УДК 004.4`2

Пліхта О.О.
Ліщинська Л.Б.

АВТОМАТИЗАЦІЯ ПРОЦЕСІВ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗА ДОПОМОГОЮ CI/CD: ПЕРЕВАГИ, ВИКЛИКИ, ПРАКТИКИ

Вінницький національний технічний університет

Анотація

У статті розглянуто сучасні підходи до автоматизації розробки програмного забезпечення на основі методології CI/CD. Описано переваги практичного впровадження таких підходів у проектах різного масштабу, починаючи з невеликих стартапів і закінчуючи великими корпораціями. Розкрито ключові виклики, які виникають при реалізації CI/CD, а також подано порівняльну характеристику популярних інструментів автоматизації. Особливу увагу приділено практичному застосуванню CI/CD: від конфігурації пайплайнів до інтеграції з сучасними DevOps-інструментами та техніками розгортання. Стаття підкреслює роль CI/CD у підвищенні продуктивності, стабільності та гнучкості процесів розробки.

Ключові слова: CI/CD, DevOps, автоматизація розробки, Continuous Integration, Continuous Delivery, Jenkins, GitHub Actions, GitLab CI/CD, пайплайн, розгортання, тестування, мікросервіси.

Abstract

The article discusses modern approaches to software development automation based on the CI/CD methodology. The advantages of practical implementation of such approaches in projects of various sizes, from small startups to large corporations, are described. The key challenges that arise in the implementation of CI/CD are revealed, and a comparative description of popular automation tools is provided. Particular attention is paid to the practical application of CI/CD: from configuration of pipelines to integration with modern DevOps tools and deployment techniques. The article emphasizes the role of CI/CD in increasing the productivity, stability, and flexibility of development processes.

Keywords: CI/CD, DevOps, development automation, Continuous Integration, Continuous Delivery, Jenkins, GitHub Actions, GitLab CI/CD, pipeline, deployment, testing, microservices.

Вступ

Інтенсивний розвиток IT-індустрії зумовлює потребу у постійному вдосконаленні підходів до розробки програмного забезпечення. Одним із ключових викликів сьогодення є забезпечення стабільної, швидкої та якісної доставки змін до програмного продукту. У цьому контексті автоматизація процесів розробки набуває особливої актуальності, зокрема через впровадження практик CI/CD (Continuous Integration / Continuous Delivery або Deployment), що стали невід'ємною частиною DevOps-культури. Ці практики дозволяють не лише скоротити час між внесенням змін та їх впровадженням у продуктивне середовище, а й зменшити кількість помилок завдяки автоматизованому тестуванню та контролю якості.

Актуальність теми дослідження зумовлена тим, що попри широке поширення концепції CI/CD, її впровадження вимагає ретельного аналізу переваг і ризиків, вибору відповідних інструментів та адаптації до специфіки конкретного проєкту. У багатьох компаніях, особливо тих, що переходять до DevOps-культури, виникають труднощі із впровадженням автоматизованих пайплайнів, налаштуванням тестування, деплойменту та забезпеченням контролю якості. Саме тому необхідно систематизувати практичні аспекти CI/CD, окреслити можливі виклики і проаналізувати типові сценарії використання [1].

Мета дослідження

Метою дослідження є аналіз і узагальнення практичних аспектів автоматизації процесів розробки програмного забезпечення за допомогою підходу CI/CD, а також визначення основних переваг,

викликів і сучасних інструментів, які використовуються для реалізації CI/CD у проєктах різного масштабу.

Практичні переваги автоматизації через CI/CD

CI/CD (Continuous Integration / Continuous Delivery або Deployment) – це методологія автоматизації процесів розробки, яка забезпечує швидке, контрольоване та передбачуване внесення змін у програмне забезпечення. На відміну від класичного каскадного підходу або навіть Agile-практик без DevOps, CI/CD створює замкнений цикл: від написання коду до його доставки у продуктивне середовище – за лічені хвилини або години, а не дні чи тижні.

