МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВЕТЕРИНАРНОЇ МЕДИЦИНИ
ТА БІОТЕХНОЛОГІЙ ІМ. С.З. ГЖИЦЬКОГО

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

# КВАЛІФІКАЦІЙНА РОБОТА

Другого (магістерського) рівня вищої освіти

на тему: “ Побудова CI/CD-процесу для автоматизації розгортання веб застосунків”

Виконав: студент 6 курсу групи Іт-62
Спеціальності 126 – „Інформаційні системи та технології”
(шифр і назва)

Колесник Юрій Віталійович
(Прізвище та ініціали)

Керівник: к.т.н., доц. Падюка Р.І.
(Прізвище та ініціали)

Рецензенти: к.т.н., доц. Березовецький С. А.
(Прізвище та ініціали)

(Прізвище та ініціали)

ДУБЛЯНИ-2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВЕТЕРИНАРНОЇ МЕДИЦИНИ ТА БІОТЕХНОЛОГІЙ ІМ. С.З. ГЖИЦЬКОГО

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Другий (магістерський) рівень вищої освіти
Спеціальність 126 "Інформаційні системи та технології"

"ЗАТВЕРДЖУЮ"
Завідувач кафедри_____
д.т.н., проф. А.М. Тригуба
"____"_____2025 р.

# *ЗАВДАННЯ*

на кваліфікаційну роботу студенту

Колеснику Юрію Віталійовичу

1. Тема роботи: «Побудова CI/CD-процесу для автоматизації розгортання веб застосунків»

Керівник роботи Падюка Роман Іванович, к.т.н., доцент.

Затверджені наказом по університету від 28 лютого 2025 року № 140/к-с.

2. Строк подання студентом роботи 05.12.2025 р.

3. Початкові дані до роботи: Вихідні дані щодо предметної області та вебзастосунку; Характеристики програмно-апаратного середовища та хмарної інфраструктури; Обраний технологічний стек розроблення; Методичні підходи до проєктування та оцінювання ефективності CI/CD-процесу. Інструментарій автоматизації CI/CD-процесу на основі Jenkins.

4. Зміст розрахунково-пояснювальної записки:

1. Аналіз стану питання в практиці та теорії
2. Обґрунтування, вибір та реалізація інструментарію вирішення задачі
3. Розробка та експериментальна перевірка CI/CD-процесу для вебзастосунку
4. Охорона праці та безпека в надзвичайних ситуаціях
5. Визначення ефективності розробленого CI/CD-процесу

Висновки та пропозиції.

Бібліографічний список.

Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): Тема, автор, керівник магістерської роботи; Мета,завдання, об'єкт, предмет дослідження; Актуальність і проблематика; Теоретична база CI CD; Обґрунтування вибору Jenkins; Архітектура рішення та потоки взаємодії;

Модель задачі наближена до комерційного проєкту; Розгортання Jenkins у робочому середовищі;Інтеграція з репозиторієм та підготовка Job; Агенти Jenkins і розподіл виконання; CI та автоматизовані перевірки якості; CD та контроль послідовності етапів; Розгортання у staging та production; Результати та ефективність впровадження

6. Консультанти з розділів:

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1, 2, 3, 5 | *Падюка Р.І., доцент кафедри інформаційних технологій* | | |
| 4 | *Городецький І.М., доцент кафедри інженерної механіки* | | |

7. Дата видачі завдання 03 березня 2025 р.

*КАЛЕНДАРНИЙ ПЛАН*

| № з/п | Назва етапів дипломної роботи | Строк виконання етапів роботи | Приміт-ка |
|---|---|---|---|
| 1 | *Написання першого розділу та означення головних завдань роботи* | 03.03.-01.04.25 | |
| 2 | *Виконання другого розділу та формування головних показників для розрахунків* | 02.04.-01.05.25 | |
| 3. | *Виконання третього розділу та формування початкових даних* | 02.05.-01.06.25 | |
| 4. | *Виконання четвертого розділу та узагальнення отриманих результатів магістерської роботи* | 02.06.-01.09.25 | |
| 5. | *Вартісне оцінення ефективності пропозицій роботи* | 02.09.-01.10.25 | |
| 6. | *Завершення оформлення розрахунково-пояснювальної записки та аркушів графічної частини* | 02.10.-01.11.25 | |
| 7. | *Завершення роботи в цілому* | 02.11.-05.12.25 | |

Студент _____ Колесник Ю.В.
(підпис)

Керівник роботи _____ Падюка Р.І.
(підпис)

УДК 004.4:004.75:004.738.5

Побудова CI/CD-процесу для автоматизації розгортання веб застосунків – Колесник Ю.В. Магістерська робота. Кафедра ІТ. – Дубляни, Львівський НУВМБ ім. С.З. Гжицького, 2025.

75 с. текст. част., 36 рис., 1 табл., 14 арк. графічної частини, 39 літ. джерел

Кваліфікаційна робота присвячена побудові CI/CD-процесу для автоматизації розгортання вебзастосунків на основі сервера Jenkins. У роботі проаналізовано сучасні підходи до безперервної інтеграції, доставки та розгортання, визначено їх роль у скороченні тривалості життєвого циклу програмного забезпечення та підвищенні його надійності. Обґрунтовано вибір Jenkins серед альтернативних інструментів CI/CD з урахуванням вимог до гнучкості, масштабованості та незалежності від хмарного провайдера. На основі змодельованого комерційного проєкту спроєктовано архітектуру та реалізовано повний конвеєр CI/CD для Java-вебзастосунку, який охоплює автоматизовану збірку, модульне й інтеграційне тестування, розгортання у staging та production середовищах і оповіщення учасників команди.

Ключові слова: CI/CD, безперервна інтеграція, безперервне розгортання, Jenkins, вебзастосунок, DevOps, автоматизація розгортання.

Keywords: CI/CD, continuous integration, continuous deployment, Jenkins, web application, DevOps, deployment automation.

ЗМІСТ

ВСТУП ……. 7
1. АНАЛІЗ СТАСТАНУ ПИТАННЯ В ТЕОРІЇ ТА ПРАКТИЦІ …… 9
1.1. Сучасні підходи до розробки та розгортання вебзастосунків …… 8
1.2. Концепції безперервної інтеграції та безперервного розгортання (CI/CD)……. 12
1.3. Інструментальні засоби та типові архітектури CI/CD-процесів..…….. 16
1.4. Постановка завдання …….. 19
2. ОБҐРУНТУВАННЯ, ВИБІР ТА РЕАЛІЗАЦІЯ ІНСТРУМЕНТАРІЮ ВИРІШЕННЯ ЗАДАЧІ…… 21
2.1. Вимоги до інструментарію побудови CI/CD процесу для вебзастосунків……. 21
2.2. Обґрунтування вибору Jenkins як базового інструменту CI/CD …… 24
2.3. Проєктування архітектури CI/CD конвеєра на основі Jenkins …… 28
2.4. Реалізація конвеєра CI/CD для автоматизації розгортання вебзастосунку…….. 32
3. РОЗРОБКА ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА CI/CD-ПРОЦЕСУ ДЛЯ ВЕБЗАСТОСУНКУ ……... 37
3.1. Моделювання задачі та постановка вимог до CI/CD-процесу……. 37
3.2. Розгортання Jenkins та підготовка інфраструктури …… 39
3.3. Реалізація CI/CD-конвеєра для вебзастосунку на основі Jenkins.……. 43
3.4. Оцінювання результатів побудови CI/CD-конвеєра та їх візуалізація…… 56
4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ……... 62
4.1. Розробка логіко-імітаційної моделі виникнення травм і аварій …… 62
4.2. Планування заходів із покращення умов праці …… 64
4.3. Безпека в надзвичайних ситуаціях …… 65

5. ВИЗНАЧЕННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО CI/CD-ПРОЦЕСУ ТА ОЦІНКА ТРУДОМІСТКОСТІ І ВАРТОСТІ ЙОГО ВПРОВАДЖЕННЯ……. 67
ЗАГАЛЬНІ ВИСНОВКИ……..…….. 71
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……... 73

# ВСТУП

Стрімкий розвиток вебтехнологій та поширення хмарних платформ зумовлюють зростання кількості вебзастосунків, що використовуються в бізнесі, державному секторі та повсякденному житті користувачів. Високі вимоги до безперервної доступності, швидкого оновлення функціональності, надійності та безпеки вебсервісів роблять традиційні ручні підходи до розгортання програмного забезпечення недостатньо ефективними. Ручне виконання операцій деплою є трудомістким, схильним до помилок, ускладнює масштабування систем і сповільнює виведення оновлень у промислову експлуатацію. У цих умовах зростає актуальність впровадження практик безперервної інтеграції та безперервного розгортання (CI/CD), які забезпечують автоматизацію основних етапів життєвого циклу вебзастосунків. Саме тому тема побудови CI/CD-процесу для автоматизації розгортання вебзастосунків є сучасною, практично значущою та відповідає потребам галузі інформаційних технологій.

Метою даної кваліфікаційної роботи є розробка та обґрунтування CI/CD-процесу для автоматизації розгортання вебзастосунків, який забезпечує скорочення часу виведення змін у продуктивне середовище, підвищення надійності релізів і зниження ризику помилок, пов’язаних із людським чинником. Галуззю застосування отриманих результатів є процеси розробки, тестування та експлуатації вебзастосунків у командах програмної інженерії, які використовують системи керування версіями, хмарну інфраструктуру, засоби контейнеризації та оркестрації сервісів. Запропоновані підходи можуть бути використані в комерційних і внутрішніх корпоративних проєктах, а також в освітніх цілях для підготовки фахівців з інформаційних технологій.

Об’єктом дослідження в роботі є процес розгортання вебзастосунків у сучасних обчислювальних середовищах.

Предметом дослідження є методи, моделі та інструментальні засоби побудови CI/CD-процесу для автоматизації розгортання вебзастосунків, а також показники, що характеризують ефективність і надійність цього процесу, зокрема

час виконання операцій деплою, частота виникнення помилок під час розгортання та стабільність роботи системи після внесення змін.

Для досягнення поставленої мети в кваліфікаційній роботі необхідно виконати такі завдання:

- ✓ обґрунтувати актуальність застосування CI/CD у розробці та супроводі вебзастосунків;
- ✓ проаналізувати теоретичні засади безперервної інтеграції та безперервного розгортання в контексті сучасних підходів до розробки програмного забезпечення;
- ✓ дослідити існуючі інструментальні платформи та технології, що використовуються для побудови CI/CD-процесів;
- ✓ визначити вимоги до процесу розгортання обраного вебзастосунку та критерії оцінювання ефективності його автоматизації;
- ✓ спроєктувати архітектуру CI/CD-процесу для цільового вебзастосунку з урахуванням середовища розгортання та особливостей інфраструктури;
- ✓ реалізувати та налаштувати пайплайни безперервної інтеграції й безперервного розгортання на базі обраних засобів;
- ✓ провести експериментальне дослідження розробленого CI/CD-процесу та виконати аналіз отриманих результатів з точки зору підвищення ефективності й надійності розгортання вебзастосунків.

# РОЗДІЛ 1
# АНАЛІЗ СТАСТАНУ ПИТАННЯ В ТЕОРІЇ ТА ПРАКТИЦІ

## 1.1. Сучасні підходи до розробки та розгортання вебзастосунків

Сучасні вебзастосунки посідають ключове місце в інформаційній інфраструктурі підприємств, державних установ та сервісів масового користування. Вони забезпечують функціонування електронної комерції, електронного урядування, дистанційної освіти, соціальних мереж, корпоративних порталів та багатьох інших сервісів. Типовий вебзастосунок має багаторівневу архітектуру, яка включає клієнтський інтерфейс (браузерний або мобільний), серверну частину з бізнес-логікою та підсистеми зберігання даних, що функціонують у розподіленому мережевому середовищі з великою кількістю одночасних користувачів.

До таких систем висуваються підвищені вимоги щодо доступності, масштабованості, безпеки та керованості. Будь-яке оновлення функціональності, виправлення помилок або зміна конфігурації має виконуватися таким чином, щоб мінімізувати простої та ризик деградації сервісу. На практиці це означає необхідність чіткої організації всіх етапів життєвого циклу програмного забезпечення: від аналізу вимог і проєктування до розробки, тестування, розгортання, експлуатації та моніторингу [1–3].

Традиційно процес розробки будувався на основі каскадної моделі, у якій етапи виконувалися послідовно, а розгортання нових версій відбувалося відносно рідко й переважно вручну. Такий підхід був прийнятним для невеликих проєктів із низькою частотою змін, однак в умовах динамічних ринків і високих очікувань користувачів він виявився надто інертним. За ручного розгортання адміністратори виконують низку операцій (копіювання файлів на сервери, зміну конфігурацій, оновлення бази даних, перезапуск служб тощо), кожна з яких є потенційним джерелом людської помилки. Відтворити точну послідовність таких дій на іншому середовищі або в подальших релізах часто складно, що знижує відтворюваність процесу та ускладнює масштабування системи.

Ситуацію додатково ускладнює поширення сервіс-орієнтованих та мікросервісних архітектур. Замість одного монолітного застосунку організація отримує десятки або сотні окремих сервісів, кожен із власним циклом оновлень, набором залежностей та вимогами до масштабування. У таких умовах «ручні» підходи до розгортання стають не лише неефективними, а й практично неможливими. Зростає потреба в систематизованій автоматизації, стандартизації процесів та формалізації кроків розгортання, що дозволяють керовано оновлювати велику кількість компонентів системи [2–4].

Відповіддю на ці виклики стало формування підходу DevOps, орієнтованого на інтеграцію процесів розробки (Development) та експлуатації (Operations) в єдиний безперервний цикл. DevOps розглядається як поєднання культурних, організаційних та технологічних практик, метою яких є скорочення часу від ідеї до її реалізації в продуктивному середовищі, підвищення якості релізів та покращення взаємодії між командами [1, 3, 5].


Рисунок 1.1 – Узагальнена «нескінченна петля» DevOps (за матеріалами BrowserStack та ManageEngine).

Життєвий цикл DevOps зазвичай описують у вигляді «нескінченної петлі», яка включає етапи планування, розробки, збірки, тестування, релізу, розгортання, експлуатації та моніторингу з подальшим поверненням до планування. На рисунку 1.1 наведено узагальнену «нескінченну петлю» DevOps, де ліва частина (Dev) відображає етапи планування, написання коду, збірки та тестування, а права (Ops) – реліз, розгортання, експлуатацію та моніторинг [2, 5, 6].

Форма «петлі» підкреслює безперервність циклу: результати моніторингу та експлуатації повертаються на етап планування у вигляді зворотного зв’язку, що забезпечує постійне вдосконалення системи. DevOps-команди відстежують повний життєвий цикл – від планування та розробки до експлуатації й підтримки – і за потреби «зсувають» перевірки на більш ранні етапи (shift left), зменшуючи ймовірність появи критичних помилок у продуктивному середовищі [1, 5].

Однією з ключових ідей DevOps є максимальна автоматизація повторюваних дій у життєвому циклі програмного забезпечення: збірки, тестування, розгортання, налаштування інфраструктури, моніторингу та реагування на збої. Автоматизовані процеси зменшують вплив людського чинника, підвищують відтворюваність результатів та спрощують масштабування системи. У поєднанні з підходами інфраструктури як коду (Infrastructure as Code, IaC) та контейнеризацією це дозволяє описувати не лише застосунок, а й оточення його виконання у вигляді конфігураційних файлів, що зберігаються разом із кодом у системі контролю версій [7, 8].

Особливе значення для вебзастосунків має забезпечення швидкого й надійного оновлення. Бізнес-вимоги змінюються, додаються нові функції, виправляються вразливості, оптимізується продуктивність. У традиційних моделях між написанням коду та його появою в продуктивному середовищі можуть проходити тижні або місяці. DevOps-підхід у поєднанні з практиками безперервної інтеграції та безперервного розгортання (CI/CD) спрямований на суттєве скорочення цього інтервалу, аж до ситуації, коли зміни можуть постачатися кілька разів на день без втрати стабільності та якості [4–6].

Таким чином, сучасні підходи до розробки та розгортання вебзастосунків базуються на поєднанні гнучких методологій, DevOps-культури та технологічних рішень для автоматизації. Центральне місце в цьому поєднанні посідають практики CI/CD, які формалізують і автоматизують шлях зміни коду від коміту до продуктивного середовища.

## 1.2. Концепції безперервної інтеграції та безперервного розгортання (CI/CD)

Поняття CI/CD (Continuous Integration / Continuous Delivery або Continuous Deployment) охоплює набір практик і інструментів, що забезпечують автоматизовану збірку, тестування та розгортання програмного забезпечення. Відомі постачальники рішень (Red Hat, Atlassian, Codefresh та ін.) визначають CI/CD як автоматизований робочий процес, який замінює ручні кроки конвеєрами, що збирають, тестують і розгортають програму надійно та передбачувано [4–6, 9].

Безперервна інтеграція (Continuous Integration, CI) передбачає, що розробники часто інтегрують свої зміни до спільної гілки репозиторію, як правило кілька разів на день. Кожен коміт запускає автоматичний процес: код завантажується на сервер CI, виконується збірка, запускаються модульні та, за потреби, інтеграційні тести. Якщо збірка або тести не проходять, розробники отримують оперативний зворотний зв’язок і можуть швидко виправити проблему. Такий підхід дає змогу раніше виявляти помилки інтеграції, підтримувати «завжди робочий» стан основної гілки та уникати ситуацій, коли перед релізом доводиться інтегрувати великі обсяги накопичених змін [4, 6, 9].

