Вісник ЛДУБЖД
Bulletin of Lviv State University of Life Safety
https://journal.ldubgd.edu.ua/index.php/Visnuk
ISSN 2078-4643 (print), ISSN 2708-1389 (online)

UDC 004.415:004.75:378.147
DOI: 10.32447/20784643.33.2026.09

This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution 4.0 (CC BY 4.0) License (http://creativecommons.org/licenses/by/4.0/).

В. Є. Дзень, Ю. О. Борзов
Львівський державний університет безпеки життєдіяльності, м. Львів, Україна
ORCID: https://orcid.org/0000-0001-6546-0233 – В. Є. Дзень
https://orcid.org/0000-0002-0604-0498 – Ю. О. Борзов
vitaliy.dzen.303@gmail.com

АРХІТЕКТУРА ДЕЦЕНТРАЛІЗОВАНОГО СЕРЕДОВИЩА БЕЗПЕРЕРВНОЇ ІНТЕГРАЦІЇ ДЛЯ ОСВІТНІХ ІТ-ПРОЄКТІВ

Проблема. Впровадження практик безперервної інтеграції та розгортання (CI/CD) є важливою вимогою до сучасних освітніх ІТ-проєктів. Однак самостійна імплементація таких рішень, наприклад, розгортання з нульовим часом простою, вимагає глибокої інженерної експертизи. Більше того, ручне налаштування професійної інфраструктури потребує значних витрат часу, а людський фактор та складність конфігурації є одними з головних ризиків і невдач під час впровадження DevOps-практик. У результаті здобувачі освіти змушені зміщувати фокус із реалізації власної ідеї та розробки бізнес-логіки проєкту на вирішення складних інфраструктурних задач, що може збільшувати загальний час роботи над проєктом у кілька разів. Відсутність готових, адаптованих інфраструктурних рішень створює бар'єр, який суттєво ускладнює перехід студентських систем від локального коду до стадії стабільного релізу.

Мета. Розробка та наукове обґрунтування архітектурної моделі децентралізованого середовища безперервної інтеграції Automated CI/CD Lifecycle Orchestrator, здатного забезпечити портативність, безпечну ізоляцію обчислювальних ресурсів та автоматизовану імплементацію стратегії Blue-Green розгортання для освітніх ІТ-проєктів без необхідності залучення складних централізованих кластерів.

Методи дослідження. В основу дослідження покладено принципи децентралізації обчислювальних систем та концепцію декларативного управління Infrastructure as Code. Для ОС-рівневої віртуалізації та ізоляції середовищ застосовано технологію Docker. Управління життєвим циклом розгортання реалізовано на базі Jenkins із використанням підходу JCasC (Jenkins Configuration as Code). Для оцінки ефективності запропонованої моделі застосовано методи логічного моделювання розподілу мережевого трафіка та емпіричного вимірювання показників доступності системи під час релізу.

Основні результати дослідження. Спроєктовано портативний інфраструктурний шаблон (Starter Kit), який трансформує імперативний підхід до розгортання у декларативний, переносячи CI/CD конвеєр безпосередньо у простір індивідуального проєкту. Розроблено та імплементовано алгоритм локального Blue-Green розгортання з автоматизованою валідацією стабільності релізу (health-checks). Емпіричний порівняльний аналіз довів, що використання запропонованої архітектури радикально скорочує час первинної ініціалізації середовища та мінімізує кількість ручних кроків, ліквідуючи ризики людського фактора. Практична апробація (навантажувальне тестування) підтвердила досягнення абсолютної відмовостійкості: 0 мс простою як при успішному оновленні версій, так і при автоматичному скасуванні транзакції (Rollback) у разі спроби розгортання дефектного коду.

Висновки та конкретні пропозиції авторів. Запропонована архітектура децентралізованого середовища ефективно вирішує проблему безпечного розгортання та забезпечення відмовостійкості студентських ІТ-проєктів. Застосування створеного інфраструктурного рушія Automated CI/CD Lifecycle Orchestrator значно знижує поріг входу до використання складних Enterprise-патернів, дозволяючи розробникам фокусуватися виключно на бізнес-логіці додатка. Автори пропонують впроваджувати цей інфраструктурний шаблон як базовий стандартизований інструмент для супроводу життєвого циклу проєктів у закладах вищої освіти.

Ключові слова: smart-системи, безперервна інтеграція, Blue-Green розгортання, Infrastructure as Code, децентралізована інфраструктура, контейнеризація, освітні ІТ-проєкти, заклади вищої освіти.

V. Ye. Dzen, Yu. O. Borzov
Lviv State University of Life Safety, Lviv, Ukraine

ARCHITECTURE OF A DECENTRALIZED CONTINUOUS INTEGRATION ENVIRONMENT FOR EDUCATIONAL IT PROJECTS

Introduction. The implementation of continuous integration and continuous deployment (CI/CD) practices is an important requirement for modern educational IT projects. However, the independent implementation of such solutions, for example, zero-downtime deployment, requires profound engineering expertise. Moreover, the manual configuration of professional infrastructure requires a substantial amount of time, and the human factor, along with configuration complexity, are among the main causes of risks and failures when adopting DevOps practices. As a result, students are forced to shift their focus from realizing their own ideas and developing the project's business logic to solving complex infrastructural tasks, which can significantly increase the overall project development time. The lack of ready-made, adapted infrastructural solutions creates a barrier that severely complicates the transition of student systems from local code to a stable release stage.

Purpose. The study aims to design and scientifically validate the architectural model of a decentralized continuous integration environment, the "Automated CI/CD Lifecycle Orchestrator." This system is designed to provide portability, secure isolation of computing resources, and automated implementation of the Blue-Green deployment strategy for educational IT projects, without the need for complex centralized clusters.

Methods. The research is based on the principles of decentralized computing systems and the concept of declarative infrastructure management "Infrastructure as Code". Operating system-level virtualization and environment isolation are achieved using Docker technology. Deployment lifecycle management is implemented using Jenkins with a JCasC (Jenkins Configuration as Code) approach. To evaluate the efficiency of the proposed model, methods of logical modeling of network traffic distribution and empirical measurement of system availability metrics during the release were applied.

Results. A portable infrastructure template (Starter Kit) was designed, which shifts the classical imperative deployment approach into a declarative one, moving the CI/CD pipeline directly into the individual project's space. A local Blue-Green deployment algorithm with automated release stability validation (health-checks) was developed and implemented. Empirical comparative analysis proved that using the proposed architecture radically reduces the initial setup time of the environment and minimizes the number of manual steps, eliminating human error risks. Practical load testing confirmed the achievement of absolute fault tolerance: 0 ms downtime both during successful version updates and during automatic transaction cancellation (Rollback) in case of an attempt to deploy defective code.

