https://doi.org/10.31891/2219-9365-2025-84-1
УДК 004.451.8:621.865.8

ЗАГВОЙСЬКИЙ Ростислав
Національний Університет «Львівська політехніка»
https://orcid.org/0009-0005-2090-4255
e-mail: oleh.dmytriv.kn.2021@lpnu.ua
КАЗИМИРА Ірина
Національний Університет «Львівська політехніка»
https://orcid.org/0000-0003-1597-5647
e-mail: iryna.y.kazymyra@lpnu.ua
ТКАЧУК Катерина
Національний Університет «Львівська політехніка»
https://orcid.org/0009-0002-1788-8715
e-mail: kateryna.i.tkachuk@lpnu.ua
ДМИТРІВ Олег
Національний Університет «Львівська політехніка»
e-mail: oleh.dmytriv.kn.2021@lpnu.ua

МАСШТАБОВАНІ МЕХАНІЗМИ ВІДДАЛЕНОГО ОНОВЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В ПЕРИФЕРІЙНІЙ РОБОТОТЕХНІЦІ

Поширення розподілених роботизованих систем у промисловості створює потребу в безпечних і масштабованих механізмах віддаленого оновлення ПЗ, оскільки ручні процеси призводять до простоїв, розсинхронізації версій і затримок з випуском критичних оновлень. Запропонована архітектура поєднує контейнеризовані мікросервіси, CI/CD-конвеєри, Kubernetes-оркестрацію, GitOps-керування конфігураціями та спеціалізовані оператори передачі стану, забезпечуючи автоматизовані оновлення на різнорідних периферійних пристроях. Використання Docker із мультиархітектурою, IaC-інструментів (Ansible, TOSCA) та декларативних моделей оркестрування дозволяє адаптувати систему до обмежень edge-середовищ. Емпіричні результати демонструють передачу стану менш ніж за 42 мс для великих промислових контролерів [1] та 30-кратне прискорення пакетних оновлень порівняно з ручними процесами при мінімальному навантаженні на пристрої [2], що підтверджує ефективність і придатність описаних методів для сучасних виробничих систем.

Ключові слова: периферійна робототехніка; контейнеризація; CI/CD конвеєри; оркестрація Kubernetes; віддалене розгортання; інфраструктура як код (IaC); GitOps.

ZAHVOISKYI Rostyslav, KAZYMYRA Iryna, TKACHUK Kateryna, DMYTRIV Oleh
Lviv Polytechnic National University

SCALABLE MECHANISMS FOR REMOTE SOFTWARE UPDATES IN EDGE ROBOTICS

This article examines scalable mechanisms for remote software updates in edge robotics, focusing on automated, secure, and highly available delivery processes for heterogeneous industrial robotic systems. As distributed robots increasingly rely on frequent updates to ensure functional improvements, security patches, and model retraining, manual update procedures become insufficient due to downtime, version inconsistencies, and operational risks. The study proposes an integrated architecture combining containerized microservices, CI/CD pipelines, Kubernetes-based orchestration, GitOps configuration management, and state-transfer operators that enable seamless updates without interrupting control cycles. The approach leverages multi-architecture Docker images, Infrastructure-as-Code tools, declarative orchestration models, and robust security mechanisms, supporting reliable deployment across constrained edge environments. Experimental results demonstrate efficient state transfer under 42 ms for industrial controllers, up to a 30× acceleration of batch updates compared to manual procedures, and minimal resource overhead. The analysis synthesizes architectural patterns, update methodologies, and performance indicators relevant to large-scale robotic fleets, emphasizing reliability, security, automation, and observability as key requirements for industrial adoption. The proposed methodology provides a comprehensive foundation for designing scalable remote-update platforms that maintain continuous operation while ensuring transparency, fault tolerance, and rapid recovery. The findings highlight the practical feasibility of automated update pipelines in modern robotic DevOps ecosystems and outline future research directions, including differential updates, self-healing deployment strategies, lightweight orchestration frameworks, and ML-based anomaly detection for improving robustness in increasingly complex industrial environments.

Keywords: edge robotics; containerization; CI/CD pipelines; Kubernetes orchestration; remote deployment; Infrastructure as Code (IaC); GitOps.

Стаття надійшла до редакції / Received 30.09.2025
Прийнята до друку / Accepted 12.11.2025

АКТУАЛЬНІСТЬ ВІДДАЛЕНОГО АВТОМАТИЗОВАНОГО РОЗГОРТАННЯ В РОБОТИЗОВАНИХ СИТЕМАХ

Сучасна промислова автоматизація швидко переходить до розподілених роботизованих систем, що працюють на рівні пристроїв, периферійних та хмарних обчислень. Такі системи потребують частих оновлень програмного забезпечення для впровадження патчів безпеки, покращення моделей машинного навчання та

розширення функціональності, а темпи змін значно перевищують можливості традиційних ручних підходів [3]. Для підтримання узгодженості роботи великі географічно розподілені парки роботів мають отримувати оновлення синхронно, а критично важливі застосунки негайно реагувати на вразливості та помилки [4].

Ручні процеси оновлення погано масштабуються, вимагають значних людських ресурсів і створюють ризики розсинхронізації версій. Фізичний доступ до кожного пристрою або індивідуальні віддалені сеанси призводять до затримок і підвищують ймовірність помилок. Крім того, необхідність зупинки обладнання для застосування оновлень спричиняє простої та економічні втрати, а у випадку промислових контролерів навіть короткі переривання можуть зупинити всю виробничу лінію [5].