Безперервна доставка (Continuous Delivery) розширює CI, додаючи автоматизовану підготовку змін до релізу. Згідно з рекомендаціями Atlassian, після успішної збірки та тестування кожна зміна автоматично потрапляє до середовища тестування або проміжного середовища, а саме розгортання в продуктивне середовище може виконуватися за ініціативою команди, але

«одним натисканням кнопки» [5, 10, 11]. Це означає, що програмний продукт технічно готовий до релізу в будь-який момент, а рішення про фактичне розгортання приймається з урахуванням бізнес-контексту.

Безперервне розгортання (Continuous Deployment) робить наступний крок: усі зміни, що пройшли автоматизовані тести й перевірки, можуть автоматично розгортатися в продуктивному середовищі без ручного втручання. У такому підході сам факт успішного проходження пайплайну є достатньою умовою для того, щоб зміна стала доступною користувачам. Це потребує високого рівня дисципліни в написанні тестів, надійного моніторингу й добре продуманої стратегії відкату, але дозволяє суттєво прискорити постачання функціональності та підвищити регулярність релізів [4, 6, 10].

Узагальнена структура CI/CD-пайплайну часто описується як послідовність чотирьох основних стадій: source – build – test – deploy. У низці оглядових матеріалів Codefresh, Red Hat, Palo Alto Networks та інших постачальників рішень виділяють саме ці фази як базові для організації процесу [6, 8, 9, 12].

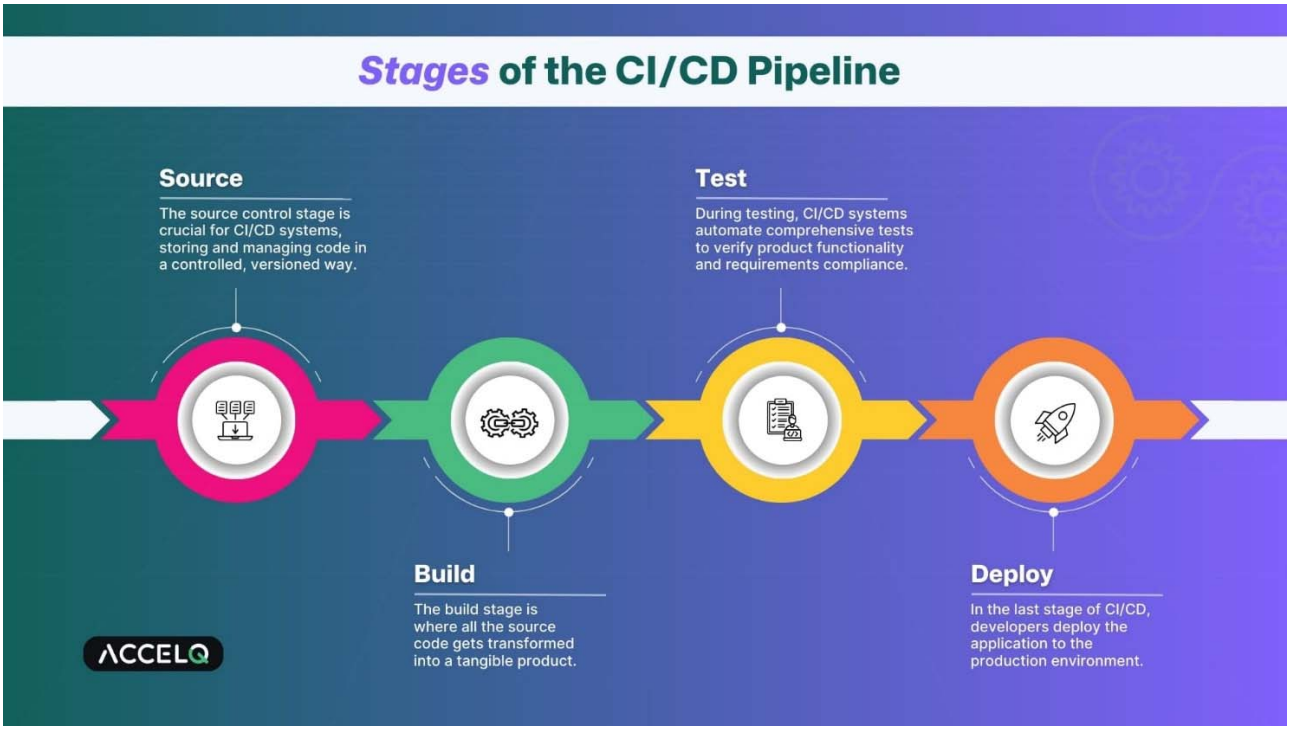


Рисунок 1.2 – Типові стадії CI/CD-пайплайну: source, build, test, deploy (за матеріалами AccelQ).

На рисунку 1.2 наведено схему такого пайплайну: на вході виступає система контролю версій (source), далі відбувається автоматизована збірка артефактів (build), після чого запускаються різні типи тестів (test), а на завершальному етапі здійснюється розгортання в цільові середовища (deploy).

Кожну зі стадій можна деталізувати. На стадії source ключову роль відіграють практики роботи з гілками (Git-flow, Trunk-based development тощо), політики злиття змін (pull/merge-requests, code review), правила іменування комітів. Саме на цьому етапі закладається прозорість історії змін та можливість автоматично визначати, які саме сценарії тестування й розгортання потрібно запускати. Інтеграція систем керування версіями з платформами CI/CD (GitLab CI/CD, GitHub Actions, Bitbucket Pipelines тощо) дозволяє запускати пайплайни при певних подіях – коміті, створенні запиту на злиття, додаванні тега релізу [10, 13, 14].

На стадії build відбувається компіляція коду, пакування застосунку та формування артефактів, придатних для розгортання. Для вебзастосунків це можуть бути архіви, виконувані пакети, Docker-образи, статичні збірки фронтенду тощо. Важливо забезпечити детермінованість і відтворюваність процесу збірки: використання тих самих версій залежностей, стандартних build-інструментів (Maven, Gradle, npm, Webpack тощо), механізмів кешування та конфігурацій, що описані у вигляді коду [6, 9, 15].

Стадія test є критичною з точки зору довіри до пайплайну. У добре спроєктованому CI/CD-процесі тести організовані як «піраміда»: більшість складають швидкі модульні тести, що виконуються при кожному коміті, а на вищих рівнях розташовані інтеграційні, системні, регресійні та навантажувальні тести, які запускаються на визначених етапах або за розкладом. Окремим напрямом є автоматизовані перевірки безпеки: статичний аналіз коду, сканування залежностей на вразливості, аналіз контейнерних образів та конфігурацій [4, 12, 16]. У сукупності це дозволяє значною мірою «зрушити перевірки вліво» (shift left), тобто виявляти проблеми ще на ранніх стадіях розробки.

На стадії deploy артефакти, що пройшли всі перевірки, розгортаються у тестові, проміжні (staging) та продуктивні середовища. Тут застосовуються різні стратегії: rolling-оновлення, blue-green-розгортання, canary-релізи. Rolling-оновлення передбачає поступове оновлення груп серверів; blue-green-підхід – підтримання двох ідентичних інфраструктур, між якими перемикається трафік; canary-релізи – попереднє розгортання для обмеженої частини користувачів із подальшим масштабуванням за відсутності проблем [6, 12].

На рисунку 1.3 показано приклад розширеної CI/CD-схеми, де до основних етапів додано елементи безпеки (DevSecOps): статичний аналіз коду, сканування залежностей, тестування безпеки, контроль відповідності політикам. Такий підхід дозволяє інтегрувати заходи безпеки у кожний етап життєвого циклу, а не розглядати їх як окремий завершальний етап [12, 16].

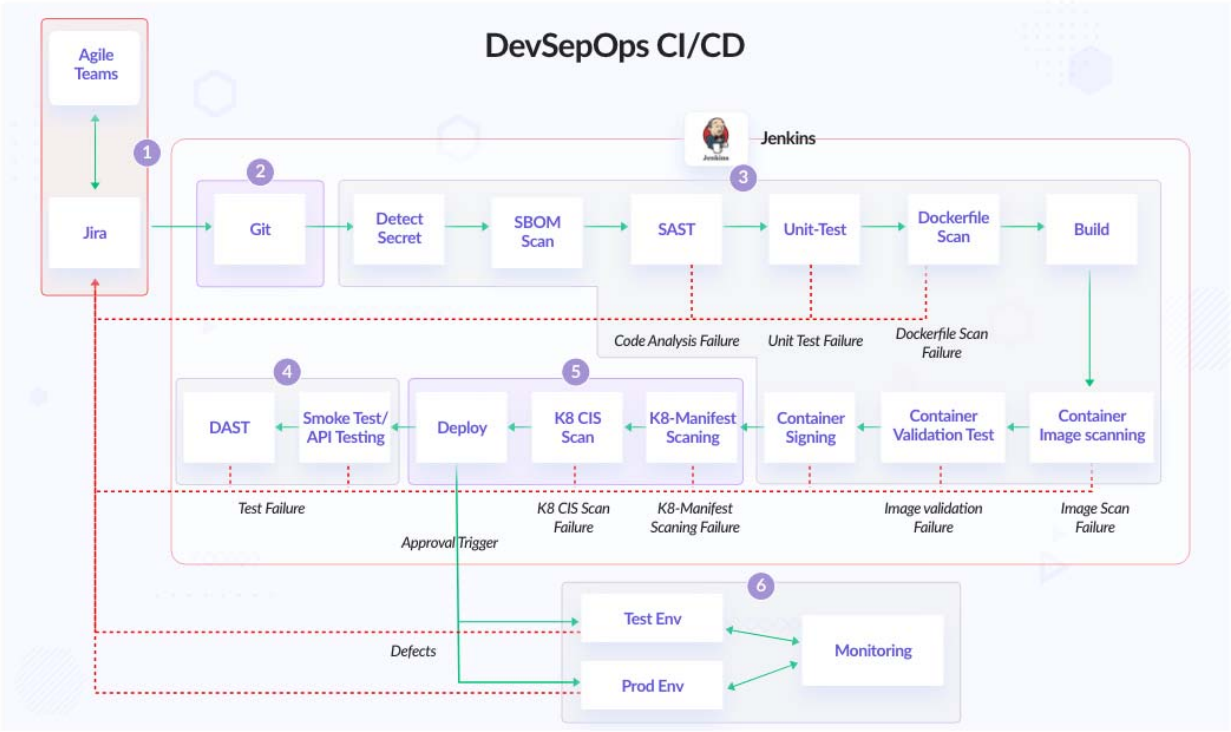


Рисунок 1.3 – Приклад інтеграції етапів безпеки в CI/CD-процес (DevSecOps-підхід) [9].

У результаті CI/CD виступає не лише як набір інструментів, а як цілісний процес, який формалізує шлях будь-якої зміни від її появи в репозиторії до розгортання в продуктивному середовищі. Ключовими властивостями такого

процесу є повторюваність (можливість багаторазово виконувати ті самі кроки з однаковим результатом), прозорість (наявність журналів виконання, артефактів, звітів тестів), керованість ризиками (використання стратегій розгортання й механізмів відкату) та масштабованість (здатність обробляти зростаючі обсяги змін і кількість середовищ) [4–6, 9].

### 1.3. Інструментальні засоби та типові архітектури CI/CD-процесів

Реалізація описаних вище принципів CI/CD на практиці ґрунтується на використанні комплексу інструментальних засобів. Їх можна умовно поділити на кілька груп: системи контролю версій, платформи CI/CD, сховища артефактів та контейнерних образів, системи управління інфраструктурою, а також засоби моніторингу і логування.

Базою для будь-якого CI/CD-процесу є система контролю версій, найпоширенішою з яких є Git. Популярні хостинги Git-репозиторіїв – GitHub, GitLab, Bitbucket одночасно виступають точками інтеграції для CI/CD-інструментів і часто мають власні вбудовані платформи для організації пайплайнів. Наприклад, GitLab CI/CD дозволяє описувати конвеєри у файлі .gitlab-ci.yml, який зберігається в корені репозиторію, а GitHub Actions використовує конфігураційні файли у каталозі .github/workflows [10, 11, 13, 14].

Серед універсальних CI-серверів важливе місце займає Jenkins. Він має розгалужену систему плагінів, підтримує різноманітні середовища виконання та сценарії збірки, однак вимагає від команди значних зусиль з налаштування й супроводу. Офіційна документація визначає Jenkins як відкритий сервер автоматизації, який може використовуватися як простий сервер безперервної інтеграції або як центр безперервної доставки для будь-якого проєкту [17–19]. У сучасних порівняннях Jenkins часто протиставляють «хмарним» рішенням GitHub Actions, GitLab CI, CircleCI, які зменшують адміністративне навантаження на команду завдяки керованій інфраструктурі.

Загалом у практиці можна виділити кілька моделей побудови CI/CD-інфраструктури: самохостинг (on-premises), коли організація розгортає Jenkins, GitLab Runner або інший сервер CI/CD у власній інфраструктурі; хмарні CI/CD-сервіси, де інфраструктура керується постачальником; гібридні варіанти, коли керування пайплайнами здійснюється в хмарі, а агенти збірки розташовані у внутрішній мережі для доступу до приватних ресурсів [4, 6, 10].

На архітектурному рівні типовий CI/CD-процес включає такі компоненти:

- Git-репозиторій, що зберігає вихідний код, конфігурації, маніфести інфраструктури як коду;
- сервер або сервіс CI/CD, який отримує події з репозиторію (push, merge-request, tag), запускає пайплайни та керує чергою завдань;
- агенти збірки (runners), що безпосередньо виконують кроки пайплайну: збірка, тестування, сканування безпеки, формування артефактів;
- сховище артефактів, де зберігаються результати збірки (бінарні файли, архіви, контейнерні образи);
- цільові середовища розгортання (тестові, staging, продуктивні кластери або сервери);
- системи моніторингу й логування, що збирають метрики та журнали, дозволяючи оцінювати стан як застосунку, так і самого CI/CD-процесу [6, 8, 12, 20].

На рисунку 1.4 представлено узагальнену архітектуру CI/CD для контейнеризованого вебзастосунку. Після коміту в Git-репозиторій запускається процес збірки контейнерного образу, його завантаження в реєстр (наприклад, Docker Hub або GitLab Container Registry), після чого сценарії CD оновлюють відповідні об’єкти розгортання (Deployment) у кластері Kubernetes за допомогою Helm-чартів або YAML-маніфестів [6, 8, 20].

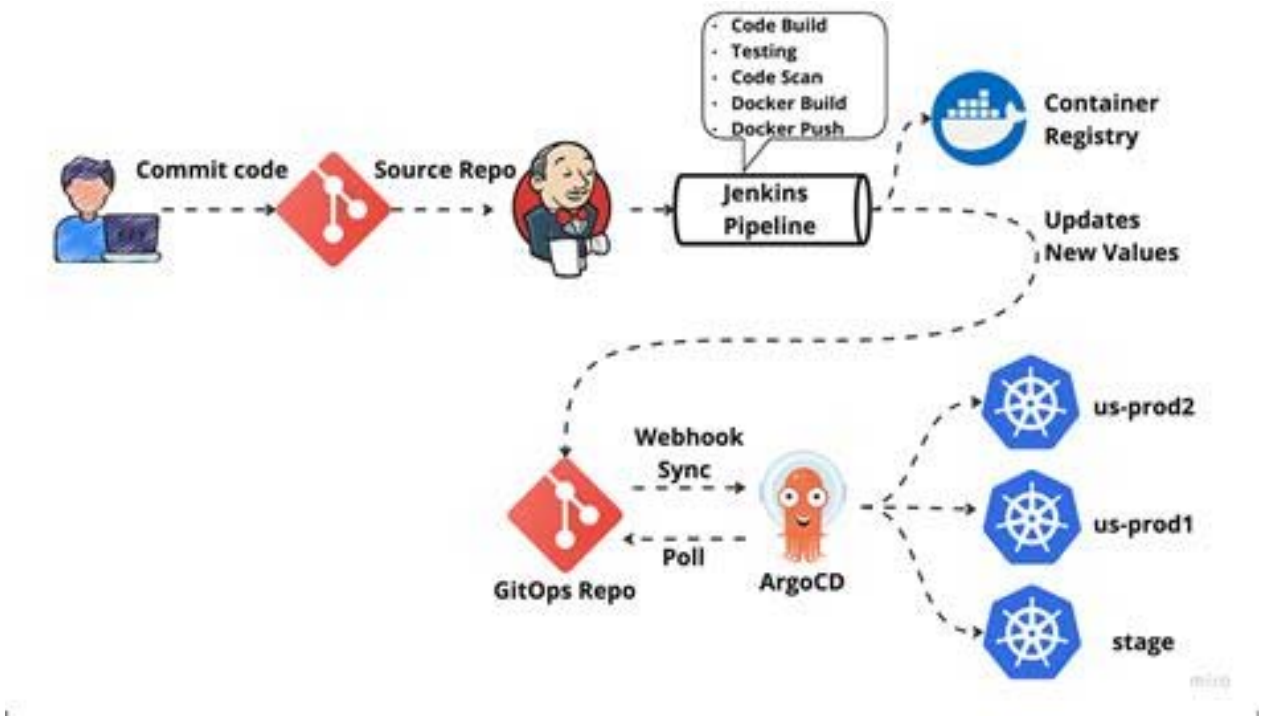


Рисунок 1.4 – Узагальнена архітектура CI/CD-процесу для контейнеризованого вебзастосунку (за матеріалами Devtron і Plural).