Conclusions. The proposed architecture of the decentralized environment effectively solves the problem of secure deployment and fault tolerance in student IT projects. Implementing the created infrastructure engine "Automated CI/CD Lifecycle Orchestrator" significantly lowers the entry barrier to utilizing complex Enterprise patterns, enabling developers to focus exclusively on the application's business logic. The authors propose integrating this infrastructure template as a basic standardized tool to support the lifecycle of projects in higher education institutions.

Keywords: smart systems, continuous integration, Blue-Green deployment, Infrastructure as Code, decentralized infrastructure, containerization, educational IT projects, higher education institutions.

Вступ. Еволюція архітектурних патернів у сучасній інженерії програмного забезпечення зумовлює глобальний перехід індустрії до складних моделей безперервної інтеграції та безперервного розгортання CI/CD [3]. Сучасна культура розробки Enterprise-рішень базується на принципах, де автоматизована доставка коду стає непомітною для користувача. Технологічні лідери впроваджують механізми, що гарантують абсолютну стабільність сервісу під час оновлень, поєднуючи технічну досконалість із глибокою відповідальністю за якість кінцевого продукту [4]. Однак проєктування та налаштування таких інфраструктурних рішень залишається високотехнологічним завданням, що потребує ґрунтовної теоретичної бази та реального практичного досвіду у сфері DevOps-інженерії. Як правило, розробники здобувають подібні компетенції вже на зрілих етапах професійної кар'єри, переймаючи досвід під час роботи над масштабними комерційними системами або завдяки взаємодії з колегами-інженерами, що мають багаторічний практичний стаж. Сучасні студентські IT-проєкти стають дедалі складнішими, часто охоплюючи розробку ресурсоємних smart-систем, таких як алгоритми аналізу щільності відеопотоків [20] або адаптивної сегментації рухомих об'єктів у реальному часі [21]. Забезпечення безперервного функціонування та тестування подібних застосунків вимагає суворої ізоляції та стабільної інфраструктурної бази. Однак в умовах створення таких проєктів самостійна імплементація надійних інфраструктурних практик стає серйозною перешкодою. Як свідчать дослідження, ручне налаштування професійної інфраструктури забирає багато часу, а людський фактор та складність конфігурації є одними з головних причин ризиків і невдач при впровадженні DevOps-практик [1, 2]. Замість концентрації на інноваційній алгоритмічній складовій, дослідженнях та бізнес-логіці своїх

систем, здобувачі освіти змушені витрачати значні обсяги часу на розв'язання проблем розгортання.

Заклади вищої освіти активно підтримують ініціативи здобувачів, надаючи необхідні обчислювальні ресурси для реалізації їхніх IT-проєктів – виділені віртуальні машини або базові операційні середовища, наприклад, чисті дистрибутиви Ubuntu чи CentOS. Така практика є вкрай позитивною та важливою, оскільки вона забезпечує студентів реальною інфраструктурою для роботи та тестування. Проте, отримавши доступ до базового сервера, розробники стикаються з необхідністю самостійно налаштовувати середовище виконання та вибудовувати процеси розгортання з нуля. Цей етап вимагає від студентів виконання невластивої їм на даному етапі ролі системних адміністраторів та DevOps-інженерів. Самостійне опанування інструментів конфігурації мережевих шлюзів, налаштування безпеки, розв'язання конфліктів залежностей та побудова конвеєрів безперервної інтеграції потребує специфічних індустріальних навичок, а також великих витрат часу та зусиль. У результаті, замість того щоб фокусуватися на розробці інноваційної бізнес-логіки та вдосконаленні архітектури самого додатка, студенти змушені витрачати свій ресурс на подолання інфраструктурних бар'єрів. Це робить повноцінну імплементацію професійних рішень, зокрема, автоматизованого Blue-Green розгортання, надзвичайно складною задачею без наявності попередньо підготовлених, стандартизованих інфраструктурних шаблонів.

Аналіз останніх досліджень і публікацій. Питання інтеграції DevOps-практик в освітній процес та пов'язані з цим дидактичні виклики активно досліджуються сучасною науковою спільнотою [12]. Аналіз альтернативних підходів показує, що у низці праць [11, 12] пропонується використання SaaS-платформ, таких як GitLab CI або GitHub Actions, для автоматизації перевірки студентського коду. Хоча ці рішення полегшують адміністрування інфраструктури, вони створюють залежність від зовнішніх провайдерів, які часто обмежуються лімітами безкоштовних тарифних планів (build minutes) та, що найважливіше, приховують від студентів низькорівневі процеси взаємодії між CI-конвеєром і сервером розгортання, перетворюючи інфраструктуру на "чорний ящик". Інші автори [8] пропонують імплементацію сучасних оркестраційних рішень для управління навчальними середовищами. Проте використання публічних хмарних CI-сервісів часто обмежується лімітами безкоштовних тарифів та не дає студентам повного розуміння внутрішньої роботи інфраструктури. З іншого боку, імплементація повноцінних оркестраційних кластерів, наприклад Kubernetes, найчастіше є архітектурно надлишковою для масштабів освітніх IT-проєктів. Оскільки ця технологія першочергово проєктувалася для управління великими, високонавантаженими та структурно складними мікросервісними Enterprise-системами, об'єктивної необхідності в її використанні для студентських завдань немає. Більше того, розгортання та підтримка подібних інструментів або єдиних централізованих Jenkins-серверів вимагає надзвичайно високої експертизи від викладацького складу, зумовлює значні фінансові витрати на апаратне забезпечення та значно ускладнює безпечну ізоляцію індивідуальних середовищ. Як показують сучасні дослідження, ручне написання та конфігурація інфраструктурних скриптів без належної автоматизації часто супроводжується критичними помилками [9], а їх зростаюча складність перешкоджає швидкому розумінню коду розробниками [5, 6]. Тому для забезпечення безпечної ізоляції навчальних проєктів доцільніше застосовувати методи ОС-рівневої віртуалізації на базі технології Docker, яка є стандартом для побудови ресурсоощадних та безпечних хмарних інфраструктур [7, 15, 16].