КОНТЕКСТ DEVOPS, IOT, ТА АВТОМАТИЗАЦІЇ В РОБОТИЗОВАНИХ СИСТЕМАХ

Практики DevOps, які спочатку були створені для хмарних систем, успішно адаптуються до IoT та робототехніки, забезпечуючи автоматизовані CI/CD-процеси, що прискорюють тестування, валідацію й розгортання програмних оновлень [6]. У розподіленому виробництві такі підходи підтримують індивідуальні збірки для гетерогенних платформ та автоматизують просування артефактів через етапи тестування, враховуючи операційні обмеження. Дослідження у сфері точного землеробства та промислового IoT показують скорочення часу розгортання з днів до хвилин і усунення розбіжностей версій завдяки DevOps-підходам [6], тоді як модельні рамки розгортання парку вирішують проблеми неоднорідності та обмеженого підключення [4].

Водночас інтеграція CI/CD у ресурсно обмежені робототехнічні системи супроводжується технічними викликами: різні архітектури процесорів (ARM, x86, RISC-V), нестабільні мережеві з’єднання та обмежені ресурси потребують використання багатоархітектурних контейнерних образів, легких стратегій контейнеризації та стійких методів оркестрації. Для зменшення цих бар’єрів з’являються локальні CI/CD-рішення для ROS 2, які забезпечують багатоархітектурні збірки та підтримують вимоги безпеки промислових систем [7].

ВИЗНАЧЕННЯ ПРОБЛЕМАТИКИ

Ключові технічні проблеми. Роботизовані системи у промисловості потребують безпечних, автоматизованих і масштабованих механізмів оновлення, оскільки ручні процеси створюють ризики для цілісності операцій, спричиняють розбіжності версій і вимагають простоїв, що може зупинити виробничі лінії та збільшити вразливість до кіберзагроз [5]. Критично важливо підтримувати оновлення без переривання циклів керування, забезпечувати безпеку протягом усього процесу розгортання та масштабувати систему відповідно до зростання кількості пристроїв. Додатковою складністю є неоднорідність парку обладнання: різні конфігурації апаратного забезпечення, операційні контексти й нестабільне підключення вимагають інтелектуальних механізмів призначення завдань і моделювання обмежень, щоб мінімізувати час оновлення та зберегти доступність у масштабі всього парку [4].

Обмеження середовища та операційної діяльності. Промислові виробничі середовища висувають жорсткі вимоги до систем оновлення, насамперед через нестабільність мережі. Перешкоди, перевантаження або велика відстань між пристроями спричиняють перебої у зв’язку та затримки централізованої координації [3]. Ускладнює ситуацію й ресурсна обмеженість периферійних робототехнічних платформ. Вбудовані контролери мають мінімальний обсяг пам’яті та сховища, що потребує легких контейнерних образів і багатоархітектурних конвеєрів для підтримки ARM, x86 та спеціалізованих прискорювачів.

Додатковий виклик створюють вимоги до безперервної роботи, оскільки критичні виробничі процеси не можуть бути зупинені навіть на короткий час. Контролери з циклічним управлінням повинні зберігати детерміновані затримки під час оновлення, що потребує складних механізмів зокрема динамічної передачі стану та безперервного перемикання, які дають змогу оновлювати програмне забезпечення в реальному часі без впливу на виробничі лінії [1].

Вихідні вимоги. Аналіз основної проблеми та екологічних обмежень дає комплексний набір функціональних та нефункціональних вимог до платформ автоматичного оновлення:

- **Надійність та доступність:** платформа повинна підтримувати безперебійні оновлення, що дозволяють уникнути перерв у виробництві. Для промислових контролерів механізми передачі стану повинні зберігати стан управління (включаючи великі набори змінних, що перевищують 100 000 елементів) та виконувати передачу в межах доступного часу циклу. Механізми відмовостійкості повинні виявляти невдалі розгортання та автоматично відновлювати стабільний стан пристроїв [1].
- **Безпека:** Комплексна безпека охоплює перевірку цілісності артефактів за допомогою криптографічного підпису, безпечні канали зв'язку для доставки оновлень (SSH, VPN, взаємний TLS), управління ідентифікацією та доступом для прийняття рішень щодо авторизації, а також комплексні аудиторські сліді для забезпечення відповідності та криміналістичного аналізу. Практики GitOps та декларативне управління конфігурацією підтримують безпеку, надаючи контрольовані за версіями та перевірені колегами визначення оновлень [8].

ISSN 2219-9365

- **Автоматизація:** Повна автоматизація процесу оновлення від фіксації коду до впровадження у виробництво виключає необхідність ручного втручання та пов'язані з цим ризики помилок. Процеси CI/CD повинні автоматично перевіряти програмне забезпечення за допомогою багатоетапного тестування (модульне, інтеграційне, симуляційне), створювати образи контейнерів з багаторівневою архітектурою, публікувати артефакти в реєстрах та запускати скоординовані впровадження на основі визначених політик [9].
- **Масштабованість:** архітектура повинна масштабуватися горизонтально для підтримки парків, що складаються з десятків або тисяч роботів. Масштабованість включає паралельне виконання оновлень на всіх пристроях, ефективний розподіл артефактів за допомогою стратегій спільного використання шарів і кешування, а також розподілену оркестрацію, що дозволяє уникнути централізованих вузьких місць. Модельні фреймворки розгортання парків забезпечують масштабовані механізми призначення, що враховують контекст і обмеження пристроїв [4].
- **Спостережуваність та відновлення системи:** Комплексна спостережуваність завдяки збиранню метрик, агрегації журналів та розподіленому відстеженню дозволяє операторам контролювати хід оновлення та діагностувати несправності. Декларативні механізми відкату повинні забезпечувати швидке повернення до відомих надійних конфігурацій, коли оновлення не проходять перевірку або виявляють операційні аномалії [10].