Важливою тенденцією є використання інфраструктури як коду (IaC) для опису не лише застосунку, а й серверів, мереж, сховищ, балансувальників навантаження. Інструменти на кшталт Terraform, AWS CloudFormation, Ansible дозволяють створювати й оновлювати інфраструктуру шляхом застосування декларативних конфігурацій, які також можуть бути інтегровані в CI/CD-пайплайни. Офіційна документація Terraform визначає цей інструмент як засіб для безпечного й ефективного побудування, зміни та версіонування інфраструктури як у хмарі, так і в локальних середовищах [7, 21, 22]. Це забезпечує узгодженість середовищ, спрощує розгортання нових екземплярів системи та зменшує ризик «дрейфу конфігурацій» між ними.

У контексті безпеки дедалі більшого поширення набуває підхід DevSecOps, за якого механізми захисту інтегруються в кожен етап життєвого циклу – від аналізу коду до моніторингу продуктивного середовища. На схемах DevSecOps-пайплайнів, подібних до показаної на рисунку 1.3, окремими етапами виділяють статичний аналіз, сканування залежностей, тестування на проникнення, контроль відповідності нормативним вимогам. Оглядові матеріали

Palo Alto Networks та інших постачальників підкреслюють важливість CI/CD-безпеки для запобігання вразливостям у процесі постачання ПЗ [12, 16].

Окрему увагу необхідно приділити інтеграції CI/CD з системами моніторингу й спостережуваності (observability). Збір метрик (latency, error rate, throughput), логів і трас дозволяє не лише контролювати стан продуктивного середовища, а й аналізувати якість самого CI/CD-процесу: час збірки, тривалість тестів, частоту невдалих розгортань, середній час відновлення після збоїв (MTTR). На основі цих даних можна приймати обґрунтовані рішення щодо оптимізації пайплайнів, додавання або вилучення тестів, зміни стратегії розгортання [4, 6, 9].

Підсумовуючи вищесказане , інструментальні засоби CI/CD утворюють багатокомпонентну екосистему, де кожен елемент – від Git-репозиторію до системи моніторингу – відіграє свою роль. Вибір конкретних інструментів (Jenkins, GitLab CI, GitHub Actions, Kubernetes, Terraform тощо) залежить від вимог проєкту, обмежень інфраструктури, компетенцій команди та ліцензійної політики. У наступних розділах роботи буде обґрунтовано вибір інструментарію для побудови CI/CD-процесу конкретного вебзастосунку та продемонстровано практичну реалізацію описаних підходів.

### 1.4. Постановка завдання

Проведений аналіз сучасних підходів до розробки й розгортання вебзастосунків, а також огляд концепцій та інструментальних засобів CI/CD показують, що ключовою проблемою є побудова такого CI/CD-процесу, який би, з одного боку, був достатньо універсальним і відтворюваним, а з іншого – враховував специфіку конкретного вебзастосунку та інфраструктури його розміщення. У практичній діяльності команд розробки часто зустрічаються ситуації, коли існуючі пайплайни побудовані фрагментарно, не покривають всі етапи життєвого циклу або не мають формалізованих критеріїв оцінювання ефективності.

У межах даної кваліфікаційної роботи як об’єкт розглядається типовий вебзастосунок із клієнт-серверною архітектурою, що функціонує у мережі Інтернет та розгортається у віртуалізованому чи контейнеризованому середовищі. Конкретний стек технологій (мови програмування, фреймворки, система керування базами даних) не є визначальним для теоретичних положень, проте в подальших розділах буде обрано й обґрунтовано реальний технологічний набір для демонстрації роботи CI/CD-процесу. Вихідними даними для побудови процесу є структура репозиторію вихідного коду, вимоги до середовищ розгортання (тестове, проміжне, продуктивне), обмеження на час недоступності сервісу при оновленнях і вимоги до безпеки.

Метою даної роботи є формальне визначення того, який саме CI/CD-процес необхідно спроєктувати та реалізувати в рамках роботи. Необхідно розробити та описати такий пайплайн, який забезпечуватиме:

- ✓ автоматичний запуск збірки та тестування при кожному внесенні змін у репозиторій вебзастосунку;
- ✓ формування відтворюваних артефактів (збірок або контейнерних образів) із можливістю їх зберігання у централізованому сховищі;
- ✓ автоматизоване розгортання вебзастосунку на тестове та проміжне середовища з виконанням додаткових перевірок працездатності;
- ✓ кероване розгортання в продуктивне середовище з мінімізацією простоїв і можливістю швидкого відкату до попередньої стабільної версії;
- ✓ збір та аналіз показників, що характеризують ефективність процесу (час збірки, час розгортання, частота невдалих релізів, середній час відновлення після збоїв).

Очікуваним результатом розв’язання поставленого завдання є спроєктований та реалізований CI/CD-процес для вибраного вебзастосунку, описаний у вигляді структурних схем, конфігураційних файлів та експериментальних даних.

# РОЗДІЛ 2
# ОБҐРУНТУВАННЯ, ВИБІР ТА РЕАЛІЗАЦІЯ ІНСТРУМЕНТАРІЮ ВИРІШЕННЯ ЗАДАЧІ

## 2.1 Вимоги до інструментарію побудови CI/CD процесу для вебзастосунків

Побудова CI/CD процесу для автоматизації розгортання вебзастосунків передбачає вибір такого інструментарію, який здатний забезпечити безперервну інтеграцію змін у вихідному коді, автоматичне виконання тестів, підготовку артефактів збірки та їх доставку у цільові середовища експлуатації. У сучасних умовах швидкого оновлення програмного забезпечення критично важливо мінімізувати частку ручної праці в цих операціях та гарантувати відтворюваність результатів. Відповідно до офіційної документації Jenkins, типовий цикл побудови, тестування і розгортання може бути повністю описаний у вигляді автоматизованого конвеєра, що виконується сервером автоматизації без участі користувача [1].

У контексті кваліфікаційної роботи розглядається вебзастосунок, для якого характерні часті зміни бізнес логіки, оновлення інтерфейсу користувача та регулярні випуски нових версій. Це зумовлює підвищені вимоги до інструментів CI/CD. Зокрема, вони повинні інтегруватися із системою контролю версій, підтримувати різні стратегії розгортання, забезпечувати ізольоване виконання збірок, гнучке керування середовищами та зручні засоби моніторингу й візуалізації процесу. Оскільки вебзастосунки нерідко експлуатуються у кількох середовищах одночасно, інструментарій має дозволяти конфігурувати окремі конвеєри для тестового, проміжного та продуктивного середовищ з можливістю повторного використання логіки збірки.

Важливою вимогою є підтримка концепції "pipeline as code" тобто опису конвеєра у вигляді конфігураційного файлу, що зберігається у репозиторії разом із вихідним кодом. Такий підхід підвищує керованість змін, дозволяє проводити

код-рев'ю сценаріїв конвеєра і забезпечує прозоре відстеження еволюції процесу збірки та розгортання. Офіційна документація Jenkins Pipeline підкреслює, що реалізація конвеєра у вигляді Jenkinsfile стає єдиним джерелом правди для всієї послідовності етапів від отримання коду до постачання артефактів користувачам [2].

Ще одним суттєвим аспектом є масштабованість рішення. У міру зростання проекту збільшується кількість комітів, розширюється набір автоматизованих тестів, ускладнюються сценарії збірки та розгортання. З огляду на це інструментарій повинен забезпечувати можливість горизонтального масштабування, розподілу навантаження між кількома вузлами та побудови кластерних конфігурацій. Документація Jenkins пропонує декілька стратегій масштабування, зокрема поділ контролерів за середовищами або за командами розробників, що дозволяє рознести навантаження та зменшити взаємний вплив конвеєрів [3].

Не менш важливими є вимоги до розширюваності та інтеграції з іншими інструментами. Типовий CI/CD процес для вебзастосунків включає побудову контейнерів, завантаження образів у реєстр, розгортання у середовищі оркестрації, статичний аналіз коду, збір метрик продуктивності та логування. Відповідно, обраний інструментарій має підтримувати підключення великої кількості плагінів та розширень, а також мати відкриті API для розробки власних модулів. Сучасні огляди Jenkins підкреслюють, що система підтримує тисячі плагінів, які покривають інтеграцію з популярними системами контролю версій, засобами тестування, платформами контейнеризації й хмарними сервісами [4].

З погляду експлуатації в організаціях, де є вимоги до інформаційної безпеки, інструмент CI/CD має надавати розвинуті механізми керування доступом, розмежування прав користувачів, безпечного зберігання облікових даних та секретів, а також підтримувати актуальні механізми автентифікації й авторизації. Jenkins як сервер автоматизації, згідно з офіційною документацією, може інтегруватися з зовнішніми службами каталогів, підтримує рольову модель

та окреме зберігання облікових даних, що дозволяє мінімізувати ризики витоку чутливої інформації [1].

Для типового вебзастосунку, який розгортається у контейнеризованому середовищі, також важливі можливості оточення, в якому виконуються конвеєри. Інструментарій має забезпечувати запуск задач у окремих агентах, здатних динамічно створюватися та знищуватися після виконання завдань, що знижує вимоги до ресурсів і підвищує ізоляцію. Документація Jenkins щодо використання агентів описує модель, у якій контролер відповідає за оркестрацію, а агентні вузли виконують безпосередні операції збірки, тестування та розгортання [5].

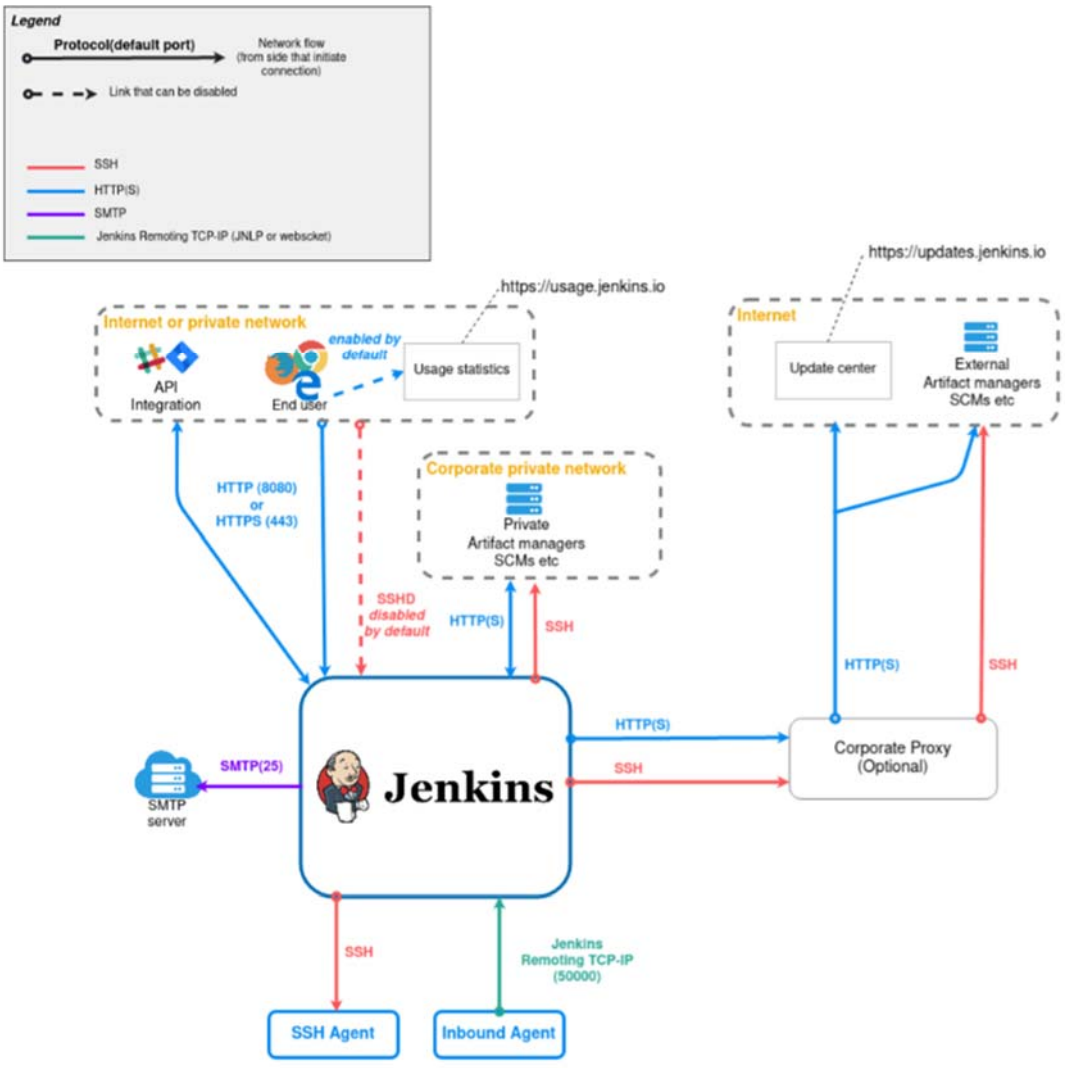


Рис. 2.1 – Схематична архітектура CI/CD для вебзастосунку на основі Jenkins

На рисунку 2.1 схематично показано узагальнену архітектуру CI/CD для вебзастосунку, у межах якої центральний сервер автоматизації взаємодіє з системою контролю версій, зовнішніми сервісами аналізу якості коду, реєстром контейнерів та цільовими середовищами розгортання. Для побудови власного рисунка доцільно орієнтуватися на типові схеми, які використовуються у навчальних матеріалах з Jenkins та DevOps і демонструють послідовність етапів Build, Test та Deploy [7].

З урахуванням зазначених вимог можна сформулювати узагальнений профіль інструменту CI/CD для даної кваліфікаційної роботи. По-перше, він має бути вільно розповсюджуваним, бажано з відкритим вихідним кодом, щоб не обмежувати навчальні та наукові експерименти. По-друге, має підтримувати роботу в різних середовищах, як на окремих фізичних або віртуальних серверах, так і в контейнерах. По-третє, повинен забезпечувати потужний механізм опису конвеєра та можливість візуального контролю за його виконанням. Саме на відповідність цим критеріям надалі оцінювалися альтернативні рішення та обґрунтовувався вибір Jenkins як базового інструменту.

## 2.2 Обґрунтування вибору Jenkins як базового інструменту CI/CD

Серед популярних засобів побудови CI/CD процесів сьогодні використовуються як хмарні рішення, інтегровані із конкретними платформами або сервісами, так і універсальні сервери автоматизації. До першої групи належать, наприклад, GitHub Actions, GitLab CI/CD, Azure DevOps Pipelines. Їхньою перевагою є тісна інтеграція з відповідними екосистемами, однак це часто супроводжується прив’язкою до конкретного провайдера, обмеженнями безкоштовних тарифів та меншим контролем над середовищем виконання. Друга група представлена, зокрема, Jenkins, який позиціонується як незалежний сервер автоматизації із відкритим вихідним кодом та розвиненою системою розширень [1], [6].

Для потреб кваліфікаційної роботи важливими були можливість розгортання інструменту в умовах локальної інфраструктури навчального або виробничого підрозділу, гнучкість налаштування архітектури конвеєрів під конкретні сценарії та наявність великої кількості навчальних матеріалів. Саме Jenkins, згідно з оглядами та аналітичними статтями, залишається одним з найбільш розповсюджених серверів CI/CD, який широко використовується у промислових та освітніх проектах [5], [12].

З погляду функціональності, Jenkins надає повний набір засобів для побудови конвеєрів. Концепція Jenkins Pipeline, детально описана у офіційній документації, дозволяє моделювати прості й складні процеси доставки програмного забезпечення, використовуючи декларативний або скриптовий підхід [2]. У декларативному варіанті структура конвеєра задається у вигляді блоку pipeline із переліком етапів Build, Test, Deploy, а у скриптовому, заснованому на Groovy, розробник має максимальну гнучкість при описі умов виконання та повторного використання коду [2], [9].

Важливою перевагою Jenkins є можливість масштабування архітектури завдяки використанню контролерів та агентів. Як зазначається у спеціалізованих оглядах, Jenkins реалізує модель master agent, в якій центральний вузол відповідає за планування та координацію задач, а агентні вузли виконують безпосередні операції збірки та тестування [3], [11]. Це дозволяє розподіляти навантаження, використовуючи різні конфігурації апаратних ресурсів для різних типів завдань, наприклад виділяти окремі агенти для ресурсомістких інтеграційних тестів або збірки контейнерів.

Ще один аргумент на користь Jenkins полягає у його розширюваності. Офіційний каталог плагінів містить понад дві тисячі розширень, які покривають інтеграцію із системами контролю версій, інструментами аналізу якості коду, платформами контейнеризації, хмарними сервісами та засобами моніторингу [4]. Таке розмаїття дозволяє будувати конвеєри, які включають, наприклад, підключення до Git, автоматичне виконання тестів за допомогою фреймворків для конкретних мов програмування, побудову Docker образів, їх завантаження

до реєстру та подальше розгортання у Kubernetes або на віртуальних серверах. Для кваліфікаційної роботи це означає можливість реалізувати повний життєвий цикл вебзастосунку в межах одного інструменту, не обмежуючись базовим набором функцій.