Попри наявність потужних інструментів автоматизації, відкритим залишається питання створення портативних, безпечних та ресурсоефективних інфраструктурних рішень, спеціально адаптованих для освітніх цілей. Проведені дослідження недостатньо уваги приділяли децентралізованим моделям розгортання, де інфраструктура постачається разом із кодом проєкту у вигляді автономного вузла. Зокрема, невирішеною залишається проблема розробки стандартизованих інфраструктурних шаблонів Starter Kits, які б дозволяли реалізовувати складні стратегії розгортання такі, як Blue-Green deployment локально або на базових віртуальних машинах без залучення складних централізованих кластерів, забезпечуючи при цьому строгу ізоляцію середовищ та професійні стандарти надійності під час релізу.

Методи дослідження. Для вирішення поставленої проблеми запропоновано комплексний підхід, що базується на архітектурному проєктуванні децентралізованих систем, методах контейнеризації та алгоритмах управління розподіленим мережевим трафіком.

Для наближення студентських IT-проєктів до Enterprise-стандартів розроблено багаторівневу екосистему Smart University Observability & Automation Hub, яка забезпечує супровід проєкту на всіх етапах життєвого циклу. Архітектура

системи складається з трьох ключових інтегрованих модулів:

Smart-Observability SDK – автономний телеметричний агент, розроблений з урахуванням сучасних парадигм спостережуваності (observability) хмарних систем [18], що вбудовується на рівні вихідного коду та автоматично експортує метрики середовища виконання, генерує JSON-логи та актуалізує OpenAPI-специфікації без необхідності ручного налаштування.

Centralized Operations Core – уніфікований хаб моніторингу та знань, який виконує агрегацію телеметрії з усіх вузлів системи (стек Prometheus, Loki, Grafana) та автоматизовано формує технічну документацію.

Lifecycle Orchestrator – рушій інфраструктурного розгортання, який автоматизує динамічну переконфігурацію середовища та відповідає за безперервну доставку коду.

Оскільки забезпечення прозорості та спостережуваності потребує надійної інфраструктурної бази, у межах цього дослідження детально розглянуто архітектуру та механізми функціонування третього модуля – Automated CI/CD Lifecycle Orchestrator.

Архітектура децентралізованого вузла розгортання. На противагу традиційній моделі використання єдиного центрального Jenkins-сервера університету, запропоновано концепцію портативного інфраструктурного шаблону Starter Kit. Ця модель реалізує децентралізоване IaC-середовище, де кожен студентський проєкт містить власний ізольований контур розгортання.

Технічна реалізація вузла базується на оркестрації контейнерів за допомогою Docker Compose, що дозволяє використовувати механізми ядра операційної системи cgroups та namespaces для суворої ізоляції обчислювальних ресурсів, гарантуючи високу доступність та масштабованість при розробці хмаро орієнтованих (cloud-native) застосунків [17]. Ядром пайплайну виступає локальний інстанс Jenkins, конфігурація якого повністю автоматизована за допомогою плагіна JCasC (Jenkins Configuration as Code) [13]. Jenkinsfile містить декларативний опис кроків збірки, тестування та розгортання. Мережева маршрутизація та роль балансувальника навантаження делегована Nginx, який динамічно переконфігуровується в процесі оновлення версій сервісу.

Візуалізацію запропонованої архітектурної моделі децентралізованого інфраструктурного рушія наведено на рис. 1.

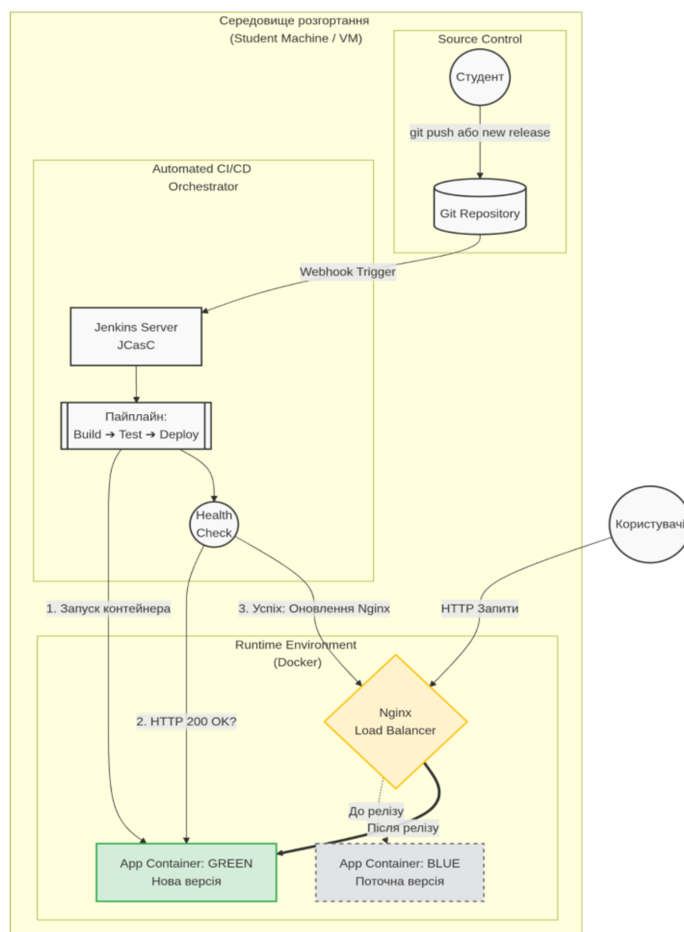

Рисунок 1 – Архітектурна модель децентралізованого інфраструктурного рушія з підтримкою автоматизованого Blue-Green розгортання

Як видно зі схеми, процес інкапсулює всі необхідні компоненти всередині єдиного локального середовища виконання VM (virtual machine). Життєвий цикл релізу ініціюється розробником безпосередньо через систему контролю версій, що генерує тригерну подію Webhook для оркестратора Jenkins. Автоматизований пайплайн виконує збірку та тестування, після чого в ізольованому Runtime-середовищі розгортається новий контейнер Green паралельно з поточною активною версією Blue. Ключовим вузлом безпеки є модуль Health Check: лише після підтвердження безпомилкового статусу нового контейнера оркестратор подає команду балансувальнику Nginx на перемикання трафіка.

Алгоритмічна та математична модель Blue-Green розгортання. Ключовою вимогою до сучасних Full-Stack та Enterprise-систем є оновлення версій без переривання обслуговування користувачів (Zero-Downtime Deployments) [19]. Для реалізації цього механізму в умовах обмежених ресурсів розроблено алгоритм локального Blue-Green розгортання.