Цілі дослідження. Основною метою цієї статті є узагальнення та аналіз архітектурних моделей, стратегій реалізації та емпіричних даних щодо автоматизованих платформ віддаленого оновлення програмного забезпечення, адаптованих до промислових робототехнічних систем. Зокрема, ця робота має на меті надати дослідникам і практикам комплексну структуру для проектування, реалізації та валідації контейнерних механізмів оновлення, які відповідають суворим вимогам виробничих середовищ.

Для досягнення цієї мети в статті розглядаються чотири конкретні завдання:

1. **Архітектурний дизайн і таксономія:** Розробити систематичну таксономію методологій оновлення (контейнеризовані мікросервіси з оркеструванням, передача стану на основі оператора, прошивка OTA, декларативне управління GitOps/TOSCA) і синтезувати архітектурні шаблони з останніх промислових розгортань. Це включає ідентифікацію основних компонентів (CI/CD-конвеєри, реєстри артефактів, рівні оркестрування, агенти пристроїв) та їхні шаблони інтеграції [11].
2. **Вибір технології та порівняльний аналіз:** Аналіз компромісів між стратегіями контейнеризації, платформами оркестрування та інструментами автоматизації розгортання. Порівняльна оцінка вивчає характеристики продуктивності (затримка оновлення, накладні витрати на ресурси), оперативну складність та придатність для різних промислових контекстів. Цей аналіз базується на емпіричних вимірах з останніх публікацій, щоб надати рекомендації, засновані на фактичних даних [12].
3. **Стратегії впровадження Периферійних Роботизованих систем:** синтез найкращих практик контейнеризації в середовищах з обмеженими ресурсами, включаючи мінімальне створення образів, оптимізацію спільного використання шарів, підтримку багатоархітектурних систем та інтеграцію з вимогами до планування в режимі реального часу. Керівництво з впровадження стосується практичних питань, таких як управління конфігурацією, методології тестування та інтеграція з існуючими виробничими системами [13].
4. **Структура та показники оцінки:** Визначити комплексні критерії оцінки, що охоплюють функціональну правильність (успішне завершення оновлення, збереження стану), показники продуктивності (затримка оновлення, час передачі стану, затримки витягування зображень), ефективність використання ресурсів (споживання пропускної здатності, накладні витрати на обчислення) та економічний вплив (скорочення часу простою, економія робочої сили). Ця структура дозволяє проводити ретельну оцінку впровадження платформ оновлення [14].

ОСНОВНИЙ МАТЕРІАЛ ДОСЛІДЖЕННЯ

Концепти та пов’язані праці. Практики безперервної інтеграції та розгортання істотно вплинули на програмне постачання не лише в хмарних, а й у розподілених периферійних системах. У таких середовищах CD забезпечує індивідуальні оновлення для вузлів із різними умовами роботи, скорочуючи час впровадження з днів до хвилин та усуваючи проблеми з версіями [6]. Дані дослідження «Accelerate» демонструють значний ефект: частіші розгортання, коротший цикл доставки та нижчі показники невдач [13]. Розширення цих підходів на IoT-edge-cloud здійснюється через модельно-орієнтоване планування флоту, яке автоматично формує сценарії оновлень із урахуванням обмежень конкретних пристроїв. Перевірки в охороні здоров’я й промисловій автоматизації підтверджують ефективність такого підходу для керування складними багатопристроєвими розгортаннями [4].

Сучасні механізми оновлень у робототехніці охоплюють як легкі OTA-рішення для мікроконтролерів, так і повноцінне контейнерне оркестрування. OTA залишається критичним для пристроїв із мінімальними ресурсами, тоді як контейнеризація забезпечує вищу гнучкість, уніфікацію й автоматизацію. У робототехнічних системах спеціалізовані розширення Kubernetes KubeROS, RobotKube, RoboKube адаптують оркестрацію до потреб ROS 2, багатороботних конфігурацій і циклів від симуляції до виробництва

[2][15]. Експерименти на периферійних кластерах демонструють до 30-кратного прискорення пакетних оновлень порівняно з ручними процесами [2]. Для промислових контролерів, де неперервність роботи є обов'язковою, застосовують оператори передачі стану, які забезпечують міграцію програм керування без розриву циклів. Передача понад 100 000 змінних менш ніж за 42 мс підтверджує можливість оновлень без простоїв [1]. Комерційні OTA-платформи (Balena, Mender) корисні, але обмежені власницькими підходами та недостатньою підтримкою складних робототехнічних систем [16].