Не менш вагомим фактором є активна спільнота та велика кількість навчальних ресурсів. Підтримка Jenkins здійснюється як офіційною командою, так і широкою спільнотою розробників, що забезпечує регулярні оновлення, виправлення вразливостей, появу нових плагінів та керівництв. Наявність офіційних інтерактивних турів, таких як "Getting started with the Guided Tour" та різноманітних покрокових інструкцій щодо побудови конвеєрів для різних технологічних стеків значно полегшує включення Jenkins до навчальних програм [2], [10], [11].

З погляду розгортання, Jenkins є досить гнучким. Його можна встановити як окремий сервіс на фізичному або віртуальному сервері, розгорнути за допомогою контейнерів або скористатися готовими шаблонами у хмарних платформах. Наприклад, сучасні хостинги пропонують готові конфігурації Jenkins, де всі базові налаштування виконані заздалегідь, а користувачеві залишається зосередитися на створенні конвеєрів [14]. Для освітніх цілей це дозволяє варіювати сценарії використання Jenkins від локального розгортання у лабораторії до девопс лабораторій у хмарній інфраструктурі.

У контексті аналізу архітектури Jenkins доцільно використати узагальнену схему, що демонструє взаємодію сервера Jenkins із агентами та зовнішніми сервісами. На рисунку 2.2 подано варіант такої діаграми, адаптований на основі матеріалу Octopus Deploy, де центральним елементом виступає Jenkins Server з підсистемами керування завданнями, плагінами та конфігураціями, а по периметру розташовано агенти, системи контролю версій, сховища артефактів і зовнішні системи розгортання [15]. На схемі чітко відображено, що Jenkins Server приймає події з репозиторію коду, ставить задачі у чергу, розподіляє їх між підключеними агентами, отримує від них результати виконання збірок і тестів, а також інтегрується з іншими інструментами, такими

як реєстр контейнерів або платформи оркестрації. Такий варіант подання архітектури дозволяє наочно показати роль Jenkins як центру автоматизації процесів CI/CD і підкреслити місце агентів у загальній схемі, що є принципово важливим для подальшого проєктування конвеєра у межах даної роботи.

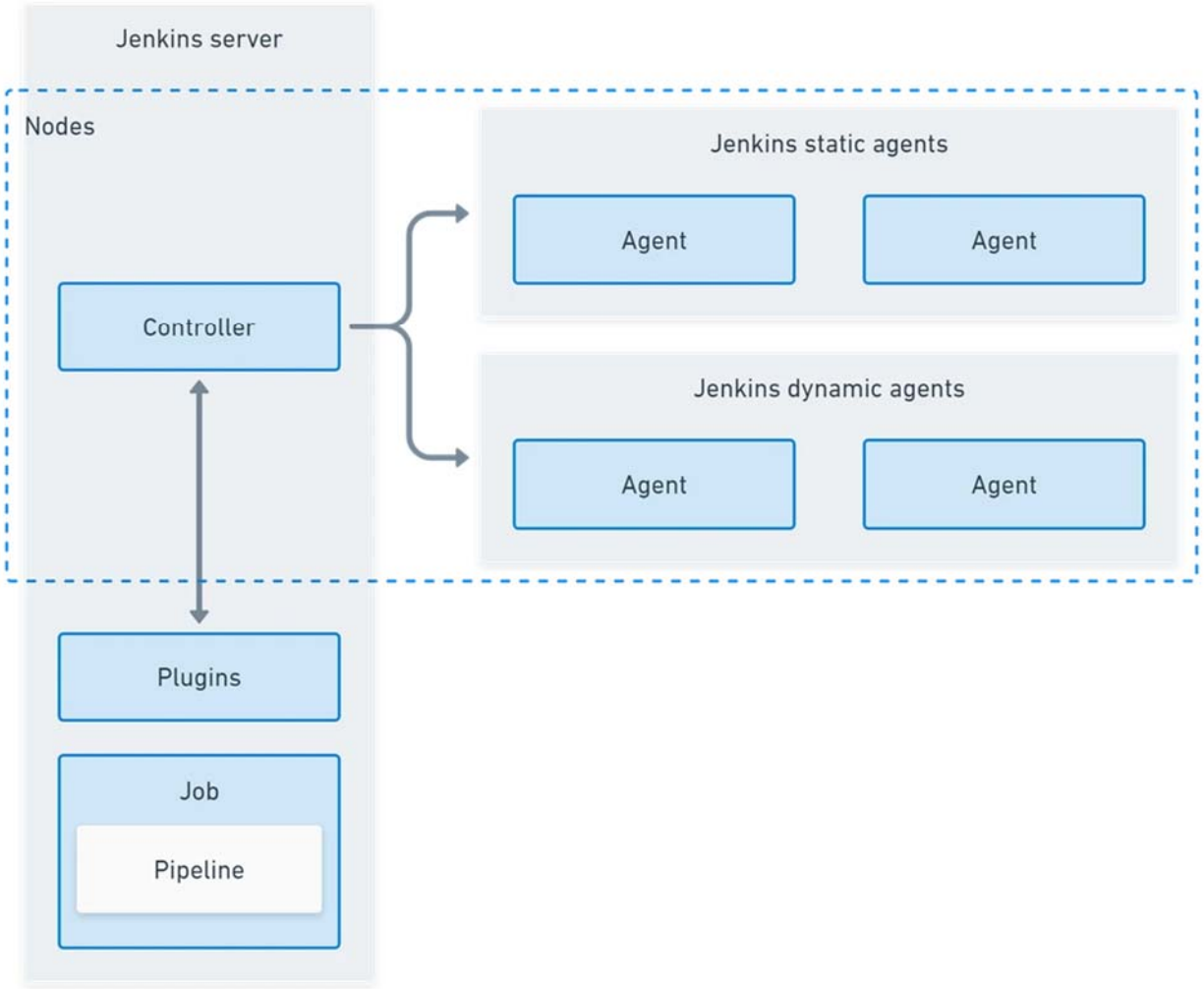


Рисунок 2.2. – Узагальнена архітектура Jenkins з контролером та статичними й динамічними агентами

Узагальнюючи наведене, можна зробити висновок, що Jenkins задовольняє ключові вимоги до інструментарію, сформульовані у підрозділі 2.1. Він є відкритим сервером автоматизації, підтримує опис конвеєра у вигляді коду, надає розвинуті засоби масштабування та інтеграції з іншими компонентами інфраструктури, а також супроводжується широкою базою знань. Це дозволяє

обґрунтовано обрати Jenkins як основу для побудови CI/CD процесу автоматизованого розгортання вебзастосунків у межах даної кваліфікаційної роботи.

## 2.3 Проєктування архітектури CI/CD конвеєра на основі Jenkins

Після обґрунтування вибору Jenkins необхідно перейти до проєктування конкретної архітектури CI/CD конвеєра для цільового вебзастосунку. На логічному рівні конвеєр можна описати як послідовність етапів, що починаються із отримання вихідного коду з репозиторію та закінчуються розгортанням перевіреного артефакту у продуктивному середовищі. Офіційна документація Jenkins Pipeline рекомендує мінімально виділяти етапи Build, Test та Deploy, доповнюючи їх за потреби додатковими перевірками, аналізом якості коду, скануванням безпеки та іншими активностями [2], [7].

У контексті логічного проєктування конвеєра доцільно опиратися на узагальнені схеми, які демонструють послідовність базових етапів життєвого циклу застосунку. На рисунку 2.3 наведено адаптований варіант діаграми, де конвеєр Jenkins подано у вигляді ланцюжка взаємопов’язаних стадій Build, Test і Deploy [7]. У цій схемі етап Build відповідає за формування артефактів на основі вихідного коду, етап Test охоплює автоматизовані модульні та інтеграційні тести, а етап Deploy забезпечує розгортання перевіреної версії вебзастосунку у цільове середовище. Такий варіант візуалізації дозволяє чітко виділити ключові точки контролю в конвеєрі, показати послідовність переходу артефактів між стадіями та продемонструвати, як Jenkins реалізує безперервну інтеграцію і доставку змін. У подальшій реалізації конвеєра, описаній у даній роботі, саме така логіка поділу на етапи використовується як концептуальна основа для побудови Jenkinsfile і налаштування окремих кроків збірки, тестування та розгортання.

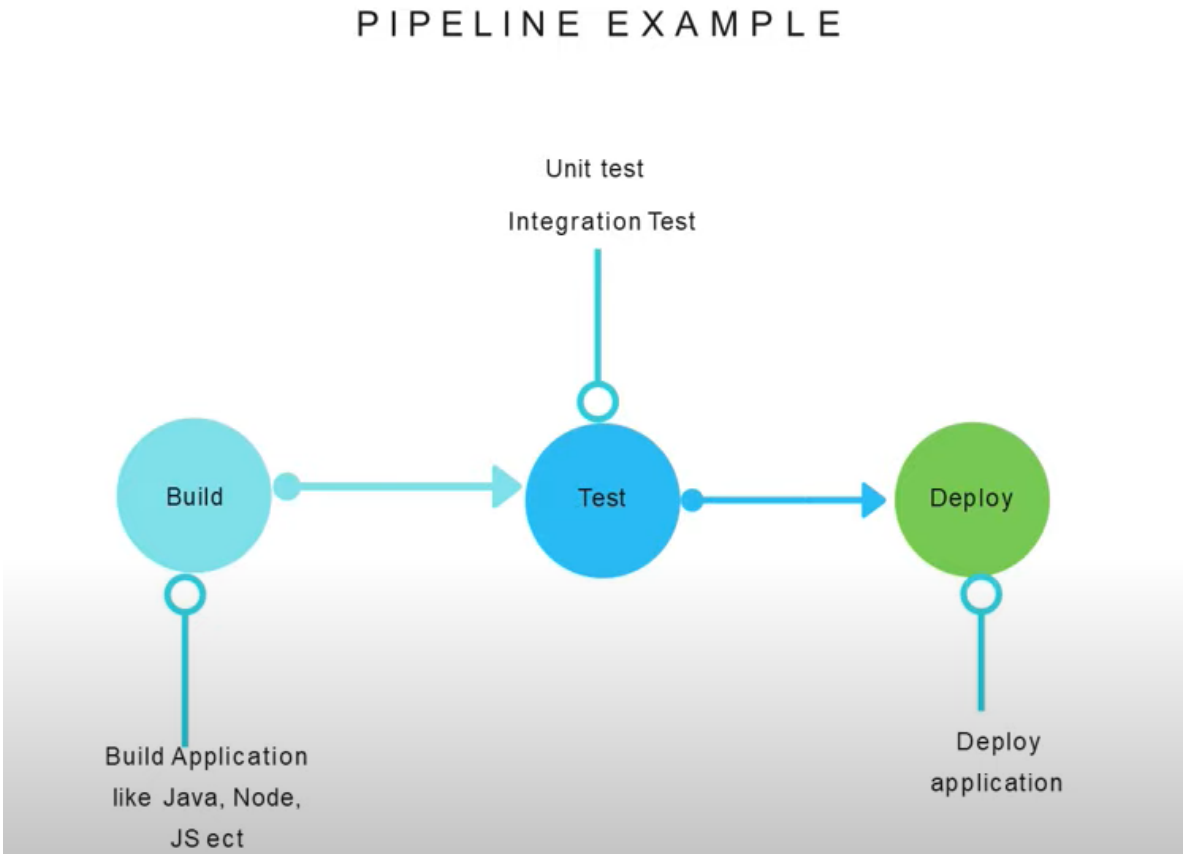


Рисунок 2.3. – Узагальнений приклад конвеєра Jenkins з етапами Build, Test і Deploy (на основі Earthly)

Структурно архітектура рішення передбачає виокремлення кількох логічних рівнів. Перший рівень становить система контролю версій, у якій зберігається вихідний код вебзастосунку та файл Jenkinsfile. Другий рівень представлений сервером Jenkins, що виконує роль контролера. Третій рівень формують агенти, на яких безпосередньо виконуються етапи конвеєра. Четвертий рівень складають зовнішні сервіси, такі як контейнерний реєстр, система оркестрації контейнерів, сховище артефактів або віддалений веб сервер. Подібну багаторівневу модель описано в офіційній документації Jenkins щодо використання агентів та в матеріалах, присвячених масштабуванню й архітектурі контролерів [5], [8], [11].

Особливу увагу слід приділити вибору стратегії використання агентів. Для навчального або невеликого виробничого середовища доцільно застосовувати динамічних агентів, що запускаються у контейнерах, наприклад на основі Docker образів. У цьому випадку кожне виконання конвеєра отримує

власний ізольований робочий простір із наперед визначеним набором інструментів, необхідних для збірки та тестування. Після завершення задачі контейнер із агентом може бути знищений, що зменшує витрати на підтримку інфраструктури й мінімізує ризики накопичення застарілих станів середовища. Офіційні інструкції Jenkins описують використання контейнеризованих агентів як рекомендовану практику для розвантаження контролера та досягнення повторюваності збірок [5].

Проектуючи конкретний конвеєр для вебзастосунку, необхідно визначити спосіб автентифікації Jenkins до зовнішніх ресурсів. Наприклад, для доступу до приватного репозиторію Git можна використовувати SSH ключ або токен доступу, що зберігаються у сховищі облікових даних Jenkins. Для завантаження образів у реєстр контейнерів або розгортання на віддаленому сервері налаштовуються окремі облікові записи або технічні користувачі. Усі ці дані слід зберігати централізовано та використовувати через відповідні змінні середовища або параметри конвеєра, не вбудовуючи секрети безпосередньо у Jenkinsfile. Такий підхід відповідає рекомендаціям Jenkins щодо безпечного управління обліковими даними й дозволяє спростити подальшу зміну паролів або ключів [1], [23].

Для деталізації архітектури CI/CD процесу контейнеризованого вебзастосунку доцільно використати узагальнену схему, яка відображає взаємодію між системою контролю версій, сервером Jenkins, реєстром контейнерів та середовищем оркестрації. На рисунку 2.4 показано, як зміни у вихідному коді, зафіксовані у Git, автоматично запускають конвеєр Jenkins, що виконує збірку застосунку, формує Docker образ, публікує його в реєстрі контейнерів, після чого Kubernetes кластер завантажує оновлений образ і розгортає нову версію сервісу [16]. Такий спосіб візуалізації дозволяє чітко простежити шлях артефакту від моменту коміту до розгортання, підкреслити роль Jenkins як центрального оркестратора CI/CD процесу та показати місце реєстру контейнерів і Kubernetes у загальній архітектурі. У межах даної

кваліфікаційної роботи саме така логіка взаємодії компонентів використовується як концептуальна основа для побудови конвеєра розгортання вебзастосунку.

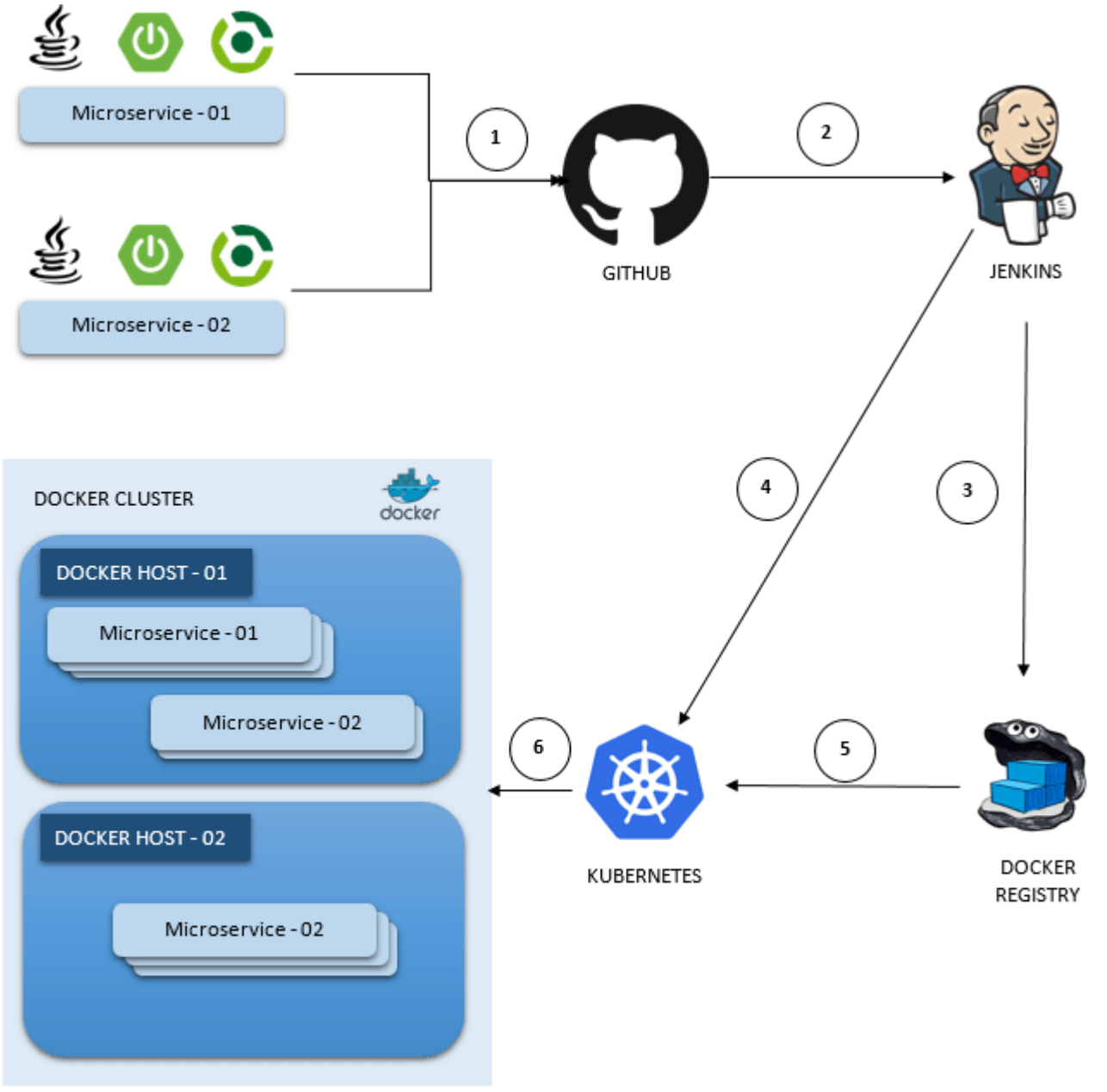


Рисунок 2.4 – Приклад архітектури CI/CD процесу для контейнеризованого вебзастосунку на основі Jenkins, Docker registry та Kubernetes

Важливим елементом проєктування є вибір між централізованим та розподіленим розміщенням конвеєрів. У невеликих проектах усі конвеєри можуть виконуватися на одному контролері, тоді як для великих систем доцільно

розділяти їх між декількома контролерами, наприклад за середовищами або командами. Документація Jenkins з масштабування описує стратегії поділу контролерів та рекомендує уникати надмірної концентрації задач на одному вузлі, оскільки це ускладнює супровід та може знизити надійність [3], [8]. У межах кваліфікаційної роботи використовується відносно проста конфігурація з одним контролером, однак під час аналізу результатів будуть розглянуті можливості її масштабування.