Реальне застосування CI/CD демонструє відчутний приріст продуктивності. Наприклад, у компанії Shopify, де щодня генерується понад 10 000 pull-запитів, впровадження CI дозволило скоротити час перевірки змін з кількох годин до кількох хвилин, що безпосередньо вплинуло на швидкість оновлень продукту.

CI забезпечує автоматичну перевірку інтеграцій змін кожного розробника з основною гілкою репозиторію, запобігаючи конфліктам і накопиченню помилок. Натомість CD дозволяє автоматизувати випуск нових версій, наприклад через blue-green deployment, canary releases чи feature toggles, що знижує ризики виходу з ладу в продуктивному середовищі.

Основні виклики впровадження CI/CD

Втім, ефективна реалізація CI/CD на практиці стикається з низкою складнощів. Наприклад, для команди, що раніше не використовувала автоматизовану інфраструктуру, побудова першого пайплайну може зайняти тижні. Це включає налаштування систем збірки, написання тестів, створення шаблонів YAML для пайплайнів, підключення сховищ секретів, налаштування оркестрації контейнерів, тощо. Окремою проблемою є непідготовленість коду – наприклад, відсутність тестів або tightly coupled-модулі, які ускладнюють тестування. У великих monolith-проєктах CI-процеси часто займають багато часу через довгу збірку або складні залежності. У таких випадках необхідне попереднє рефакторингування або навіть перехід до мікросервісної архітектури. Фінансові витрати – ще один виклик. Використання хмарних CI/CD-сервісів, таких як GitHub Actions чи CircleCI, може призводити до значного навантаження на бюджет при збільшенні кількості пайплайнів, тоді як локальні рішення вимагають серверів і адміністрування.

Порівняльна характеристика інструментів CI/CD

Для розробників і DevOps-інженерів вибір інструменту CI/CD має стратегічне значення. Порівняльна характеристика інструментів наведена в таблиці 1.

Таблиця 1 – Порівняльна характеристика інструментів CI/CD

Інструмент	Тип розгортання	Переваги	Недоліки
GitHub Actions	Хмарний, вбудований	Простота інтеграції з GitHub, багато шаблонів, безкоштовний для open-source	Ліміти на хвилини, обмежена кастомізація ресурсів
GitLab CI/CD	Хмарний / On-Prem	Гнучкість, підтримка Docker/Kubernetes, власні раннери	Складніша конфігурація, потреба в глибокому DevOps-досвіді
Jenkins	On-Prem / Docker	Надзвичайна кастомізація через плагіни, повна автономія	Висока складність підтримки, застарілий UI, потреба у адмініструванні
CircleCI	Хмарний	Висока швидкість, ефективна підтримка Docker, хороші аналітичні панелі	Платна модель після початкових лімітів
Azure DevOps	Хмарний / On-Prem	Інтеграція з Azure, потужна підтримка enterprise-циклів	Надлишкова складність для малих команд

На основі порівняльної характеристики можна зробити висновок, що GitHub Actions підходить для невеликих та open-source проєктів завдяки простоті інтеграції та мінімальному налаштуванню [2].

GitLab CI/CD ідеальний для команд, що хочуть об’єднати керування репозиторієм, пайплайни та моніторинг в одній платформі [3]. Jenkins – найкраще рішення для великих організацій з потребою в гнучкій кастомізації, хоча й складний в адмініструванні [4]. CircleCI вирізняється високою продуктивністю та підходить для проєктів з активним використанням Docker і швидкими релізами [5]. Azure DevOps ефективний у корпоративному середовищі з інтеграцією в екосистему Microsoft та складними життєвими циклами ПЗ [6].

Практичне застосування

CI/CD-підхід активно впроваджується в реальні проєкти як малих команд, так і великих компаній. Його головна цінність проявляється у здатності забезпечити стабільність, масштабованість та швидке реагування на зміну вимог у динамічному середовищі розробки.

Розглянемо типовий сценарій практичного застосування. У невеликому стартапі, що розробляє веб-додаток, команда вирішує запровадити GitHub Actions для автоматичної перевірки pull-запитів. Для цього достатньо створити файл конфігурації в репозиторії (.github/workflows/main.yml), у якому прописується послідовність дій: збірка проєкту, запуск юніт-тестів, перевірка лінтером, і за потреби – деплой на тестовий сервер. Це дозволяє уникнути ручної перевірки коду, пришвидшує рецензування та зменшує ризик «зламаного» основного коду.