Підходи Infrastructure-as-Code та контейнеризація Docker формують основу відтворюваних оновлень, а багатоархітектурні образи покривають різнорідне обладнання; CI/CD-пайплайни ROS 2 вже підтримують ARM64, AMD64 і RISC-V [7]. Оркестрація Kubernetes забезпечує декларативне управління станом, автоматизовані оновлення та інтеграцію неконтейнеризованих компонентів через користувацькі оператори [8]. Ansible з ідемпотентними плейбуками надає гнучкі механізми як для первинного розгортання, так і для циклічних оновлень. Мова TOSCA доповнює екосистему моделями топологій і робочих процесів, а поєднання з GitOps забезпечує контрольовані, версіоновані та прозорі оновлення [8].

Архітектура системи. Комплексна платформа оновлення для промислових робототехнічних систем складається з п'яти основних архітектурних рівнів, кожен з яких вирішує конкретні завдання в процесі постачання програмного забезпечення:

- **Рівень розробки та контролю версій:** репозиторії Git слугують авторитетним джерелом для всіх програмних артефактів, визначень конфігурації та специфікацій інфраструктури. Стратегії розгалуження (GitFlow, розробка на основі стовбура) визначають робочі процеси просування від розробки через стадію підготовки до виробництва. Всі зміни проходять рецензування та автоматичну перевірку перед об'єднанням, встановлюючи контрольні точки якості та аудиторські сліді.
- **Рівень конвеєра CI/CD:** автоматизовані конвеєри збірки, що запускаються комітами репозиторію, виконують багатоетапну валідацію: модульне тестування перевіряє правильність компонентів, інтеграційне тестування перевіряє міжкомпонентні інтерфейси, а тестування на основі симуляції виконує сценарії із замкнутим циклом у віртуальних середовищах. Успішна валідація запускає побудову контейнерних образів з багатоархітектурною підтримкою за допомогою Docker BuildKit з крос-компіляцією або нативними ARM/x86-бігунами. Побудовані образи криптографічно підписуються і публікуються в реєстрах артефактів (Harbor, Docker Registry, хмарні контейнерні реєстри) з семантичними тегами версій і незмінними дайджестами [9].
- **Рівень оркестрування та розгортання:** Цей рівень перетворює наміри розгортання на конкретні дії в усьому парку роботів. Модельні механізми розгортання парку вирішують проблеми задоволення обмежень, щоб призначити версії програмного забезпечення пристроям на основі можливостей апаратного забезпечення, топології мережі, операційного контексту та обмежень політики. Оператори GitOps постійно моніторять репозиторії Git на предмет змін маніфесту та автоматично узгоджують стан кластера. Контролери Kubernetes керують контейнеризованими мікросервісами, а спеціалізовані оператори обробляють передачу стану для промислових контролерів та інтеграцію з неконтейнеризованими компонентами [4][8].
- **Рівень периферійних пристроїв:** Роботизовані одиниці запускають контейнерні середовища виконання (Docker, containerd, CRI-O), які керуються агентами оркестрування (Kubernetes kubelet, спеціальні агенти). Локальні агенти виконують інструкції з розгортання: витягують образи з реєстрів, зупиняють існуючі контейнери, запускають оновлені контейнери та виконують перевірки працездатності. Для промислових контролерів оператори передачі стану координують роботу з контрольованими середовищами виконання, щоб витягувати стан з працюючих екземплярів, передавати його до нових версій та виконувати безперебійне переключення [1].
- **Рівень спостережності та контролю:** Централізований моніторинг агрегує метрики (стан розгортання, використання ресурсів, частоту помилок), журнали (вихідні дані додатків, системні події) та траси (розподілені потоки транзакцій) з усіх пристроїв парку. Панелі інструментів забезпечують видимість у реальному часі прогресу оновлення та стану системи. Правила оповіщення запускають сповіщення при виникненні аномалій, а автоматизовані політики виправлення можуть ініціювати відкат або запускати робочі процеси реагування на інциденти [10].

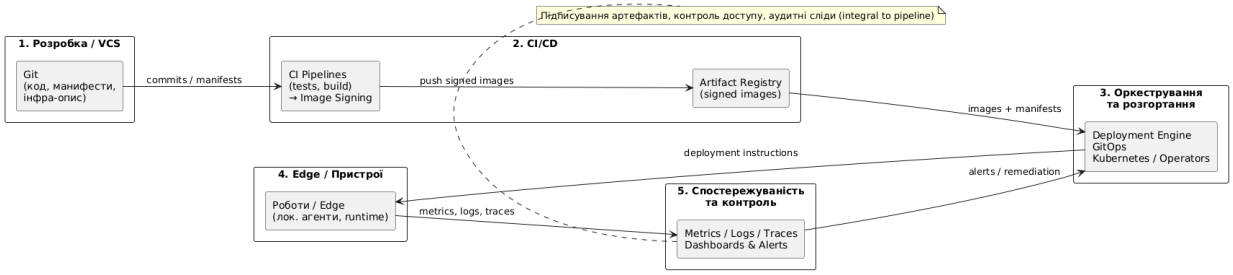

Рис. 1. Архітектура системи

На рисунку 1 представлено діаграму інформаційних потоків в цій архітектурі що проходять структурований процес: розробники фіксують зміни коду в Git, запускаючи CI-конвеєри, які створюють і тестують артефакти. У разі успішної побудови контейнерні образи публікуються в реєстрах і оновлюються маніфести розгортання. Рівні оркестрування виявляють зміни в маніфестах і обчислюють плани розгортання з урахуванням обмежень пристроїв і політик розгортання (канарка, синьо-зелена, послідовна). Агенти пристроїв виконують розгортання, витягуючи образи та оновлюючи контейнери, що працюють. Системи спостереження збирають телеметричні дані протягом усього процесу, що дозволяє операторам контролювати хід роботи та втручатися в разі необхідності.