## 2.4 Реалізація конвеєра CI/CD для автоматизації розгортання вебзастосунку

Після проєктування архітектури наступним кроком є практична реалізація конвеєра CI/CD у Jenkins. На першому етапі виконується розгортання самого Jenkins. Для цілей кваліфікаційної роботи можливо використати варіант встановлення у контейнері, що дозволяє швидко розгорнути типовий екземпляр Jenkins із попередньо налаштованою конфігурацією. Такі підходи описуються у сучасних посібниках, присвячених розгортанню Jenkins у хмарному середовищі або на локальному сервері, де контролер запускається як контейнер з підключеними томами для зберігання даних та плагінів [13], [21].

Далі здійснюється початкове налаштування інтерфейсу, створення адміністративного користувача, встановлення рекомендованих плагінів та підключення до системи контролю версій, у якій зберігається вихідний код вебзастосунку. Зокрема, необхідно налаштувати плагін для роботи з обраною системою, створити облікові дані доступу та визначити URL репозиторію. У рамках цієї роботи передбачається використання репозиторію, що містить як вихідний код вебзастосунку, так і файл Jenkinsfile, у якому описаний весь конвеєр CI/CD.

Створення конвеєра доцільно виконувати у два етапи. Спочатку розробляється Jenkinsfile з базовою структурою, що включає етапи Build, Test та Deploy, а потім він поступово доповнюється додатковою логікою. Офіційні

керівництва Jenkins демонструють приклади побудови такого файлу, де у блоці pipeline задається агент, що відповідає за виконання всього конвеєра, а у блоці stages визначаються окремі етапи [2], [14]. Для цілей кваліфікаційної роботи файл Jenkinsfile міститиме етап отримання коду з репозиторію, встановлення залежностей, запуску тестів, побудови артефакту (наприклад, контейнерного образу) та його розгортання у цільовому середовищі.

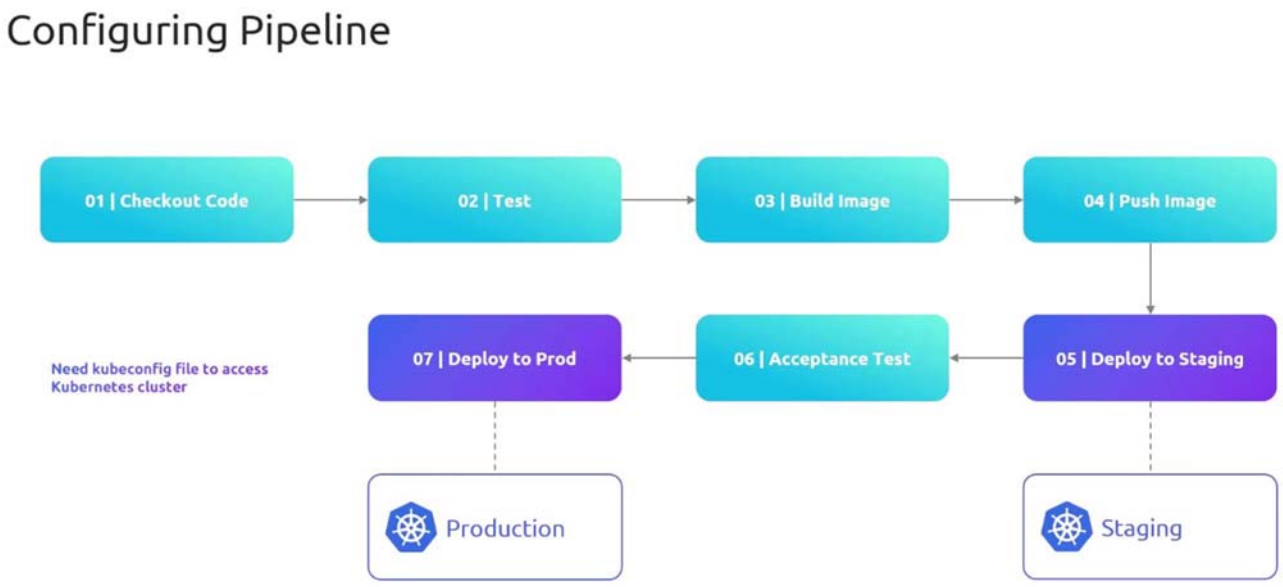


Рисунок 2.5. – Узагальнена блок-схема потоку конвеєра Jenkins від отримання коду до розгортання у тестовому та продуктивному середовищах.

Для наочного подання загальної логіки роботи розробленого конвеєра доцільно використати узагальнену блок-схему потоку виконання, яка відображає всі основні кроки від отримання змін у репозиторії до розгортання оновленої версії у продуктивному середовищі. На рисунку 2.5 наведено адаптований варіант діаграми з навчальних матеріалів KodeKloud, присвячених налаштуванню Jenkins Pipeline для Kubernetes, де показано повний потік конвеєра від операції отримання коду з системи контролю версій через етапи тестування, побудови і публікації контейнерного образу до послідовного розгортання у середовищах перевірки та промислової експлуатації [17]. На схемі послідовно відображено кроки перевірки коду, запуску автоматизованих тестів, збірки образу, його завантаження до реєстру контейнерів, розгортання у

проміжному середовищі, виконання приймальних випробувань і подальше перенесення оновленої версії у продуктивне середовище з використанням конфігурацій доступу до кластера Kubernetes. Такий варіант блок-схеми добре узгоджується з логікою проєктованого у даній роботі конвеєра на основі Jenkins і може розглядатися як прототип для побудови власної схеми, що відображає конкретні етапи збірки, тестування та розгортання цільового вебзастосунку.

На наступному кроці налаштовується тригер запуску конвеєра. Найпоширенішим варіантом є запуск при кожному коміті у визначену гілку репозиторію або при створенні запиту на злиття. Це забезпечує безперервну інтеграцію змін: кожна правка коду проходить повний цикл перевірок перед потраплянням у продуктивне середовище. Для окремих сценаріїв може знадобитися плановий запуск, наприклад нічні збірки, або ручний запуск для спеціальних гілок. Jenkins підтримує всі ці варіанти за рахунок тригерів, які конфігуруються безпосередньо у Jenkinsfile або в конфігурації проєкту [2], [19].

Важливою частиною реалізації є інтеграція з системами тестування та аналізу якості коду. Так, у межах етапу Test можуть запускатися модульні тести, генеруватися звіти у форматі JUnit, проводитися статичний аналіз коду з використанням відповідних плагінів. Jenkins надає засоби для агрегування результатів тестування, їх візуалізації та побудови історичних трендів, що дозволяє відстежувати динаміку якості проєкту [9], [19]. У кваліфікаційній роботі передбачається інтеграція з щонайменше одним інструментом аналізу якості коду та демонстрація, як результати перевірок впливають на загальний статус конвеєра.

Етап Deploy у реалізованому конвеєрі відповідає за доставку артефактів у цільове середовище. Якщо вебзастосунок контейнеризовано, на цьому етапі виконується побудова Docker образу, завантаження його до реєстру та оновлення розгортання у середовищі оркестрації або на окремому сервері. У випадку класичних вебзастосунків без контейнеризації етап Deploy може включати копіювання файлів на віддалений сервер, виконання міграцій бази даних, перезапуск служб веб-сервера та інші операції. Публікації, присвячені Jenkins як

інструменту для CI/CD, наводять різноманітні приклади реалізації етапу розгортання як для контейнеризованих, так і для традиційних сценаріїв [7], [33], [35].

У завершальній частині побудови конвеєра доцільно продемонструвати не лише його логічну структуру, а й реальний вигляд у користувацькому інтерфейсі, який використовує команда розробки під час щоденної роботи. На рисунку 2.6 наведено приклад візуалізації конвеєра Jenkins в інтерфейсі Blue Ocean, адаптований на основі офіційної документації Jenkins [9]. У цьому інтерфейсі кожен етап конвеєра подано у вигляді окремого блоку, розташованого послідовно в одному ряду, що дозволяє візуально простежити рух зміни від початкового отримання коду до фінального розгортання. Для кожного етапу відображається статус виконання, тривалість та, за потреби, доступ до детальних логів, що дає змогу швидко ідентифікувати проблемні кроки.

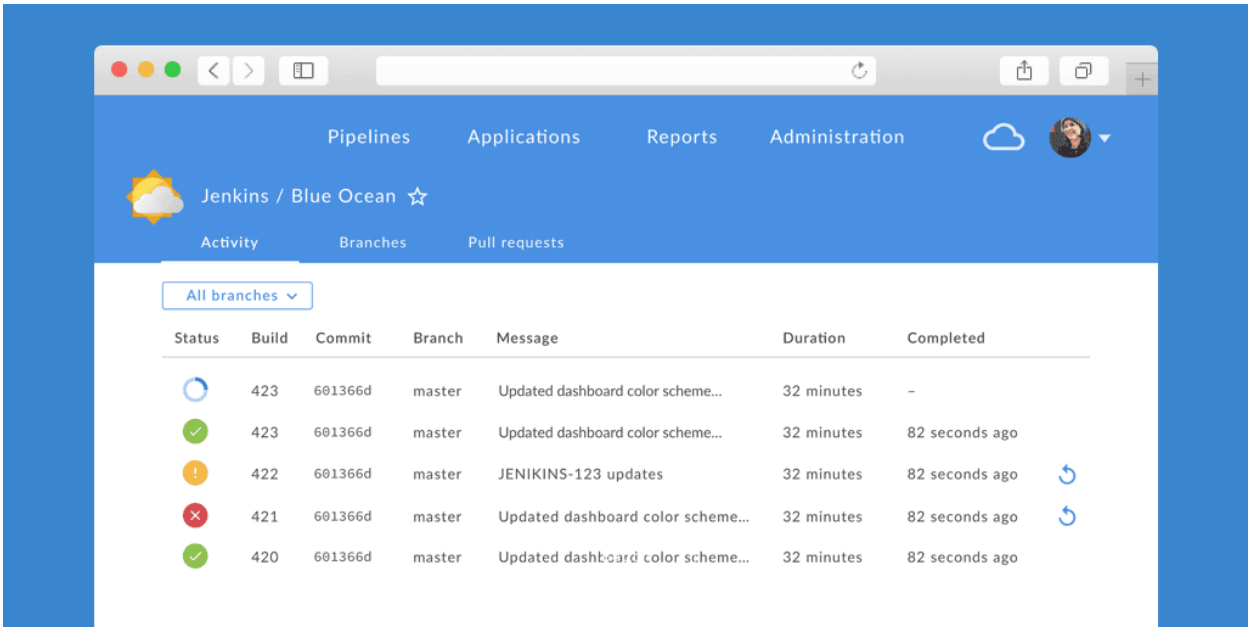


Рисунок 2.6. – Візуалізація конвеєра Jenkins у інтерфейсі Blue Ocean

Особливістю Blue Ocean є наочне групування запусків конвеєра за окремими збірками та гілками репозиторію, а також інтерактивне відображення історії виконання. Це дозволяє розробникам та інженерам з експлуатації оперативно оцінювати стабільність конвеєра, відстежувати вплив окремих змін

на результат збірок і аналізувати динаміку тривалості етапів. У контексті даної кваліфікаційної роботи використання такого виду візуалізації дає змогу продемонструвати, як створений конвеєр CI/CD для вебзастосунку виглядає у практичній експлуатації, які саме етапи він містить і яким чином Jenkins надає зворотний зв’язок щодо кожного запуску.

Завершальним кроком реалізації є налаштування механізмів моніторингу, повідомлень та збору метрик. Jenkins підтримує інтеграцію з різноманітними системами повідомлень та моніторингу, що дозволяє надсилати сповіщення про результати збірок електронною поштою, у корпоративні месенджери або системи інцидент-менеджменту. Це дає змогу оперативно реагувати на помилки конвеєра, аналізувати причини збоїв та підтримувати стабільність процесу розгортання. У межах кваліфікаційної роботи буде продемонстровано базове налаштування таких сповіщень та показано їх роль у щоденній експлуатації CI/CD конвеєра.

Розглянуті підходи до реалізації конвеєра CI/CD на основі Jenkins демонструють, що обраний інструментарій дозволяє повністю автоматизувати процес доставки вебзастосунку від моменту внесення змін у код до оновлення продуктивного середовища. У наступному розділі буде детально проаналізовано результати функціонування розробленого конвеєра, оцінено його ефективність та надійність, а також запропоновано можливі напрями подальшого вдосконалення.

# РОЗДІЛ 3.
# РОЗРОБКА ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА CI/CD-ПРОЦЕСУ ДЛЯ ВЕБЗАСТОСУНКУ

У цьому розділі змодельовано задачу, наближену до реального комерційного проєкту зі створення вебзастосунку, та поетапно реалізовано CI/CD-процес на основі Jenkins. Послідовно описано постановку задачі розгортання інфраструктури, налаштування Jenkins і реалізацію конвеєра безперервної інтеграції, тестування та розгортання.

## 3.1 Моделювання задачі та постановка вимог до CI/CD-процесу.

Для демонстрації побудови CI/CD-процесу розглядається умовний комерційний проєкт зі створення вебзастосунку, призначеного для автоматизації певних бізнес-процесів замовника. Команда проєкту складається з керівника (проєктного менеджера), п’яти розробників, двох тестувальників та одного DevOps-інженера. Така структура є типовою для середнього за масштабом проєкту розробки веборієнтованої інформаційної системи.

Замовник висуває низку вимог, які безпосередньо впливають на організацію CI/CD-процесу. По-перше, передбачається розробка застосунку мовою програмування Java з використанням стандартного стеку для веброзробки, що включає середовище виконання OpenJDK та інструмент автоматизованої збірки Maven. По-друге, загальний бюджет реалізації, включно з витратами на хмарну інфраструктуру, обмежений сумою 50 000 доларів США, а термін виконання встановлений на рівні чотирьох місяців. По-третє, замовник очікує отримувати оновлену версію вебзастосунку кожні два тижні, причому кожний інкремент має містити реалізовані функції, узгоджені на етапі планування.

Окремо підкреслюється вимога до якості: код має бути покритий автоматизованими тестами, а кількість критичних помилок і нефункціональних

компонентів повинна бути мінімальною. Це означає, що традиційний підхід із ручною збіркою та тестуванням є недостатнім, оскільки створює високі ризики зриву термінів та зростання технічного боргу. Для забезпечення регулярних двотижневих релізів із прогнозованою якістю необхідна автоматизація основних етапів життєвого циклу розробки.

На етапі планування DevOps-інженер спільно з командою аналізує вимоги замовника та формує набір технічних критеріїв до CI/CD-процесу. Визначається, що процес має підтримувати щонайменше три середовища: середовище розробки, тестове (staging) та продуктивне. Усі зміни в основній гілці репозиторію повинні автоматично проходити етапи збірки та модульного тестування. Для важливих функціональних інкрементів, які переходять у staging та продуктивне середовище, додатково передбачається інтеграційне та регресійне тестування.

З урахуванням досвіду DevOps-інженера, проаналізованих у теоретичній частині роботи вимог до інструментарію та обмежень бюджету обирається Jenkins як центральний компонент CI/CD-процесу. Таке рішення обумовлене відкритою моделлю розповсюдження, гнучкістю налаштувань, наявністю великої кількості плагінів, доброю сумісністю з Java-стеком (OpenJDK, Maven), а також можливістю розгортання в локальному або хмарному середовищі без прив’язки до конкретного провайдера. Потенційні недоліки Jenkins, пов’язані з потребою самостійної підтримки інфраструктури та складністю конфігурації плагінів, у даному випадку компенсуються наявністю у команді спеціаліста з достатнім досвідом роботи з цим інструментом.