Нехай система перебуває у просторі станів розгортання

$S = \{V_{blue}, V_{green}\}$, де V_{blue} – поточна стабільна версія додатка, а V_{green} – нова версія реліз-кандидат. Мережевий шлюз Nginx виконує функцію балансувальника [14], розподіляючи вхідний трафік відповідно до вагових коефіцієнтів W_{blue}, та W_{green}, для яких справедливо:

$$W_{blue} + W_{green} = 1$$

У початковому стані, коли функціонує лише попередня версія, розподіл трафіка має вигляд (1, 0). Під час тригерування пайплайну CI/CD модуль оркестрації ініціює запуск контейнера з версією V_{green}. Для валідації успішності запуску вводиться булева функція стану працездатності інстансу $H(V_i) \in \{0, 1\}$. Модуль Jenkins автоматично здійснює запити до ендпоїнтів актуатора розгорнутого інстансу, або звертається до підсистеми Smart-Observability для визначення значення $H(V_{green})$.

Перемикання трафіка дозволяється виключно за умови проходження автоматизованої верифікації:

$$H(V_{green}) = 1$$

За виконання цієї умови модуль розгортання генерує нову конфігурацію для Nginx та ініціює атомарну операцію гарячого перезавантаження, яка математично описується як транзакція зміни вагових коефіцієнтів:

$$(W_{blue}, W_{green}) \rightarrow (0, 1)$$

Це рівняння формалізує принцип неподільності вхідного мережевого трафіка, сукупний обсяг якого становить 100%, що в еквіваленті часток дорівнює 1. Оскільки класична стратегія Blue-Green deployment не передбачає часткового балансування навантаження між двома версіями, на відміну від підходу Canary release, вагові коефіцієнти можуть набувати виключно дискретних булевих значень: $W_i \in \{0, 1\}$. Таким чином, рівняння доводить, що мережевий шлюз працює як строгий бінарний перемикач: трафік завжди повністю спрямовується лише на один активний контейнер, гарантуючи, що користувачі фізично не зможуть отримати доступ до фонового середовища оновлення до моменту повної верифікації нового релізу.

У разі, якщо $H(V_{green}) = 0$ – критичний збій під час ініціалізації або провальне проходження health-check, транзакція скасовується, трафік продовжує надходити до V_{blue}, а контейнер V_{green} примусово зупиняється та видаляється, що гарантує збереження працездатності системи під час виконання (runtime).

Результати дослідження. Практична імплементація розробленої архітектурної моделі децентралізованого інфраструктурного рушія Automated CI/CD Lifecycle Orchestrator дозволила отримати низку кількісних та якісних результатів, що підтверджують ефективність запропонованого підходу порівняно з класичними ручними рішеннями.

Завдяки використанню технології контейнеризації на базі Docker та опису інфраструктури як коду було створено стандартизований інфраструктурний шаблон Starter Kit. Цей шаблон дозволяє розгорнути повноцінний CI/CD конвеєр, що включає локальний сервер автоматизації Jenkins та мережевий шлюз Nginx, безпосередньо в середовищі розробки студента. Це вирішило проблему ізоляції обчислювальних ресурсів, усунуло ризики компрометації облікових даних між різними проєктами та децентралізувало навантаження.

Перехід від імперативного адміністрування до декларативного підходу дозволив суттєво знизити поріг входу для розробників. Автоматизація життєвого циклу релізу ліквідувала необхідність ручного підключення до серверів, управління змінними середовища, оновлення конфігурації та створення бекапів. Порівняльний аналіз операційної складності у таблиці 1 демонструє радикальне скорочення часових витрат та усунення ризиків, пов'язаних із людським фактором.

Таблиця 1

Порівняльний аналіз операційної складності розгортання

Критерій порівняння	Традиційний підхід	Запропонована модель
Спосіб конфігурації інфраструктури	Імперативний (покрокове встановлення ПЗ, ручне налаштування мереж)	Декларативний (автоматизований запуск через docker-compose та JCasC)
Час первинної ініціалізації середовища	2–10 години (пошук туторіалів, встановлення правильних версій ПЗ на сервер, вирішення конфліктів портів, налаштування доступу)	1-4 годин (інтеграція готового портативного шаблону у свій проєкт та адаптація конфігурації за інструкцією)
Кількість ручних кроків для випуску одного релізу	**11 кроків:** Перемикання на гілку з останньою версією, локальна збірка, підготовка нової конфігурації, SSH підключення до сервера , завантаження нової версії через SCP/FTP, створення бекапів конфігурації, завантаження або оновлення нової конфігурації на сервері, створення бекапів старої версії застосунку, ручна зупинка старої версії, запуск нової версії додатку із використанням нової конфігурації, перевірка логів та працездатності	**1 крок**: Створення нової версії релізу або git push (залежно від побажань та налаштувань розробника сервісу)
Час активного залучення розробника на один реліз	15-40 хвилин (фокус на інфраструктурних задачах)	1 хвилина
Схильність до людських помилок	Висока - ризик хибної конфігурації, пропуску кроків	Практично відсутня - процес детермінований
Реакція на критичний збій релізу	Потребує ручного відновлення з бекапів, що були створені на попередніх кроках. Проєкт залишається недоступним (Downtime 100%) протягом усього часу відновлення.	Автоматичне скасування транзакції перемикання трафіка при виявленні помилки, миттєвий Rollback (Downtime 0%)

Для перевірки ефективності алгоритму послідовного перемикання трафіка за стратегією Blue-Green було проведено серію навантажувальних тестувань порівняно з традиційним підходом. Експеримент проводився у контрольованому ізольованому середовищі на базі віртуальної машини з характеристиками (2 vCPU, 4 GB RAM, ОС Ubuntu 22.04 LTS). З метою імітації реальних умов експлуатації генерувався постійний потік вхідних HTTP-запитів за допомогою інструменту навантажувального тестування Apache JMeter. Тривалість кожної тестової сесії становила 10-15 хвилин із частотою 10-20 запитів на секунду. Оцінка «Zero Downtime» (безперервна доступність) фіксувалася на прикладному рівні: критерієм успіху була відсутність відкинутих з'єднань або помилок HTTP 5xx під час атомарного перезавантаження конфігурації Nginx.

Під час використання традиційного підходу (рис. 2) процес оновлення версії супроводжувався повною зупинкою сервісу на час копіювання файлів та перезапуском додатка, що призвело до тривалого обриву з'єднань, падіння статусу запитів до нуля. Натомість використання запропонованої моделі (рис. 3) продемонструвало, що в момент атомарного перемикання конфігурації Nginx потік успішних відповідей (HTTP 200) залишався стабільним та безперервним. Це емпірично доводить досягнення показника часу простою на рівні 0 мс.