Завдяки модульності архітектури та чіткому розподілу функцій між компонентами, система залишається масштабованою та легко адаптується до змін у вимогах або середовищі розгортання. Компоненти системи взаємодіють через захищені канали, а кожен етап процесу оновлення супроводжується логуванням і перевіркою результатів, що підвищує прозорість та контрольованість. Архітектурне рішення орієнтоване на стабільну роботу в умовах нестабільного мережевого з'єднання, з можливістю повторного виконання або відновлення процесу у разі збою.

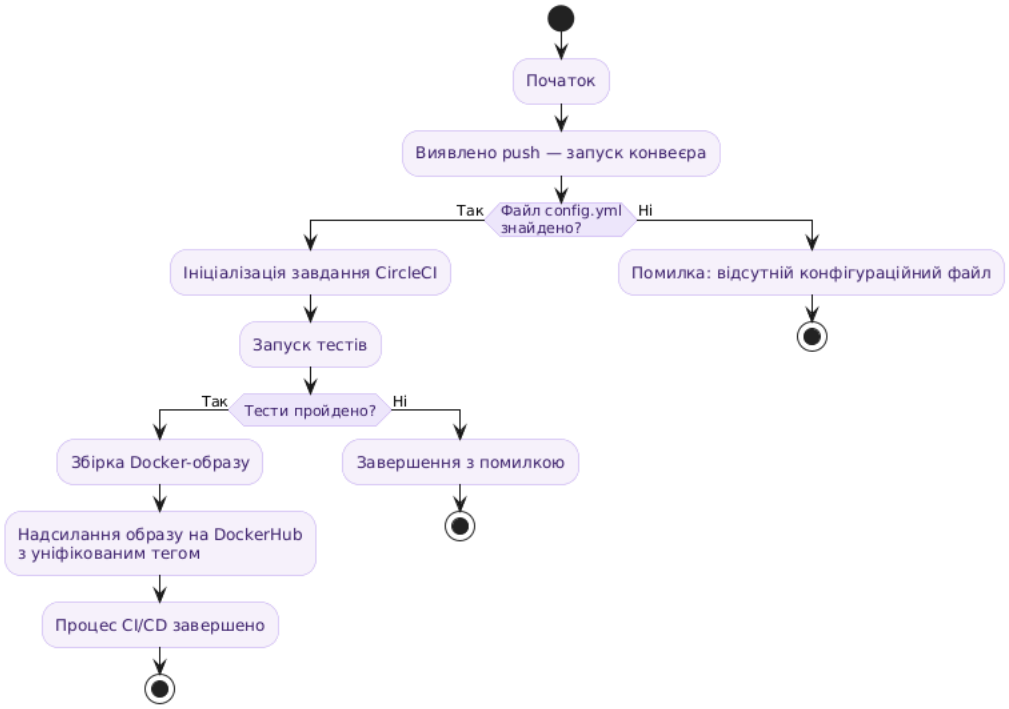

Рис. 2. Алгоритм роботи системи

Автоматизований процес безперервної інтеграції та доставки (CI/CD) забезпечує збірку, тестування та завантаження Docker-образу в DockerHub за допомогою сервісу CircleCI. Схему алгоритму процесу зображено на рис. 2. Процес починається з того, що розробник виконує push змін до віддаленого репозиторію (наприклад, GitHub або GitLab). Після цього перевіряється наявність конфігураційного файлу config.yml, який є обов'язковим для запуску процесів у CircleCI.

Якщо конфігураційний файл наявний, запускається CI-пайплайн. У рамках цього пайплайна виконується тестування коду, і на основі його результатів приймається рішення про подальші дії:

- якщо юніт тести не проходять, процес завершується з помилкою та надсилається відповідне повідомлення розробнику для усунення недоліків;
- якщо юніт тести успішні, відбувається збірка Docker-образу.

Після успішної збірки система авторизується в DockerHub і виконує публікацію (push) образу з уніфікованим тегом, що дозволяє легко ідентифікувати останню версію контейнера. Завершення цього етапу означає успішне завершення CI/CD процесу.

Наступним етапом є розгортання нового програмного забезпечення на цільових пристроях.

Процес починається із запуску Ansible playbook адміністратором. Далі система встановлює SSH-з'єднання з цільовим пристроєм (наприклад, віртуальною машиною або реальним контролером робота).

Після встановлення з'єднання завантажується (pull) актуальний образ контейнера з DockerHub. Якщо попередній контейнер працює, він коректно зупиняється і видаляється для уникнення конфліктів. Потім запускається новий контейнер з оновленим програмним забезпеченням.

На завершення система перевіряє, чи контейнер успішно працює:

- якщо контейнер успішно працює процес розгортання вважається завершеним і адміністратор отримує підтвердження про успіх;
- якщо сталася помилка адміністратор отримує детальне повідомлення про збій, що дозволяє оперативно виявити та усунути проблему.

Такий підхід забезпечує надійність, прозорість і контрольованість процесу оновлення, що особливо важливо в умовах розподіленої та динамічної інфраструктури роботизованих систем.