У більших компаніях приклади складніші. Наприклад, при розробці мікросервісної архітектури у GitLab CI/CD створюється окремий пайплайн для кожного мікросервісу. Зміни, внесені в один сервіс, автоматично запускають пайплайн: сервіс збирається у Docker-образ, тестується, перевіряється на якість (наприклад, через SonarQube), після чого образ надсилається у Docker Registry. У фінальному етапі – мікросервіс розгортається у staging-кластері Kubernetes із використанням Helm-чартів. Такий процес дозволяє уникнути людського фактора на критичних етапах та забезпечує відтворюваність оточення.

У DevOps-практиках підприємств із високим навантаженням (наприклад, у фінансових або e-commerce компаніях), широко використовуються Jenkins або Azure DevOps, які інтегруються з внутрішніми системами моніторингу, управління безпекою та деплоєм. Наприклад, Jenkins може бути інтегрований із LDAP для авторизації, Nexus або Artifactory як репозиторіями артефактів, а також Prometheus/Grafana для аналізу метрик виконання пайплайнів. Це дозволяє масштабувати CI/CD-процеси на десятки команд та сотні компонентів.

Особливої уваги заслуговують підходи до progressive delivery – такі як canary deployment або blue-green deployment, коли нова версія програми розгортається спочатку лише на невеликий відсоток користувачів, і лише після перевірки показників – на всіх інших. Такі практики можливі лише за умов повної автоматизації CI/CD, і значно знижують ризик масштабних збоїв.

Отже, CI/CD на практиці – це не просто автоматизований запуск тестів, а ціла екосистема процесів, інструментів та практик, які дозволяють командам зосередитися на створенні цінності, а не на рутинних операціях. Його впровадження в реальні проєкти приносить відчутні результати: прискорення розгортання, підвищення стабільності, прозорість розробки та зниження кількості помилок у продакшн-середовищі.

Висновок

У результаті проведеного дослідження встановлено, що використання CI/CD значно підвищує ефективність процесів розробки ПЗ, сприяє зменшенню часу виведення продукту на ринок і покращує загальну якість програмного забезпечення. Систематизовано ключові переваги автоматизації, такі як швидше виявлення помилок, безперервне тестування, надійне розгортання та підвищення командної продуктивності.

Разом з тим, виявлено низку типових викликів, зокрема складність налаштування пайплайнів, технічний борг, потребу в високій культурі DevOps. Для подолання цих бар’єрів рекомендовано впроваджувати CI/CD поетапно, з урахуванням потреб конкретного проєкту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Gruver, G., & Mouser, T. (2021). *Engineering the Digital Transformation: Continuous Delivery and DevOps at Scale in the Enterprise*. IT Revolution Press.

2. GitHub Actions. – URL: https://docs.github.com/en/actions (дата звернення 01.05.2025).

3. GitLab CI/CD. – URL: https://docs.gitlab.com/ee/ci/ (дата звернення 01.05.2025).

4. Jenkins. – URL: https://www.jenkins.io/doc/ (дата звернення 01.05.2025).
5. CircleCI. – URL: https://circleci.com/docs/ (дата звернення 01.05.2025).
6. Azure DevOps. – URL: https://learn.microsoft.com/en-us/azure/devops/ (дата звернення 01.05.2025).

Пліхта Олександр Олександрович – студент групи 1ПІ-24м, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, e-mail: s.plihta000@gmail.com

Ліщинська Людмила Броніславівна – д-р техн. наук, професор, професор кафедри програмного забезпечення, Вінницький національний технічний університет, м. Вінниця, e-mail: llb@vntu.edu.ua

Plikhta Oleksandr Oleksandrovych – Student of group 1PI-24m, Faculty for Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: s.plihta000@gmail.com

Lishchynska Lyudmyla Bronislavivna – Dr. Sc. (Eng.), Full Professor, Professor of Program Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: llb@vntu.edu.ua