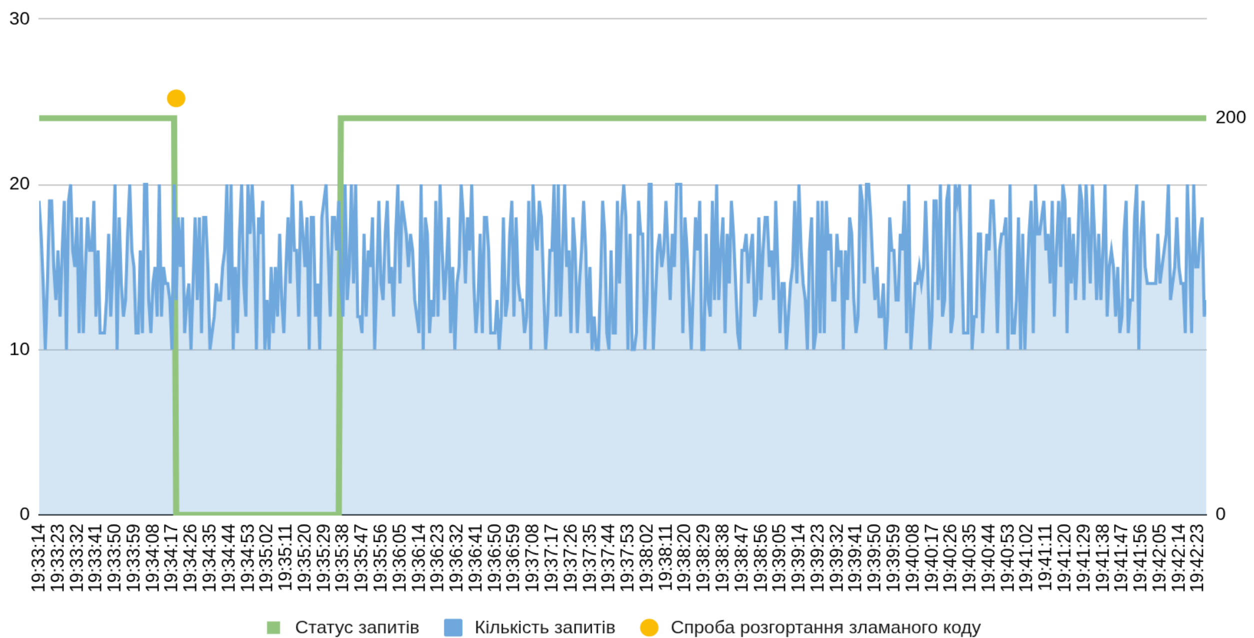

Рисунок 2 – Динаміка обробки HTTP-запитів під час перемикання трафіка з використанням традиційного підходу

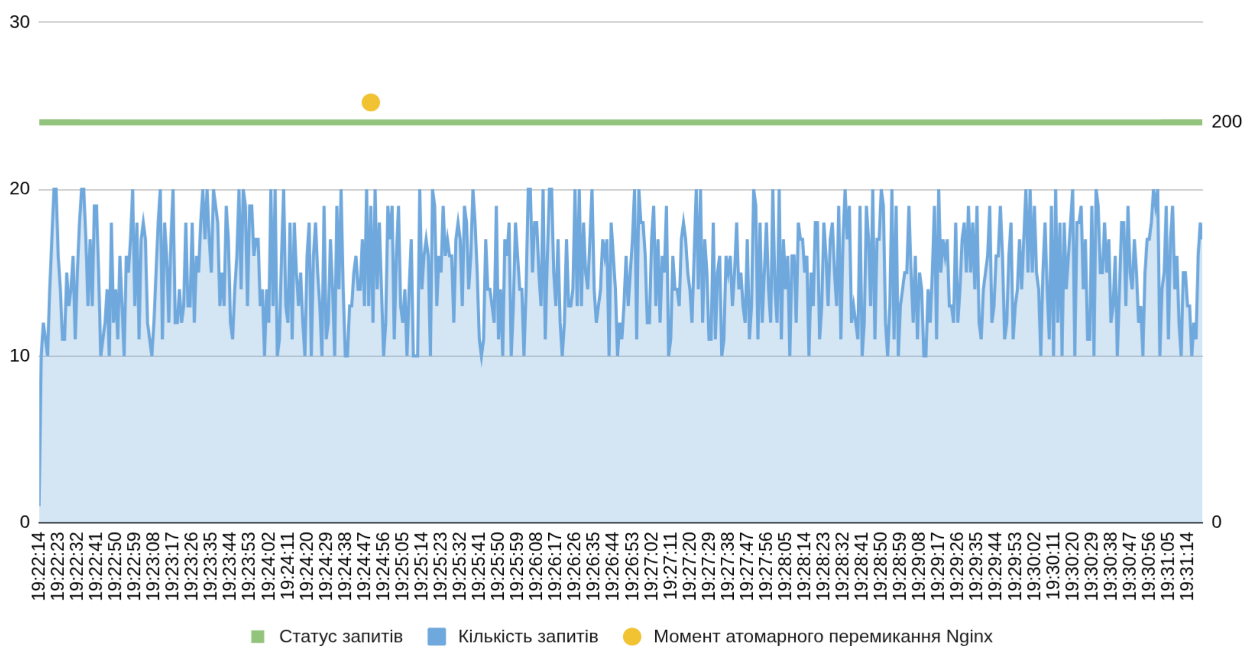

Рисунок 3 – Динаміка обробки HTTP-запитів під час перемикання трафіка з використанням запропонованої моделі

Ключовою вимогою до корпоративних Enterprise-систем є здатність протистояти розгортанню дефектних релізів. Для перевірки захисних механізмів було змодельовано розгортання критично нестабільного коду, який призводить до поломки додатка при запуску. Аналіз результатів тестування традиційного підходу (рис. 4) показує, що запуск дефектного коду на сервері призвів до негайного падіння системи, і сервіс залишався недоступним для користувачів аж до моменту ручного втручання та відновлення попередньої версії з бекапу.

Застосування розробленої архітектури (рис. 5) демонструє кардинально інший результат: модуль оркестрації успішно виявив негативний статус “здоров'я” нового контейнера ще до перемикання трафіка. Транзакцію було автоматично заблоковано, а кінцеві користувачі не відчули деградації сервісу – система зберегла 100% доступність, продовжуючи маршрутизувати запити на стабільний контейнер.