У результаті у межах даної кваліфікаційної роботи формулюється прикладна задача: спроєктувати, розгорнути та задокументувати CI/CD-процес для умовного Java-вебзастосунку, побудований на основі Jenkins, який забезпечує автоматизовану збірку, тестування, доставку й розгортання змін у тестовому та продуктивному середовищах із періодичністю, що відповідає двотижневому циклу інкрементальної розробки.

### 3.2 Розгортання Jenkins та підготовка інфраструктури

Першим кроком практичної реалізації CI/CD-процесу є підготовка інфраструктури для розгортання Jenkins. Для цього використовується віртуальна машина в хмарному середовищі, наприклад у провайдера DigitalOcean. Обирається регіон, що забезпечує прийнятний час відгуку для цільової групи користувачів, та конфігурація, достатня для навчального і демонстраційного сценарію: один віртуальний процесор, 1 ГБ оперативної пам’яті та 25 ГБ дискового простору. Як операційну систему обрано Ubuntu 22.04 LTS (x64), яка добре задовольняє вимоги Jenkins та інструментів Java-стеку.

Послідовність створення віртуальної машини в хмарній панелі керування зображується на рисунках 3.1–3.5. На них відображено вибір операційної системи, апаратних ресурсів, методу автентифікації та підтвердження створення сервера.

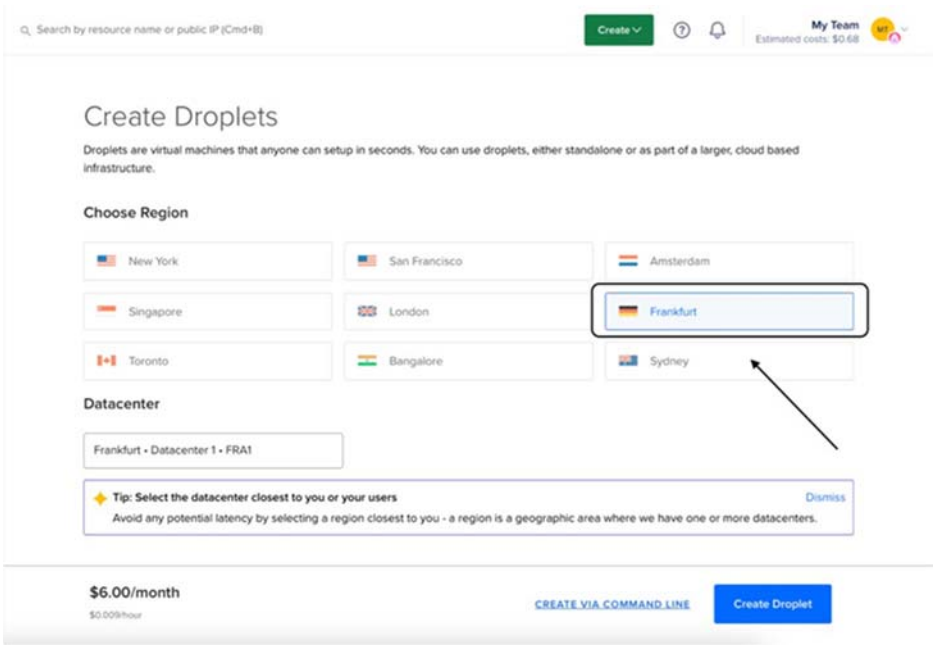


Рисунок 3.1 – Вибір початкових параметрів віртуальної машини для розгортання Jenkins

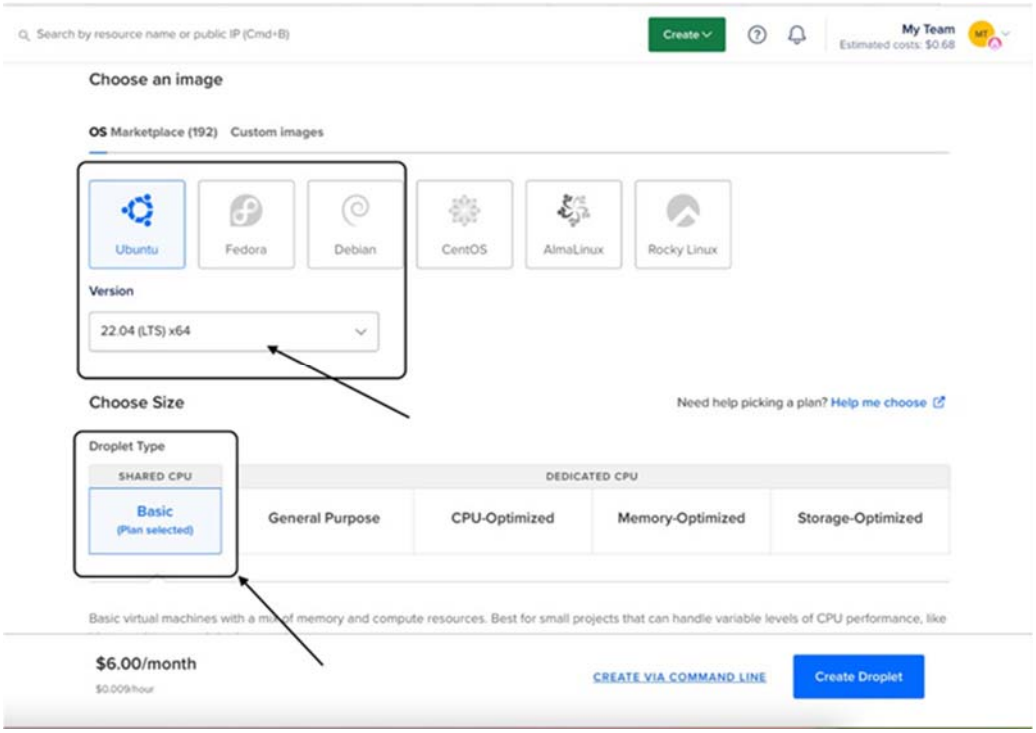


Рисунок 3.2 – Налаштування операційної системи та обсягу ресурсів віртуальної машини.

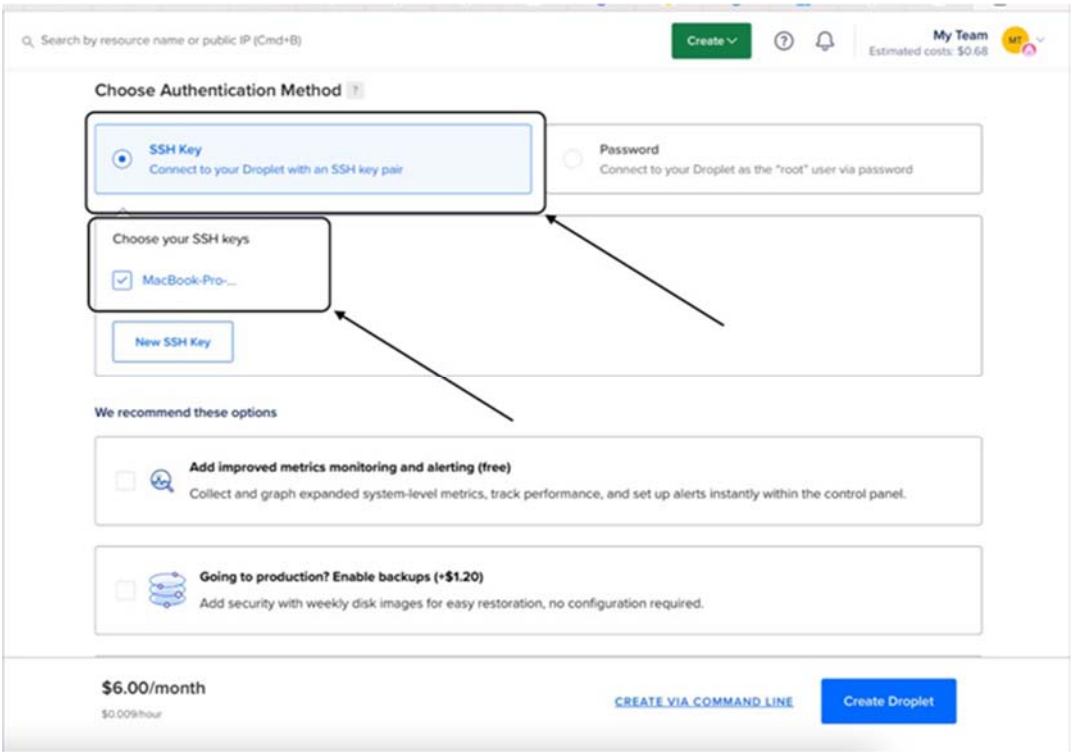


Рисунок 3.3 – Вибір методу автентифікації та завдання SSH-ключа для доступу до сервера.

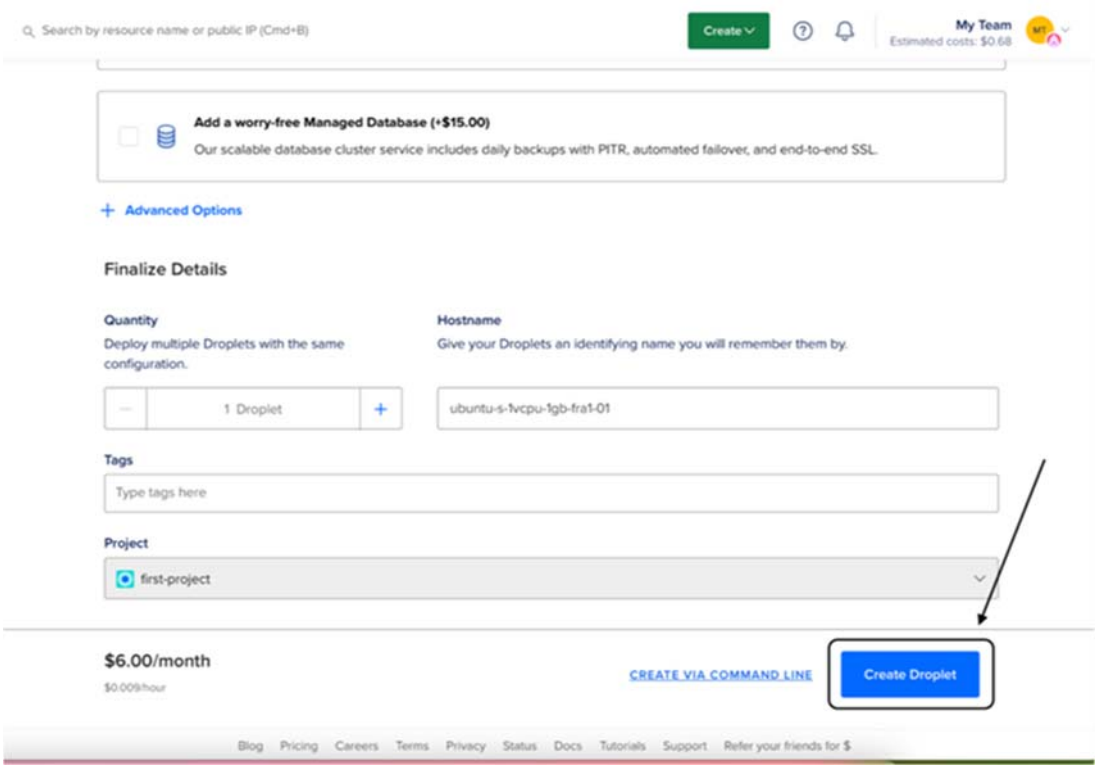


Рисунок 3.4 – Підтвердження параметрів і створення віртуальної машини в хмарному середовищі.

Для підвищення безпеки доступу до сервера використовується автентифікація на основі пари відкритий/закритий SSH-ключів. Приватний ключ зберігається виключно на клієнтській машині розробника або DevOps-інженера, тоді як відкритий ключ додається в налаштуваннях хмарного провайдера при створенні віртуальної машини.

Підключення до сервера з клієнтських ОС Linux або macOS здійснюється через термінал командою:

*ssh username@xxx.xxx.xxx.xxx -i ~/.ssh/id_rsa*

де username – ім’я користувача на сервері, xxx.xxx.xxx.xxx – публічна IP-адреса сервера, а ~/.ssh/id_rsa – шлях до приватного ключа. На клієнтських системах Windows можливе використання як PowerShell, так і окремих SSH-клієнтів.

Після першого підключення виконуються базові дії з оновлення системи та встановлення необхідних компонентів. Для дистрибутивів Ubuntu/Debian застосовується, наприклад, така послідовність команд:

*sudo apt update && sudo apt upgrade -y*
*sudo apt install -y openjdk-17-jre*

Після встановлення середовища виконання Java встановлюється Jenkins. На практиці це здійснюється через додавання офіційного репозиторію Jenkins та встановлення пакета за допомогою пакетного менеджера. Спрощений приклад команд може мати вигляд:

*sudo apt install -y jenkins*
*sudo systemctl status jenkins*

Після успішного встановлення Jenkins запускається як системна служба, що за замовчуванням прослуховує порт 8080. Подальша робота відбувається через вебінтерфейс. Відкривши у браузері адресу http://xxx.xxx.xxx.xxx:8080, де xxx.xxx.xxx.xxx – IP-адреса сервера, користувач потрапляє до початкового вікна налаштування Jenkins (рис. 3.6).

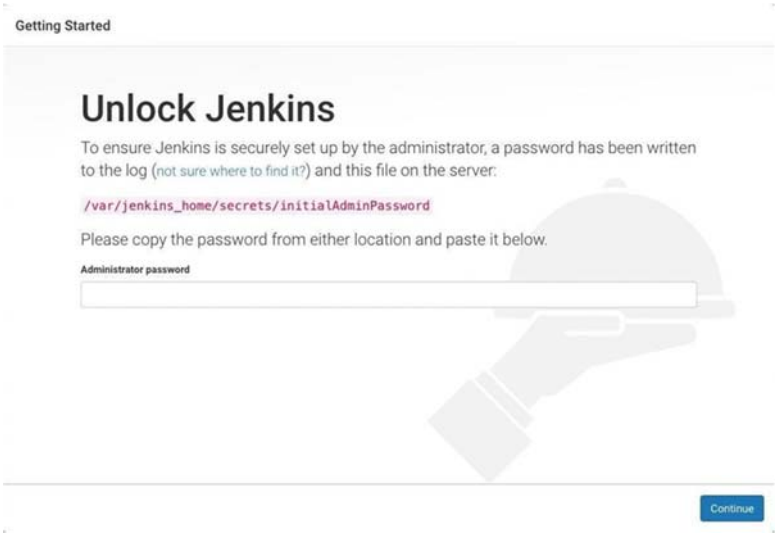


Рисунок 3.5 – Початкове вікно Jenkins з вимогою введення початкового адміністративного пароля.

Для продовження конфігурації необхідно ввести одноразовий пароль розблокування, який створюється під час першого запуску Jenkins. Цей пароль зберігається у файлі initialAdminPassword у службовому каталозі Jenkins або відображається в журналі служби при старті (рис. 3.6).

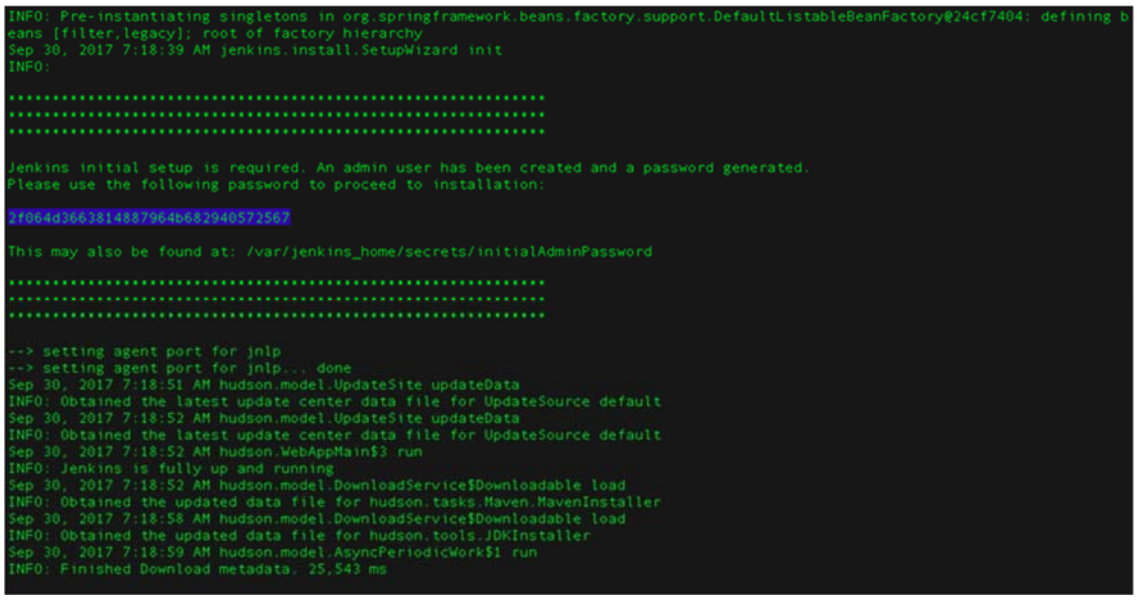


Рисунок 3.6 – Пошук початкового адміністративного пароля Jenkins у системному журналі

Після введення пароля майстер початкового налаштування пропонує створити адміністративний обліковий запис, обрати базовий набір плагінів та завершити первинну конфігурацію сервера. На цьому етапі формується мінімально необхідне середовище для подальшого створення конвеєрів CI/CD.