Реалізація системи. Реалізація платформи автоматичного оновлення базується на узгодженій роботі кількох модулів, що утворюють повний конвеєр доставки. Конвеєри CI/CD у форматі pipeline-as-code визначають етапи побудови, тестування й публікації, забезпечуючи відтворюваність та інтеграцію з реєстрами артефактів і системами розгортання. Модулі автоматизації на основі Ansible інкапсулюють логіку оновлень, тоді як динамічна інвентаризація синхронізує список цільових пристроїв із системами управління парком. Конфігураційні сховища (Git, Vault, ConfigMaps/Secrets) централізують параметри та секрети, а шаблонізатори (Jinja2, Helm) формують узгоджені маніфести з вбудованою валідацією. Підсистема спостережності на базі Prometheus, Grafana та трасування забезпечує телеметрію й налагодження складних взаємодій, а модуль управління парком підтримує інвентаризацію для формування коректних планів розгортання.

Контейнеризація потребує спеціальних оптимізацій для периферійної робототехніки. Мінімальні базові образи, включно зі спеціальними варіантами для ROS 2, зменшують розмір контейнерів і поверхню атаки [13]. Багатоетапні збірки скорочують фінальний образ, що критично для компонентів C++, Go та Rust. Оптимізована структура шарів підвищує ефективність кешування та мінімізує обсяг переданих змін, а спільні стабільні шари прискорюють доставку споріднених сервісів [12]. Для гетерогенних парків важливими є багатоархітектурні образи, які автоматично обирають відповідну платформу; їх формування спрощують Buildx і маніфест-листи. Додаткове посилення безпеки відмова від root, файлові системи лише для читання, обмеження можливостей ядра та обов'язкове сканування образів у CI знижує ризики вразливостей, формуючи стійку модель контейнеризації.

Оркестрація розгортань реалізується через ієрархічні інвентари Ansible й структуровані плейбуки, що покривають сценарії від початкового забезпечення до відкату. Конфігурації середовищ ізольовано зберігаються у Git-гілках, що дає змогу тестувати процедури перед впровадженням у виробництво. Секрети ніколи не версіонуються: вони захищаються Ansible Vault або менеджерами секретів, а в Kubernetes керуються через Secrets та спеціальні оператори. Декларативні маніфести Kubernetes і чарти Helm задають бажаний стан, який контролери та GitOps-оператори автоматично підтримують. Перевірка платформи включає тестування сценаріїв: первинне налаштування демонструє повну автоматизацію, рутинні оновлення підтверджують наскрізний цикл доставки, швидкі патчі безпеки зменшують вікно вразливості. Стійкість системи доводять сценарії порушень мережі з автоматичним завершенням оновлень, а здатність до відкату підтверджують GitOps-механізми. Для критичних контролерів продемонстровано безперервне оновлення з переносом стану без зупинки циклу керування, що підтверджує придатність підходу до систем реального часу [1].

Проведення експерименту розробленої системи. Для перевірки надійності реалізованої платформи автоматизованого оновлення було проведено два експерименти: типовий сценарій оновлення та сценарій із примусовою відсутністю локального образу, що дозволяє оцінити стійкість до збоїв. У першому випадку в репозиторій GitHub було додано зміну в коді, після чого створення нового тега ініціювало процес CI/CD, що включав збірку, тестування та публікацію Docker-образу v1.1.1. Початковий етап оновлення показано на рис. 3.

```
oleh@hp:~$ cd my-robot-platform/
oleh@hp:~/my-robot-platform$ git add .
oleh@hp:~/my-robot-platform$ git commit -m "Test commit"
[main 2ea3c1d] Test commit
 1 file changed, 1 insertion(+), 1 deletion(-)
oleh@hp:~/my-robot-platform$ git tag v1.1.1
oleh@hp:~/my-robot-platform$ git push origin v1.1.1
Username for 'https://github.com': olehdm
Password for 'https://olehdm@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 323 bytes | 323.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/olehdm/robots-updates.git
 * [new tag]         v1.1.1 -> v1.1.1
oleh@hp:~/my-robot-platform$
```

Рис. 3. Внесення змін до проєкту та завантаження на GitHub

Pipeline на CircleCI успішно зібрав та опублікував оновлений образ, після чого адміністратор запустив Ansible-playbook, який виконав оновлення цільових вузлів. Результат роботи автоматизації зображено на рис. 4.

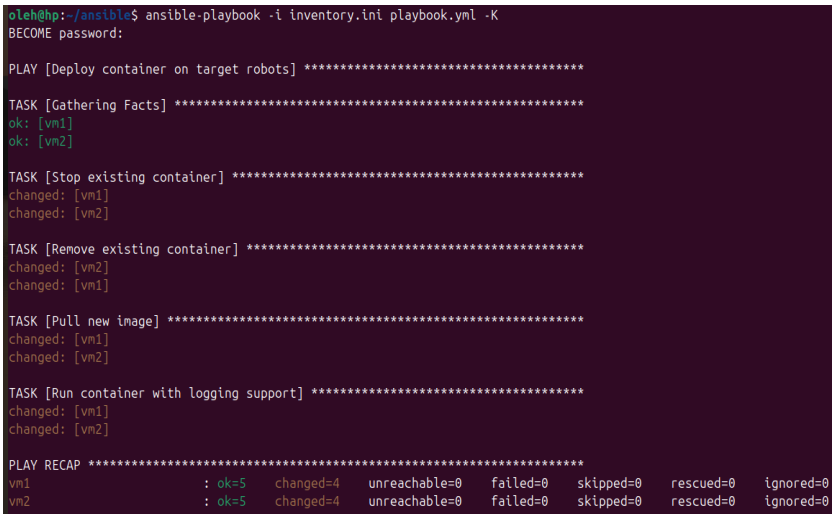

Рис. 4. Результат виконання процесу оновлення

Система виконала повний цикл: зупинку старого контейнера, завантаження нового образу та запуск оновленої версії. Оновлення відбулося на двох віртуальних машинах без помилок, що підтверджує коректність базового сценарію.