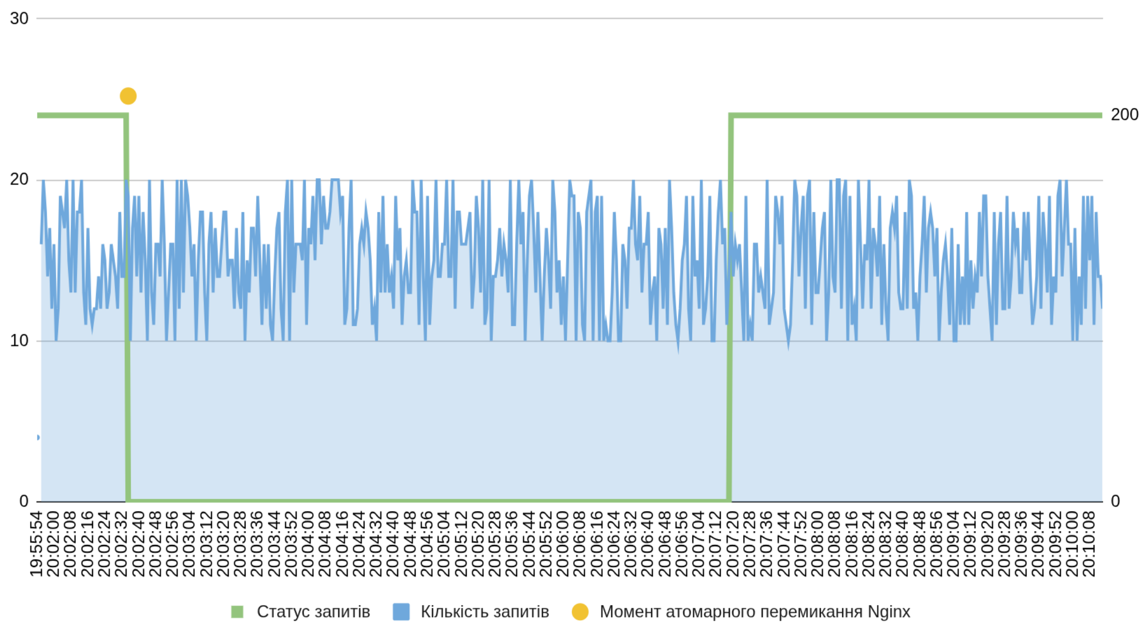

Рисунок 4 – Динаміка обробки HTTP-запитів під час імітації розгортання дефектного релізу з використанням традиційного підходу

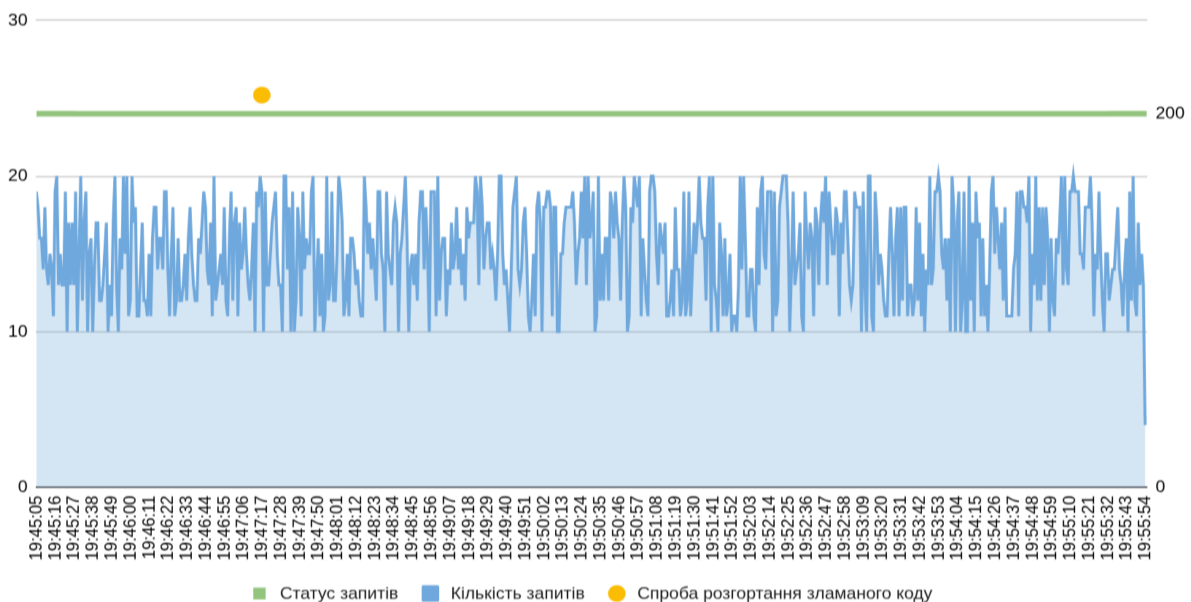

Рисунок 5 – Динаміка обробки HTTP-запитів під час імітації розгортання дефектного релізу з використанням запропонованої моделі

Незважаючи на доведену ефективність, запропонована архітектура має певні обмеження. Децентралізована модель оптимізована насамперед для монолітних або базових мікросервісних студентських проєктів. У разі горизонтального масштабування до складних багатосервісних систем із десятками залежних вузлів використання локального Docker Compose стає недостатнім, що вимагатиме переходу до централізованих оркестраторів рівня Kubernetes. Крім того, інкапсуляція CI/CD конвеєра та рантайм-середовища на одній віртуальній машині висуває підвищені вимоги до безпеки ОС-рівневої ізоляції, щоб уникнути вичерпання апаратних ресурсів під час одночасної збірки (build) та обробки користувацьких запитів.

Висновки. У статті розв'язано актуальну науково-практичну задачу – подолання розриву між базовими академічними підходами до розгортання програмного забезпечення та сучасними Enterprise-стандартами DevOps/SRE в межах освітнього процесу. Для цього запропоновано та успішно імплементовано децентралізовану архітектурну модель інфраструктурного рушія Automated CI/CD Lifecycle Orchestrator, реалізовану у вигляді портативного IaC-шаблону Starter Kit.

Емпіричні дослідження та порівняльний аналіз довели беззаперечну перевагу розробленої системи над традиційними ручними методами адміністрування. Перехід до декларативного підходу дозволив суттєво спростити та скоротити час первинної ініціалізації ізольованого середовища. Крім того, автоматизація CI/CD конвеєра зменшила операційну складність випуску релізу з 11 послідовних ручних кроків до єдиної тригерної події у системі контролю версій. Це дозволяє практично повністю нівелювати ризики критичних збоїв, пов'язаних із людським фактором.