Завершивши базове налаштування, переходять до проєктування та реалізації власне CI/CD-процесу для умовного вебзастосунку.

### 3.3 Реалізація CI/CD-конвеєра для вебзастосунку на основі Jenkins.

Побудова CI/CD-конвеєра розпочинається з визначення цільової архітектури процесу. У межах даної роботи прийнято, що вихідний код вебзастосунку зберігається в репозиторії Git (наприклад, GitHub), Jenkins виконує роль сервера автоматизації, окремі агенти відповідають за збірку та

тестування, а результатом є готові артефакти, що розгортаються спочатку у тестовому, а потім у продуктивному середовищі. Узагальнена структурна схема реалізації безперервної інтеграції подана на рисунку 3.7.

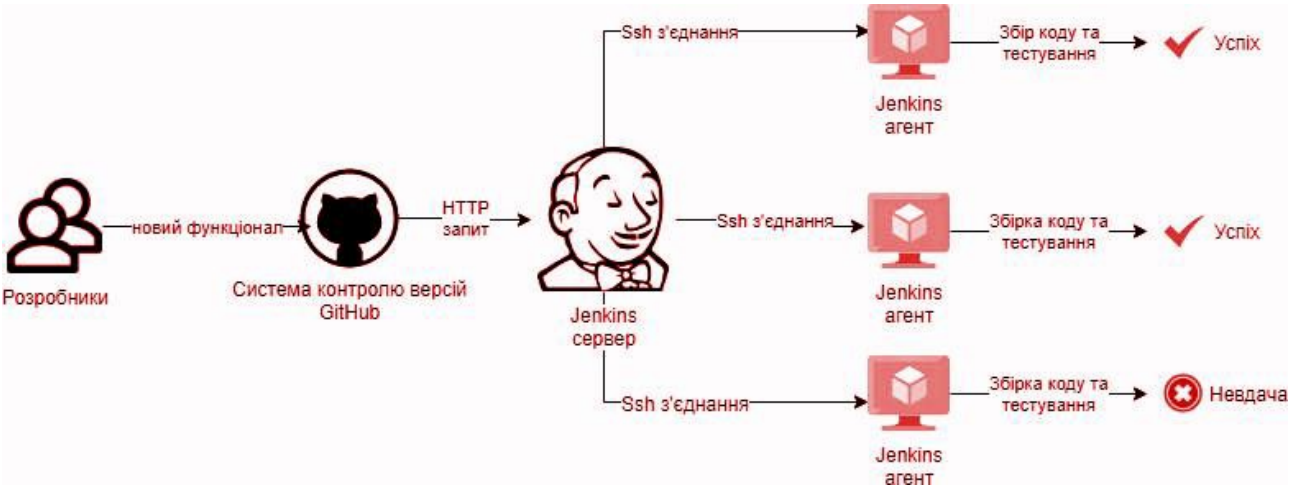


Рисунок 3.7 – Структурна схема процесу безперервної інтеграції з використанням Jenkins, агентів та репозиторію коду GitHub

У процесі розробки приймається загальноприйнята стратегія роботи з гілками в Git. Для реалізації нових функціональних можливостей створюються окремі гілки, які після завершення розробки та первинного тестування в межах робочого середовища розробника відправляються на злиття до основної гілки (наприклад, master або main). Подання запиту на злиття слугує тригером для Jenkins: за допомогою вебхука сервер отримує повідомлення від Git-платформи та запускає конвеєр збірки та тестування.

**Налаштування плагінів та джерела вихідного коду**

Для інтеграції Jenkins з GitHub використовується плагін GitHub Plugin. Його встановлення виконується через розділ керування плагінами. У вебінтерфейсі Jenkins обирається пункт «Manage Jenkins», далі – «Manage Plugins», після чого відкривається вкладка з управлінням розширеннями (рис.

3.8, 3.9). У вкладці доступних плагінів виконується пошук за назвою GitHub Plugin та здійснюється інсталяція.


Рисунок 3.8 – Перехід до розділу «Manage Jenkins» у вебінтерфейсі Jenkins

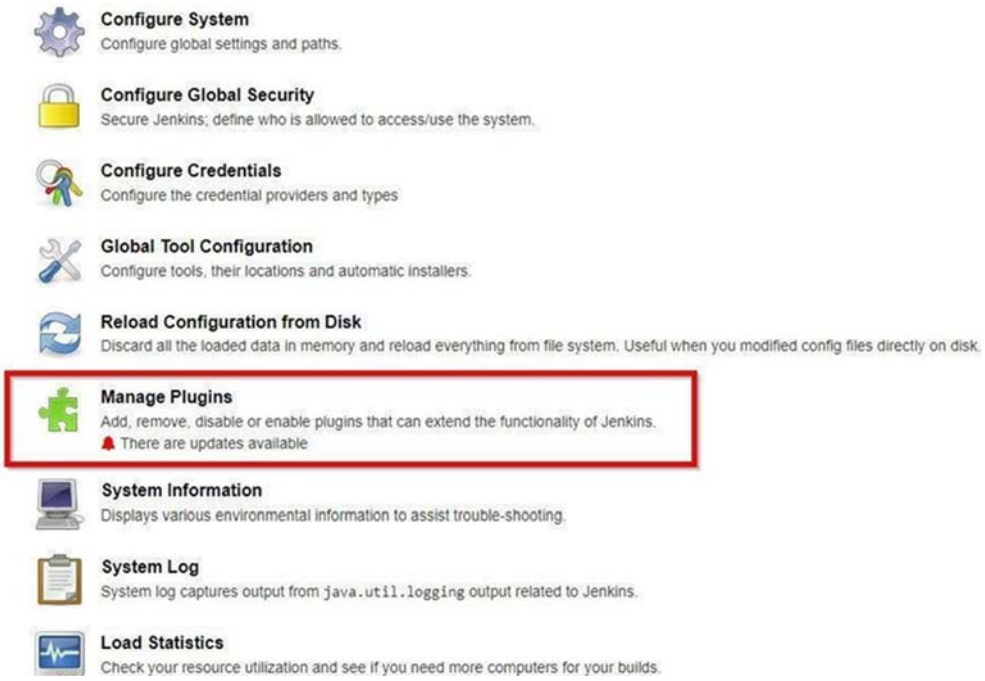


Рисунок 3.9 – Вікно керування плагінами Jenkins та пошук плагіна GitHub

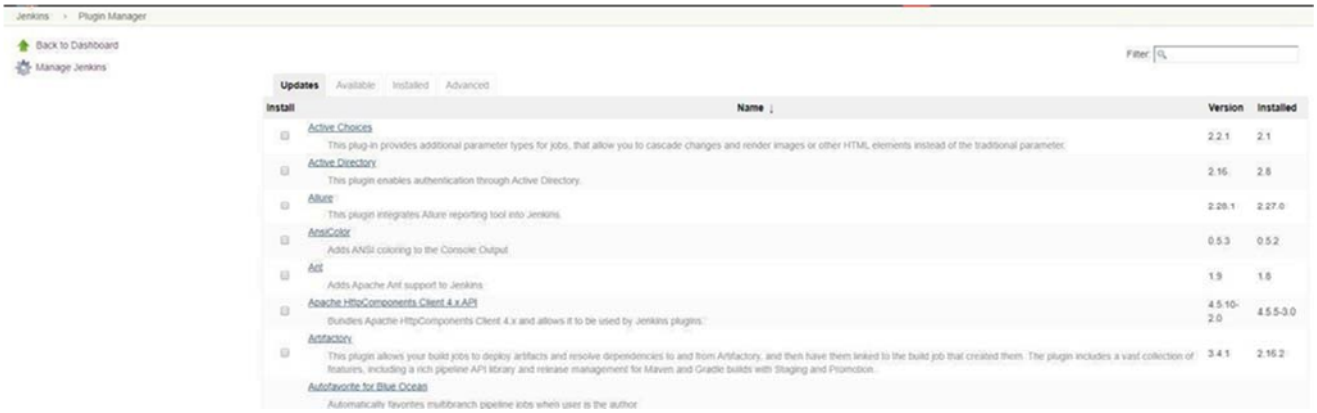


Рисунок 3.10 – Встановлення плагіна GitHub Plugin без перезавантаження сервера

Після інсталяції плагіна конфігурується перше завдання Jenkins. Створюється новий Job типу «Freestyle project» (рис. 3.11), якому надається логічна назва, наприклад Build, що відображає його роль у конвеєрі.


Рисунок 3.11 – Створення нового завдання типу Freestyle project у Jenkins

У налаштуваннях завдання у вкладці «Source Code Management» активується тип «Git» та задається URL репозиторію, де розташований вихідний код вебзастосунку. За потреби додаються облікові дані для доступу до приватного репозиторію. У полі «Branch Specifier» зазначається гілка, зміни в якій мають оброблятися конвеєром, наприклад */master (рис. 3.12).

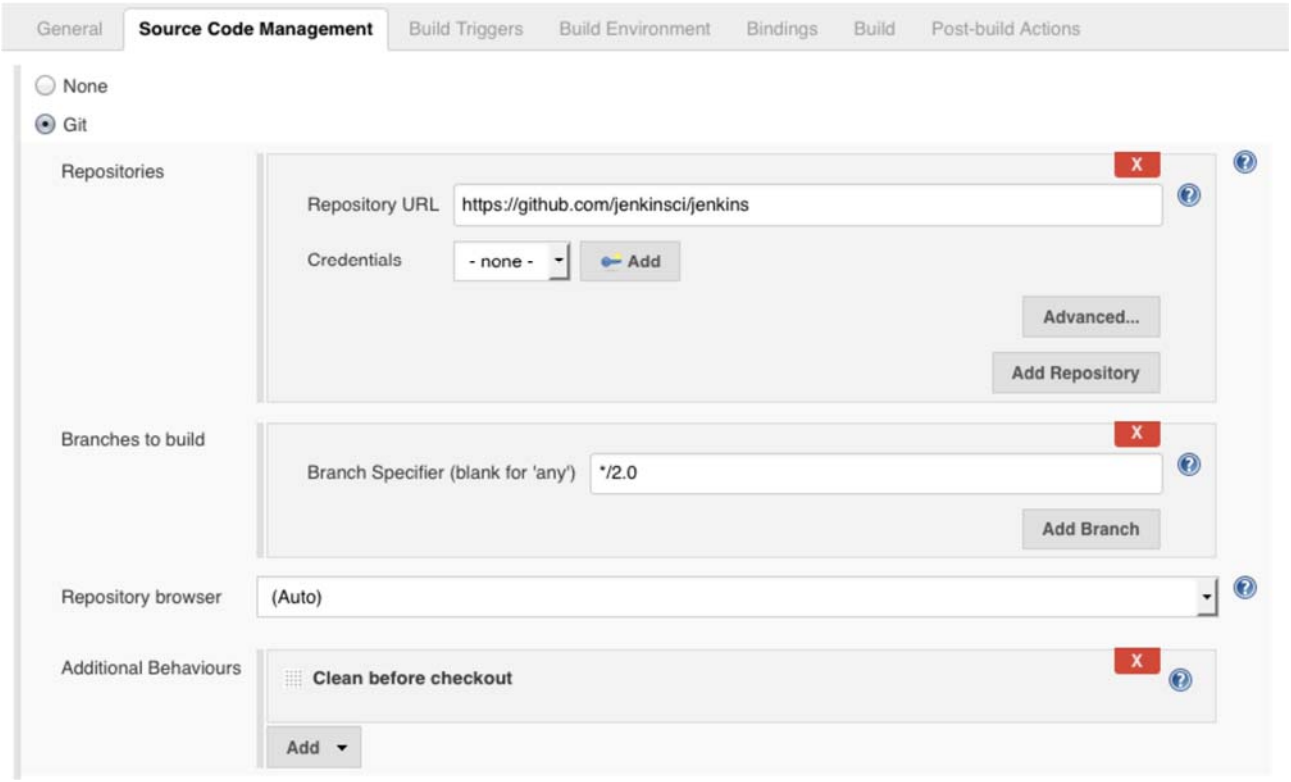


Рисунок 3.12 – Налаштування джерела вихідного коду та гілки Git у конфігурації Jenkins Job

Таким чином Jenkins отримує інформацію про місцезнаходження вихідного коду, який буде використовуватися під час автоматизованої збірки та тестування.

**Підключення агентів та розподіл навантаження**

Для забезпечення масштабованості та ізоляції процесів збірки й тестування використовуються агенти Jenkins. У якості агентів застосовуються окремі віртуальні машини на базі Linux, до яких Jenkins підключається по SSH. Перед реєстрацією агента необхідно переконатися, що на цільовій машині запущено SSH-службу, створено користувача з правами доступу та налаштовано пару SSH-ключів.

Реєстрація агента здійснюється через розділ «Manage Jenkins» – «Manage Nodes». На цій сторінці створюється новий вузол (New Node), якому задається

ім’я та тип (Permanent agent) (рис. 3.13). Після підтвердження відкривається форма детальної конфігурації агента (рис. 3.14).

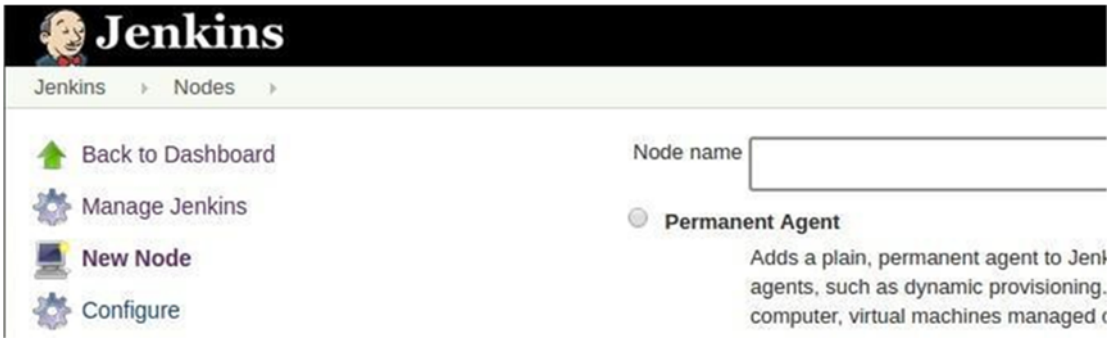


Рисунок 3.13 – Створення нового агента Jenkins у розділі Manage Nodes

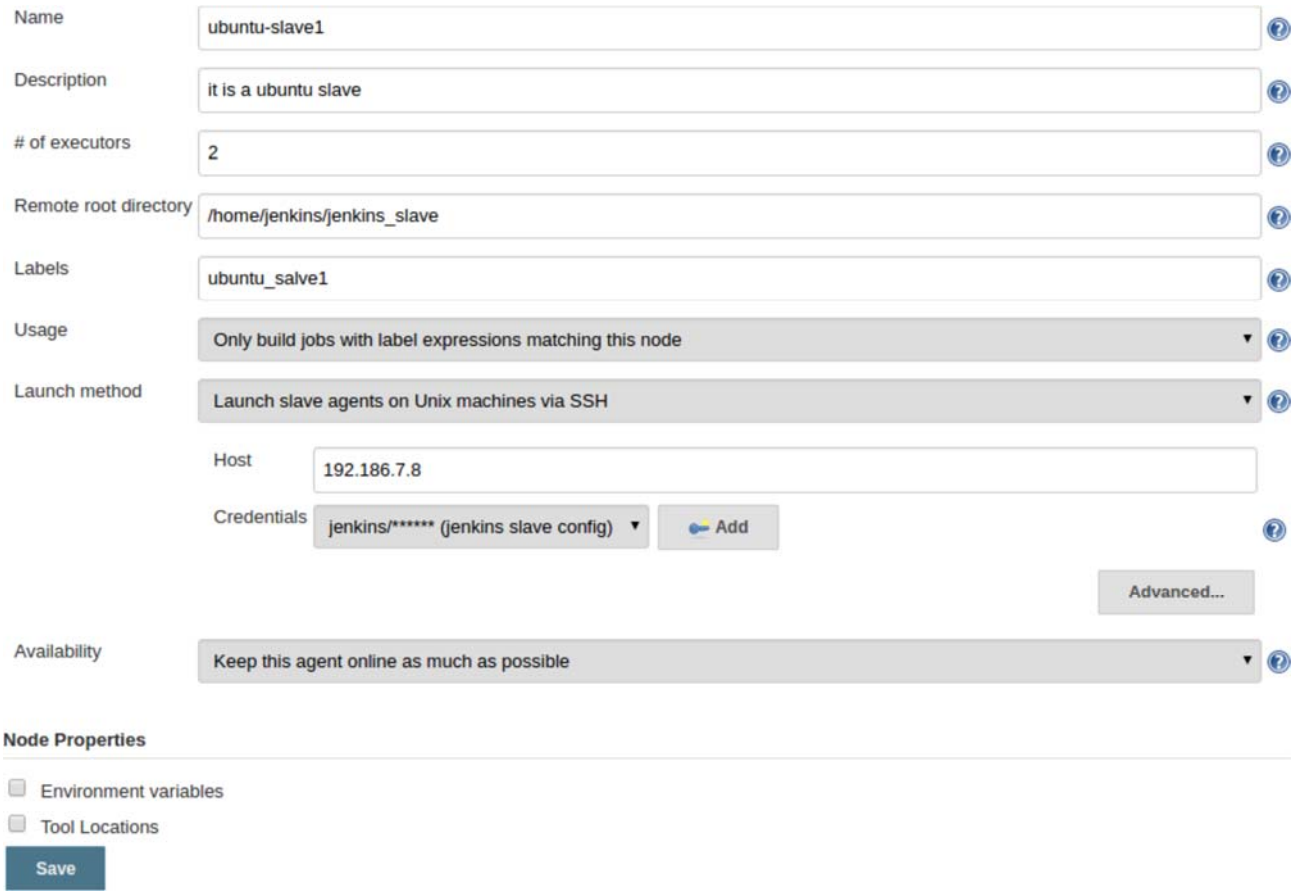


Рисунок 3.14 – Конфігурація параметрів агента Jenkins (опис, директорія, адреса, облікові дані)

У параметрах агента визначається максимальна кількість паралельних завдань, робочий каталог, адреса хоста, а також вибираються облікові дані з раніше налаштованим SSH-ключем. Після збереження конфігурації Jenkins

перевіряє можливість підключення до агента. У разі успіху агент готовий до виконання збірок.