Другий експеримент імітує ситуацію, коли на одній із машин було вручну видалено контейнер та локальний образ. Після повторного запуску того ж playbook система автоматично завантажила відсутній образ із DockerHub, коректно розгорнула новий контейнер і відновила роботу сервісу, тоді як інша машина, що мала актуальну версію, не зазнала змін. Цей механізм демонструє селективне оновлення лише тих вузлів, які дійсно потребують актуалізації, що зменшує навантаження на інфраструктуру. Підсумковий результат оновлення показано на рис. 5.

Рис. 5. Оновлення лише вузла з відсутнім Docker-образом

У результаті експериментів підтверджено, що розроблена платформа забезпечує стабільне та відмовостійке віддалене оновлення програмного забезпечення роботів: система перевіряє наявність Docker, встановлює його за потреби, завантажує нові образи, перезапускає контейнер лише при наявності оновлення та автоматично налаштовує параметри середовища. Механізм контролю версій працює коректно за

відсутності нової версії контейнер не перезапускається, що зменшує простій і навантаження. Гнучкість системи забезпечується конфігурацією в Ansible-плейбуку, де змінюються лише назва образу, тег і параметри контейнера, без модифікації загальної логіки. Повний цикл оновлення триває орієнтовно від 1 до 10 хвилин залежно від швидкості завантаження та початкового стану машини. Отримані результати демонструють ефективність, масштабованість і придатність платформи до регулярного використання в робототехнічних DevOps-процесах.

АНАЛІЗ ТА ДИСКУСІЯ

Оцінка продуктивності сучасних автоматизованих платформ оновлення для периферійних робототехнічних систем підтверджує їхню придатність до промислового застосування за умови ретельного вибору архітектурних рішень. Затримка оновлення у контейнеризованих середовищах зазвичай коливається від кількох секунд до кількох хвилин і залежить від розміру образів та пропускної здатності мережі, при цьому накладні витрати контейнеризації залишаються низькими менше 5% погіршення продуктивності для типової периферійної робототехніки [16]. Пакетна оркестрація демонструє до 30-кратного прискорення порівняно з ручними оновленнями [2], а оператори передачі стану забезпечують перенесення понад 100 000 змінних менш ніж за 42 мс, що відповідає часовим обмеженням промислових циклів [1]. Значні переваги дають оптимізації розподілу образів: спільні шари скорочують час завантаження до 65% і зменшують використання пропускної здатності до 30% [12]. Попри загальну ефективність, додатки із субмілісекундними вимогами можуть потребувати середовищ виконання реального часу або оптимізованих контейнерних рантаймів [17]. Інтелектуальне автомасштабування на основі TAPS покращує відгук системи та зменшує витрати ресурсів, перевершуючи стандартні методи горизонтального масштабування [14].

Архітектура платформи підтверджує свою універсальність у різних промислових доменах. У дослідницькій робототехніці вона спрощує роботу з гетерогенними платформами, дозволяючи швидко розгортати нові версії ПЗ, підтримувати паралельні конфігурації та безпечно відкотити зміни. У виробничих середовищах ключовою перевагою є можливість безперервного оновлення промислових контролерів без простоїв, тоді як централізоване управління парком гарантує видимість версій і відповідність політикам якості. У складській автоматизації платформа підтримує масштабовані оновлення AMR і застосування Canary-стратегій для мінімізації ризиків, а в польовій робототехніці автономну роботу й диференціальні оновлення, важливі за нестабільного підключення. Для успішного впровадження потрібна інтеграція платформи з інфраструктурою підприємства аутентифікацією, політиками сегментації мережі, системами моніторингу та процесами управління змінами, що забезпечує безпеку та узгодженість у виробничих умовах.

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

У статті виконано узагальнений аналіз сучасних автоматизованих платформ віддаленого оновлення програмного забезпечення для промислових робототехнічних систем, зосереджений на контейнеризації та оркестрації у периферійних середовищах. Запропонована архітектура інтегрує контейнеризовані сервіси, CI/CD-конвеєри, Kubernetes-орієнтоване оркестрування та GitOps, забезпечуючи узгоджене, безпечне й відмовостійке оновлення ПЗ на різнорідному обладнанні. Серед ключових результатів систематизація методів оновлення, архітектурних шаблонів і показників продуктивності, зокрема передачі стану <42 мс, 30-кратного прискорення пакетних оновлень і мінімальних накладних витрат контейнеризації.

Платформа підтверджує можливість створення масштабованого та безпечного механізму доставки оновлень для промислових роботів, де контейнеризація забезпечує портативність, оркестратори відповідають за оновлення у масштабі, а оператори стану дозволяють виконувати критичні оновлення без простоїв. GitOps гарантує прозоре керування версіями та надійність відкатів. Робота окреслює перспективні напрями подальших досліджень, включно з механізмами самовідновлення, ML-виявленням аномалій, диференціальними оновленнями та легковаговими платформами оркестрації, що є важливими в умовах зростання складності та підключеності промислових роботизованих систем.