Результати навантажувального тестування підтвердили високу надійність імплементованого алгоритму локального Blue-Green розгортання. На відміну від класичного підходу, що неминуче супроводжується відмовою в обслуговуванні під час оновлення файлів, запропонований оркестратор забезпечує безшовне атомарне перемикання мережевого трафіка (Downtime = 0 мс). Більше того, система продемонструвала абсолютну стійкість до розгортання дефектного коду: автоматизована перевірка стану працездатності контейнера успішно блокує нестабільні транзакції та миттєво виконує автоматичне повернення до попередньої працездатної версії, зберігаючи 100% доступність сервісу для кінцевих користувачів, що є ключовим стандартом для сучасних розподілених сервісів [10].

Практичне впровадження розробленого Starter Kit у навчальний процес закладів вищої освіти значно знижує поріг входу студентів до використання складних архітектурних патернів. Це надає майбутнім інженерам можливість фокусуватися виключно на розробці бізнес-логіки систем, працюючи у безпечному та повністю автоматизованому середовищі.

Перспективою подальших досліджень у даному напрямку є розширення екосистеми шляхом глибокої інтеграції розробленого оркестратора з телеметричними агентами Smart-Observability SDK. Це дозволить реалізувати механізм автоматичного відкату поточного інстансу до останньої стабільної версії на основі аналізу метрик у реальному часі в разі виявлення критичних збоїв в процесі безперервного функціонування.

Список літератури:

1. Pérez-Sánchez J., Rafi S., Carrillo de Gea J. M., Ros J. N., Fernández Alemán J. L. A theory on human factors in DevOps adoption. Computer Standards & Interfaces. 2025. Vol. 92. P. 103907. URL: https://www.sciencedirect.com/science/article/pii/S092054892400076X.

2. Kumar A., Nadeem M., Shameem M. Metaheuristic-based cost-effective predictive modeling for DevOps project success. Applied Soft Computing. 2024. Vol. 163. P. 111834. DOI: https://doi.org/10.1016/j.asoc.2024.111834.

3. Grande R., Vizcaíno A., García F. O. Is it worth adopting DevOps practices in Global Software Engineering? Possible challenges and benefits. Computer Standards & Interfaces. 2024. Vol. 87. P. 103767. DOI: https://doi.org/10.1016/j.csi.2023.103767.

4. Port D., Taber B., Emkani P. Investigating effectiveness and compliance to DevOps policies and practices for managing productivity and quality variability. Journal of Systems and Software. 2024. Vol. 213. P. 112030. DOI: https://doi.org/10.1016/j.jss.2024.112030.

5. Bessghaier N., Ouni A., Sayagh M., Mkaouer M. W. A search-based file recommendation approach for infrastructure-as-code evolution. Journal of Systems and Software. 2026. Vol. 234. P. 112746. DOI: https://doi.org/10.1016/j.jss.2025.112746.

6. Quéval P.-J., Hörner N. E., Ntentos E., Zdun U. On the understandability of coupling-related practices in infrastructure-as-code based deployments. Information and Software Technology. 2025. Vol. 185. P. 107761. DOI: https://doi.org/10.1016/j.infsof.2025.107761.

7. De Benedictis M., Lioy A. Integrity verification of Docker containers for a lightweight cloud environment. Future Generation Computer Systems. 2019. Vol. 97. P. 236–246. DOI: https://doi.org/10.1016/j.future.2019.02.026.

8. Sarmiento, E., Leite, J., & Aliaga, A. H. (2024). Introducing Computer Science Undergraduate Students to DevOps Technologies from Software Engineering Fundamentals. ACM. https://doi.org/10.1145/3639474.3640071

9. Drosos G.-P., Sotiropoulos T., Alexopoulos G., Mitropoulos D., Su Z. When Your Infrastructure Is a Buggy Program: Understanding Faults in Infrastructure as Code Ecosystems. Proceedings of the ACM on Programming Languages. 2024. Vol. 8 (OOPSLA2). DOI: https://doi.org/10.1145/3689799.

10. Damera T. Zero-Downtime Migration Strategies for Large-Scale Distributed Services. International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 2025. Vol. 11. P. 179–187. DOI: https://doi.org/10.32628/CSEIT2511123934.

11. Grotta A., Prado E. P. V. DevOps Didactic Transposition in IS Higher Education: A Systematic Literature Review. AMCIS 2022 Proceedings. 2022. P. 15. URL: https://aisel.aisnet.org/amcis2022/sig_ed/sig_ed/15.

12. Garcia P. S. C. et al. Current DevOps Teaching Techniques: A Systematic Literature Review. Simpósio Brasileiro de Engenharia de Software (SBES). 2024. P. 389–398. DOI: https://doi.org/10.5753/sbes.2024.3503.

13. Jenkins Contributors. Jenkins Configuration as Code. Jenkins Documentation. URL: https://www.jenkins.io/projects/jcasc/.

14. F5 NGINX. HTTP Load Balancing. NGINX Documentation. URL: https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/.

15. Nüst D. et al. Ten simple rules for writing Dockerfiles for reproducible data science. PLoS Comput Biol. 2020. Vol. 16(11). P. e1008316. DOI: https://doi.org/10.1371/journal.pcbi.1008316.

16. Gomes D. A., Mestre P., Serôdio C. Measuring the Reproducibility of Scientific Research Based on Computing Environment Provisioning Methods (SDG). Journal of Lifestyle and SDGs Review. 2025. Vol. 5(2). P. e02313. DOI: https://doi.org/10.47172/2965-730X.SDGsReview.v5.n02.pe02313.

17. Nascimento B. et al. Availability, Scalability, and Security in the Migration from Container-Based to Cloud-Native Applications. Computers. 2024. Vol. 13(8). P. 192. DOI: https://doi.org/10.3390/computers13080192.

18. Kosińska J. et al. Toward the Observability of Cloud-Native Applications: The Overview of the State-of-the-Art. IEEE Access. 2023. Vol. 11. P. 73036–73052. DOI: https://doi.org/10.1109/ACCESS.2023.3281860.

19. Pappula K. K. Containerized Zero-Downtime Deployments in Full-Stack Systems. IJAIBDCMS. 2022. Vol. 3(4). P. 60–69. URL: https://ijaibdcms.org/index.php/ijaibdcms/article/view/233.

