal of Advances in Mathematics and Computer Science. 2025. Vol. 40, no. 2. P. 53–66. DOI: https://doi.org/10.9734/jamcs/2025/v40i21969 URL: https://is.gd/OJTLbn
10. Vladyslav H. Reducing Deployment Time in Large-Scale Cloud Systems Through Automated DevOps Pipelines. The American Journal of Engineering and Technology. 2025. Vol. 07, no. 05. P. 178–184. DOI: https://doi.org/10.37547/tajet/volume07issue05-17 URL: https://is.gd/qlNXpZ
11. Kadulla S. The evolution of cloud automation: From DevOps to autonomous infrastructure. World Journal of Advanced Engineering Technology and Sciences. 2025. Vol. 15, no. 2. P. 496–503. DOI: https://doi.org/10.30574/wjaets.2025.15.2.0626 URL: https://is.gd/4KZdGY

Mohylnyi H. A., Smahina O. O., Pereiaslavska S. O. ANALYSIS OF THE IMPACT OF CONTAINERIZATION ON THE SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) IN THE DEVOPS PARADIGM

The article is devoted to the study of the impact of containerization technology on the transformation of the software development life cycle (SDLC) within the context of the DevOps methodology, including an analysis of the mechanisms for integrating container platforms into continuous integration and continuous delivery processes. For the first time, the impact of containerization on all phases of the SDLC has been systematized through the lens of DevOps practices, including the development of a comprehensive model for integrating container technologies into automated pipelines. A classification of security controls for containerized microservices at different stages of the life cycle is proposed. Metrics for evaluating the effectiveness of container solutions in enterprise environments have been defined. The specifics of Kubernetes orchestration have been taken into account. It has been established that, compared to traditional virtualized approaches, containerization reduces application deployment time in large-scale cloud systems by 60–75 percent. It has been found that container environments have security vulnerabilities, with critical issues arising particularly during the image-building process. It is emphasized that orchestration also introduces security risks. A structural model has been developed. Docker and Kubernetes have been integrated into DevSecOps pipelines. Security controls have been automated at all stages. It has been demonstrated that cloud automation is evolving, and classical DevOps is undergoing transformation. Particular attention is paid to the construction of autonomous container-based infrastructure. It is noted that scaling containerized microservices requires optimization; accordingly, cloud registries provide best practices.

Within the framework of this study, it has been established that containerization transforms the software development life cycle, leading to the unification of development, production, and testing environments. The necessity of specialized security control tools when implementing container technologies in DevOps, as well as automated monitoring, has been substantiated. A multi-layered security model has been proposed to illustrate a comprehensive approach to securing container environments. It has been determined that Kubernetes, as an orchestration platform, ensures application scalability.

Keywords: *containerization, Docker, Kubernetes, DevOps, software development lifecycle.*

Дата першого надходження статті до видання: 17.02.2026
Дата прийняття статті до друку після рецензування: 11.03.2026
Дата публікації (оприлюднення) статті 11.05.2026