References

1. Koziolek, H., Burger, A., & Puthan Peedikayil, A. (2024). Fast state transfer for updates and live migration of industrial controller runtimes in container orchestration systems. *Journal of Systems and Software*, 112004. https://doi.org/10.1016/j.jss.2024.112004
2. Zhang, Y., Wurll, C., & Hein, B. (2023). KubeROS: A Unified Platform for Automated and Scalable Deployment of ROS2-based Multi-Robot Applications. У *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. https://doi.org/10.1109/icra48891.2023.10160632
3. Zhang, J., Keramat, F., Yu, X., Hernandez, D. M., Queralta, J. P., & Westerlund, T. (2022). Distributed Robotic Systems in the Edge-Cloud Continuum with ROS 2: a Review on Novel Architectures and Technology Readiness. У *2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE. https://doi.org/10.1109/fmec57183.2022.10062523
4. Song, H., Dautov, R., Ferry, N., Solberg, A., & Fleurey, F. (2022). Model-based fleet deployment in the IoT–edge–cloud continuum. *Software and Systems Modeling*. https://doi.org/10.1007/s10270-022-01006-z
5. Koziolek, H., Burger, A., Abdulla, P. P., Rückert, J., Sonar, S., & Rodriguez, P. (2021). Dynamic Updates of Virtual PLCs Deployed as Kubernetes Microservices. У *Software Architecture*(с. 3–19). Springer International Publishing. https://doi.org/10.1007/978-3-030-86044-8_1

6. Lopez-Viana, R., Diaz, J., Diaz, V. H., & Martinez, J.-F. (2020). Continuous Delivery of Customized SaaS Edge Applications in Highly Distributed IoT Systems. *IEEE Internet of Things Journal*, *7*(10), 10189–10199. https://doi.org/10.1109/jiot.2020.3009633

7. Zmuda, J. v., Koch, T., & Ahmed, S. (2025). Continuous Integration and Delivery of ROS2 projects. У *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*(с. 2388–2395). IEEE. https://doi.org/10.1109/case58245.2025.11163918

8. Linsbauer, S., Hark, R., Koziolek, H., & Eskandani, N. (2024). Runtime Orchestration of Distributed Control System Services with TOSCA, Kubernetes, and GitOps. У *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*(с. 40–47). IEEE. https://doi.org/10.1109/icsa-c63560.2024.00013

9. Mateo-Casali, M. A., Boza, A., Candea, C., Fraile, F., Morand, L., & Nahshon, Y. (2025). Continuous Integration, Deployment and Validation: Supporting Scalable Industrial Ecosystems. У *2025 IEEE International Conference on Engineering, Technology, and Innovation (ICE/ITMC)*(с. 1–9). IEEE. https://doi.org/10.1109/ice/itmc65658.2025.11106635

10. Venkata Krishna Koganti. (2025). Autonomous CI/CD Meshes: Self-healing deployment architectures with AI-ML Orchestration. *World Journal of Advanced Engineering Technology and Sciences*, *15*(2), 2731–2745. https://doi.org/10.30574/wjaets.2025.15.2.0777

11. Luo, Z., Li, D., Wan, J., Wang, S., Wang, G., Cheng, M., & Li, T. (2024). Component integration manufacturing middleware for customized production. *Advanced Engineering Informatics*, *59*, 102317. https://doi.org/10.1016/j.aei.2023.102317

12. Liu, Y., Yang, B., Wu, Y., Chen, C., & Guan, X. (2022). How to Share: Balancing Layer and Chain Sharing in Industrial Microservice Deployment. *IEEE Transactions on Services Computing*, 1–14. https://doi.org/10.1109/tsc.2022.3230699

13. Busch, J.-P., Reiher, L., & Eckstein, L. (2024). Enabling the Deployment of Any-Scale Robotic Applications in Microservice Architectures through Automated Containerization*. У *2024 IEEE International Conference on Robotics and Automation (ICRA)*(с. 17650–17656). IEEE. https://doi.org/10.1109/icra57147.2024.10611586

14. Roy, S., Dobson, R., Campbell, J., & Kalafatis, S. (2024). Efficient Resource Allocation for Multi-Robot Collaboration via Traffic-Aware Pod Autoscaling. У *2024 7th Conference on Cloud and Internet of Things (CIoT)*(с. 1–8). IEEE. https://doi.org/10.1109/ciot63799.2024.10757045

15. Lampe, B., Reiher, L., Zanger, L., Woopen, T., van Kempen, R., & Eckstein, L. (2023). RobotKube: Orchestrating Large-Scale Cooperative Multi-Robot Systems with Kubernetes and ROS. У *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. https://doi.org/10.1109/itsc57777.2023.10422370

16. Garcia, A., Tardos, J., Ferrando, J. L., & Fernandez, T. (2023). Containerized Edge Architecture for Centralized Industry 4.0 Fleet Management. У *2023 IEEE 9th World Forum on Internet of Things (WF-IoT)*. IEEE. https://doi.org/10.1109/wf-iot58464.2023.10539473

17. Lumpp, F., Fummi, F., Patel, H. D., & Bombieri, N. (2022). Containerization and Orchestration of Software for Autonomous Mobile Robots: a Case Study of Mixed-Criticality Tasks across Edge-Cloud Computing Platforms. У *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. https://doi.org/10.1109/iros47612.2022.9981581