20. Peleshko D., Ivanov Y., Sharov B., Izonin I., Borzov Y. Design and Implementation of Visitors Queue Density Analysis and Registration Method for Retail Videosurveillance Purposes. Proceedings of the 2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP). 2016. P. 159–162. DOI: https://doi.org/10.1109/DSMP.2016.7583531.

21. Ivanov Y., Peleshko D., Makoveichuk O., Izonin I., Malets I., Lotoshynska N., Batyuk D. Adaptive moving object segmentation algorithms in cluttered environments. 2015 13th International Conference on The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM). 2015. P. 97–99. DOI: https://doi.org/10.1109/CADSM.2015.7230806.

References:

1. Pérez-Sánchez, J., Rafi, S., Carrillo de Gea, J. M., Ros, J. N., & Fernández Alemán, J. L. (2025). A theory on human factors in DevOps adoption. Computer Standards & Interfaces, 92, 103907. https://doi.org/10.1016/j.csi.2024.103907

2. Kumar, A., Nadeem, M., & Shameem, M. (2024). Metaheuristic-based cost-effective predictive modeling for DevOps project success. Applied Soft Computing, 163, 111834. https://doi.org/10.1016/j.asoc.2024.111834

3. Grande, R., Vizcaíno, A., & García, F. O. (2024). Is it worth adopting DevOps practices in Global Software Engineering? Possible challenges and benefits. Computer Standards & Interfaces, 87, 103767. https://doi.org/10.1016/j.csi.2023.103767

4. Port, D., Taber, B., & Emkani, P. (2024). Investigating effectiveness and compliance to DevOps policies and practices for managing productivity and quality variability. Journal of Systems and Software, 213, 112030. https://doi.org/10.1016/j.jss.2024.112030

5. Bessghaier, N., Ouni, A., Sayagh, M., & Mkaouer, M. W. (2026). A search-based file recommendation approach for infrastructure-as-code evolution. Journal of Systems and Software, 234, 112746. https://doi.org/10.1016/j.jss.2025.112746

6. Quéval, P.-J., Hörner, N. E., Ntentos, E., & Zdun, U. (2025). On the understandability of coupling-related practices in infrastructure-as-code based deployments. Information and Software Technology, 185, 107761. https://doi.org/10.1016/j.infsof.2025.107761

7. De Benedictis, M., & Lioy, A. (2019). Integrity verification of Docker containers for a lightweight cloud environment. Future Generation Computer Systems, 97, 236–246. https://doi.org/10.1016/j.future.2019.02.026

8. Sarmiento, E., Leite, J., & Aliaga, A. H. (2024). Introducing Computer Science Undergraduate Students to DevOps Technologies from Software Engineering Fundamentals. Proceedings of the 55th ACM Technical Symposium on Computer Science Education. https://doi.org/10.1145/3639474.3640071

9. Drosos, G.-P., Sotiropoulos, T., Alexopoulos, G., Mitropoulos, D., & Su, Z. (2024). When Your Infrastructure Is a Buggy Program: Understanding Faults in Infrastructure as Code Ecosystems. Proceedings of the ACM on Programming Languages, 8(OOPSLA2). https://doi.org/10.1145/3689799

10. Damera, T. (2025). Zero-Downtime Migration Strategies for Large-Scale Distributed Services. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 11, 179–187. https://doi.org/10.32628/CSEIT2511123934

11. Grotta, A., & Prado, E. P. V. (2022). DevOps Didactic Transposition in IS Higher Education: A Systematic Literature Review. AMCIS 2022 Proceedings, 15. https://aisel.aisnet.org/amcis2022/sig_ed/sig_ed/15

12. Garcia, P. S. C., Ferreira, J., Gonçalves, M., Carneiro, T., Figueiredo, E., & Pereira, I. M. (2024). Current DevOps Teaching Techniques: A Systematic Literature Review. Simpósio Brasileiro de

Engenharia de Software (SBES), 389–398. https://doi.org/10.5753/sbes.2024.3503

13. Jenkins Contributors. (n.d.). Jenkins Configuration as Code. Jenkins Documentation. Retrieved from https://www.jenkins.io/projects/jcasc/

14. F5 NGINX. (n.d.). HTTP Load Balancing. NGINX Documentation. Retrieved from https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/

15. Nüst, D., Sochat, V., Marwick, B., Eglen, S. J., Head, T., Hirst, T., & Evans, B. D. (2020). Ten simple rules for writing Dockerfiles for reproducible data science. PLoS Comput Biol, 16(11), e1008316. https://doi.org/10.1371/journal.pcbi.1008316

16. Gomes, D. A., Mestre, P., & Serôdio, C. (2025). Measuring the Reproducibility of Scientific Research Based on Computing Environment Provisioning Methods (SDG). Journal of Lifestyle and SDGs Review, 5(2), e02313. https://doi.org/10.47172/2965-730X.SDGsReview.v5.n02.pe02313

17. Nascimento, B., Santos, R., Henriques, J., Bernardo, M. V., & Caldeira, F. (2024). Availability, Scalability, and Security in the Migration from Container-Based to Cloud-Native Applications. Computers, 13(8), 192. https://doi.org/10.3390/computers13080192

18. Kosińska, J., Baliś, B., Konieczny, M., Malawski, M., & Zieliński, S. (2023). Toward the Observability of Cloud-Native Applications: The Overview of the State-of-the-Art. IEEE Access, 11, 73036–73052. https://doi.org/10.1109/ACCESS.2023.3281860

19. Pappula, K. K. (2022). Containerized Zero-Downtime Deployments in Full-Stack Systems. IJAIBDCMS, 3(4), 60–69. ijaibdcms.org/index.php/ijaibdcms/article/view/233

20. Peleshko, D., Ivanov, Y., Sharov, B., Izonin, I., & Borzov, Y. (2016). Design and Implementation of Visitors Queue Density Analysis and Registration Method for Retail Videosurveillance Purposes. Proceedings of the 2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP), 159–162. https://doi.org/10.1109/DSMP.2016.7583531

21. Ivanov, Y., Peleshko, D., Makoveichuk, O., Izonin, I., Malets, I., Lotoshynska, N., & Batyuk, D. (2015). Adaptive moving object segmentation algorithms in cluttered environments. 2015 13th International Conference on The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM), 97–99. https://doi.org/10.1109/CADSM.2015.7230806

© В. Є. Дзень, Ю. О. Борзов, 2026.
Науково-методична стаття.
Надійшла до редакції 10.03.2026.
Прийнята до друку 29.04.2026.
Опублікована 25.05.2026.