Щоб конкретне завдання виконувалося саме на цьому агенті, у конфігурації Job Build активується опція «Restrict where this project can be run» та вводиться мітка, задана для агента (рис. 3.15).

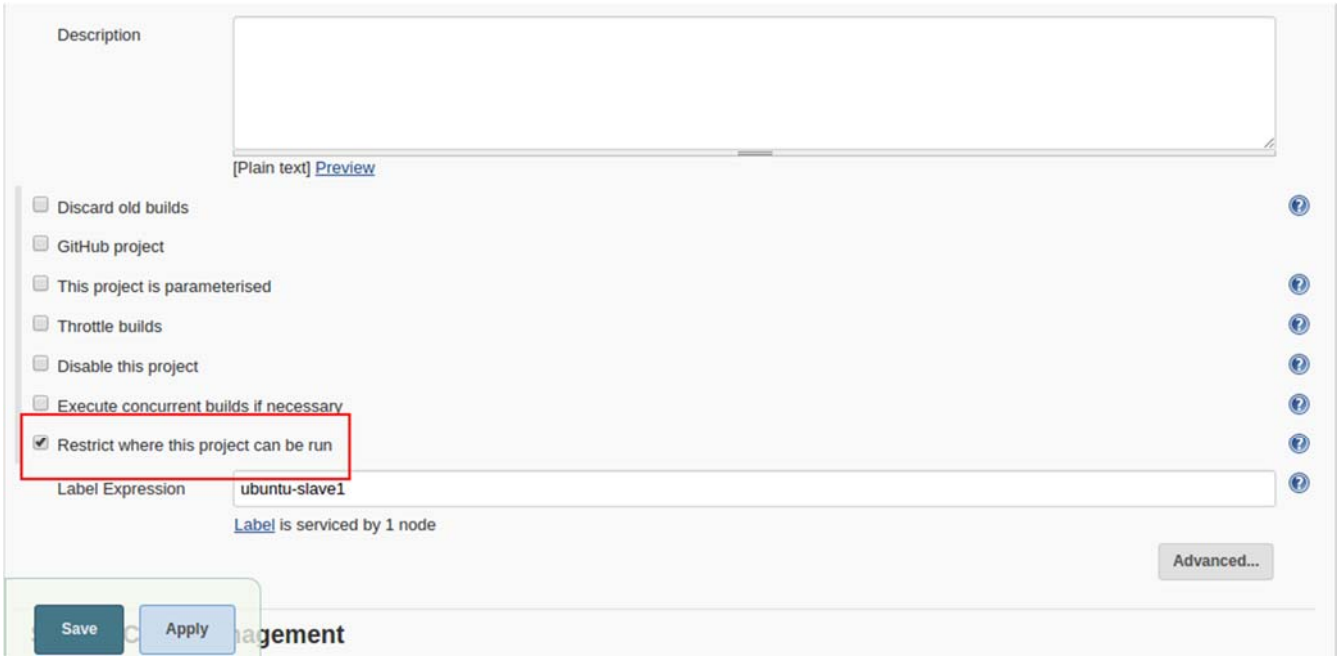


Рисунок 3.15 – Прив’язка завдання Jenkins до конкретного агента через Label Expression

**Налаштування збірки, тестування та тригерів**

Основою безперервної інтеграції є автоматизована збірка та виконання тестів. У секції «Build» Job Build додається крок типу «Invoke top-level Maven targets» (рис. 3.16). У полі «Goals» задаються цілі Maven, що визначають, які дії буде виконано.

У наведеному прикладі спочатку очищуються проміжні артефакти, а потім запускаються модульні тести. Для побудови кінцевого артефакту (наприклад, WAR-файлу) можна використовувати ціль package (команда clean package).

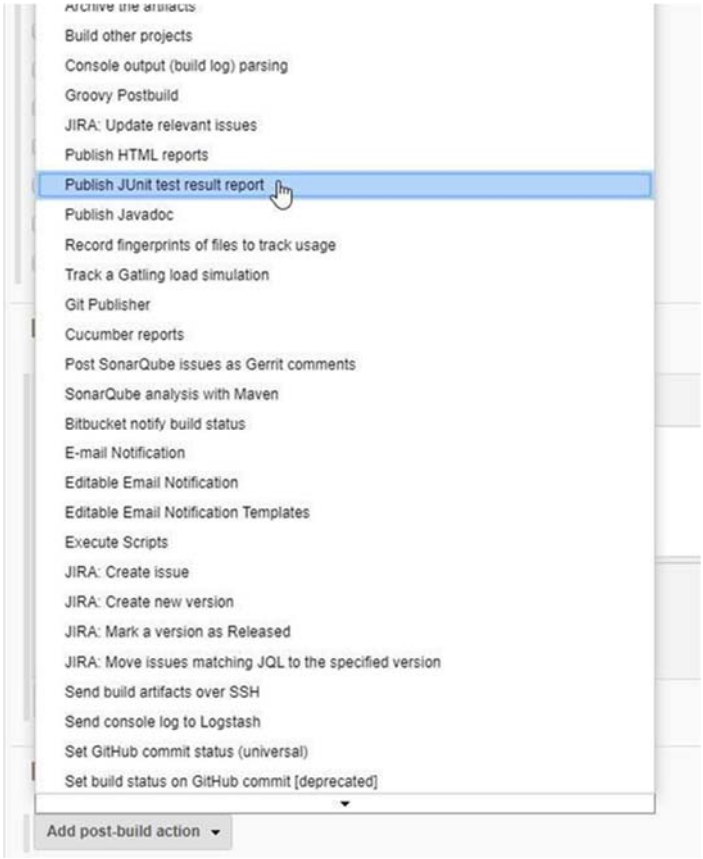


Рисунок 3.16 – Налаштування кроку збірки Java-проєкту за допомогою Maven у Jenkins

Для аналізу результатів модульного тестування використовується плагін JUnit. У розділі «Post-build Actions» додається дія «Publish JUnit test result report», де вказується шлях до звітів Maven, наприклад **/target/surefire-reports/*.xml (рис. 3.17).

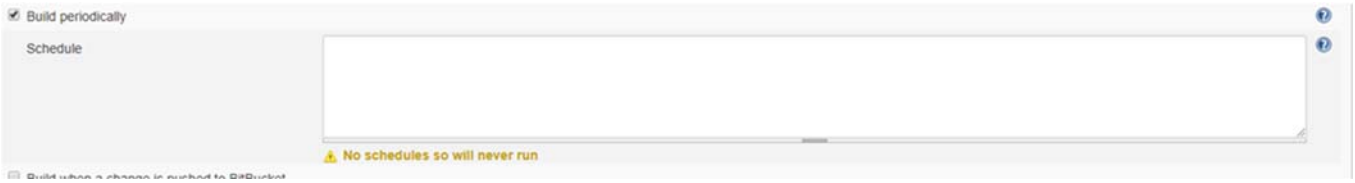


Рисунок 3.17 – Налаштування публікації результатів модульного тестування JUnit у Jenkins

Наступним кроком є визначення тригерів запуску Job. У демонстраційному сценарії використовується як періодичний запуск, так і запуск

за подією в Git. Для періодичного виконання активується опція «Build periodically» (рис. 3.19), після чого в полі «Schedule» задається вираз у форматі cron «H * * * *», що означає запуск приблизно раз на годину.

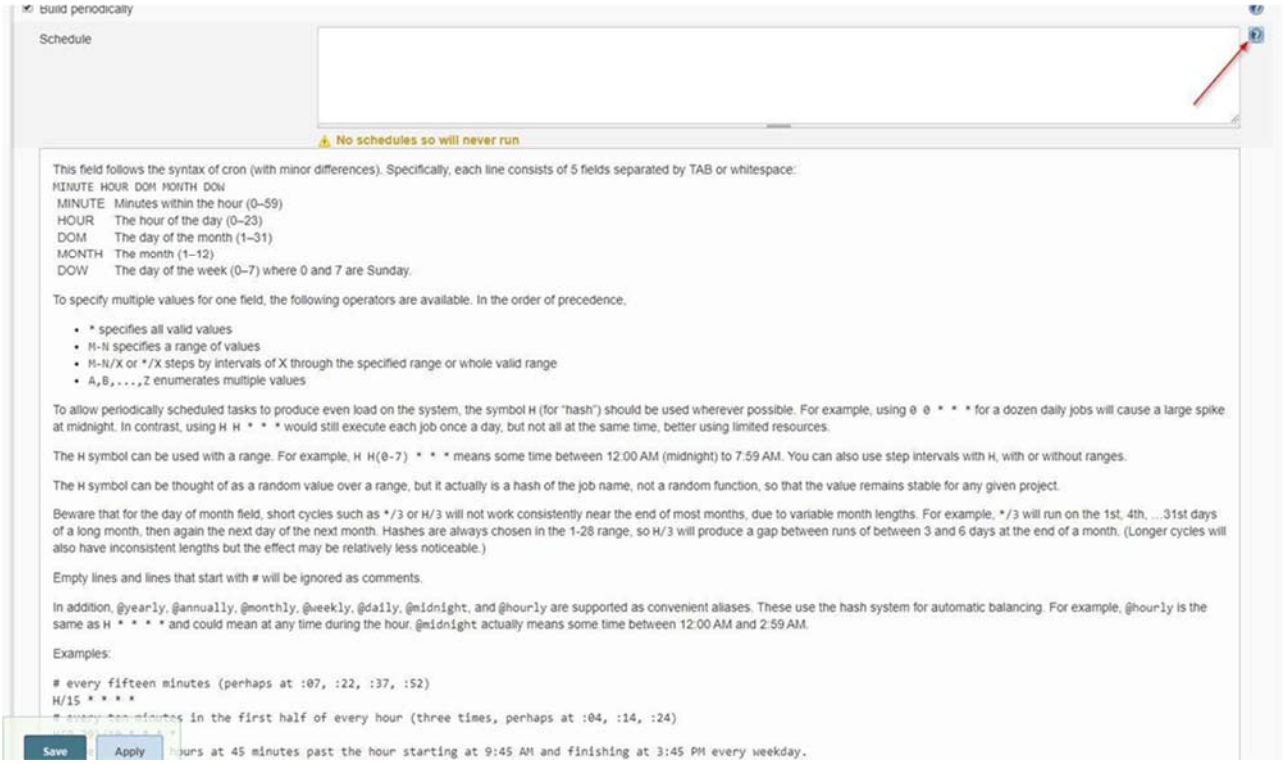


Рисунок 3.18 – Приклад налаштування періодичного запуску збірки в Jenkins

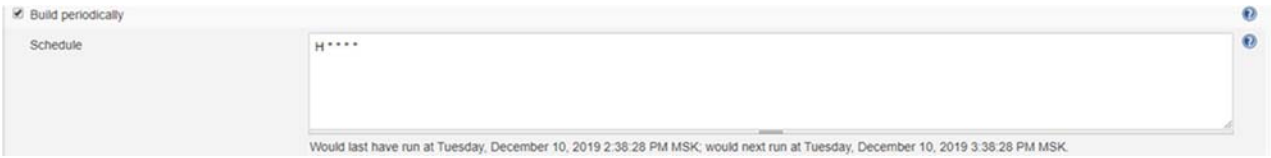


Рисунок 3.19 – Вікно редагування cron-виразу для регулярного виконання завдання

Незважаючи на наявність автоматичних тригерів, завдання завжди може бути запущене вручну кнопкою «Build Now» у вебінтерфейсі (рис. 3.20). Це зручно для перевірки коректності конфігурації або оперативного запуску позапланової збірки.

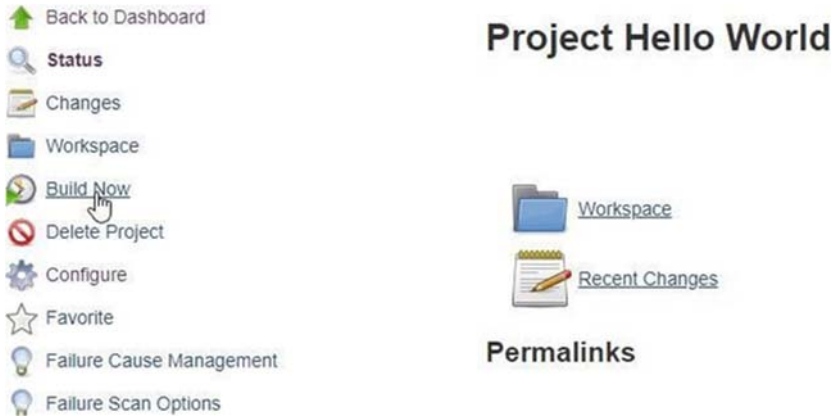


Рисунок 3.20 – Історія запусків Jenkins Job та індикатори поточного стану виконання

У разі успішного виконання збірки та проходження модульних тестів Job позначається зеленим (або синім) маркером, а в розділі з результатами тестування відображається тренд проходження тестів (рис. 3.21).

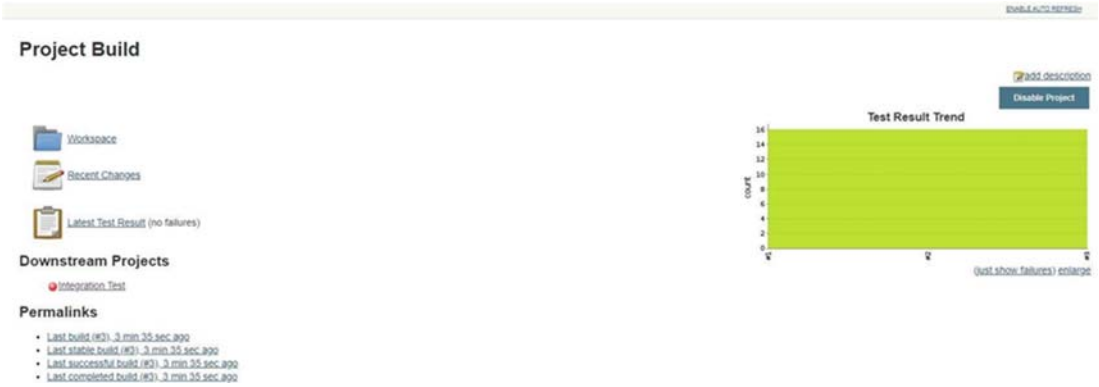


Рисунок 3.21 – Графік результатів модульних тестів Jenkins на основі JUnit-звітів

**Інтеграційне тестування та оповіщення розробників**

Для розширення покриття тестами доцільно додати інтеграційне тестування. Для цього створюється окремий Job, наприклад Integration Test (рис. 3.22), який виконує тестові сценарії поверх уже зібраного застосунку. У конфігурації цього завдання як тригер запуску задається опція «Build after other

projects are built», де зазначається Job Build та умова запуску лише у разі успішного завершення попередньої збірки (рис. 3.24). Таким чином формується послідовний ланцюжок: спочатку збірка та модульні тести, потім інтеграційні тести.


Рисунок 3.22 – Створення Jenkins Job для інтеграційного тестування вебзастосунку

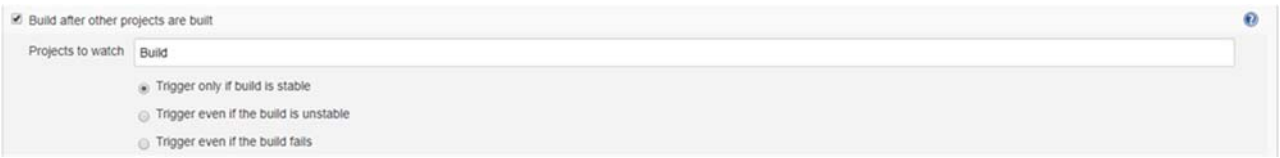


Рисунок 3.23 – Налаштування запуску інтеграційних тестів після успішної збірки проєкту

Для забезпечення зворотного зв’язку з розробниками налаштовується система оповіщень. У розділі «Post-build Actions» для відповідних Job додається дія «E-mail Notification», де вказуються адреси одержувачів та умови відправлення листів (наприклад, лише у випадку невдачі або при будь-якому завершенні) (рис. 3.24). У результаті кожен розробник оперативно отримує інформацію про статус збірки та тестів свого коду.

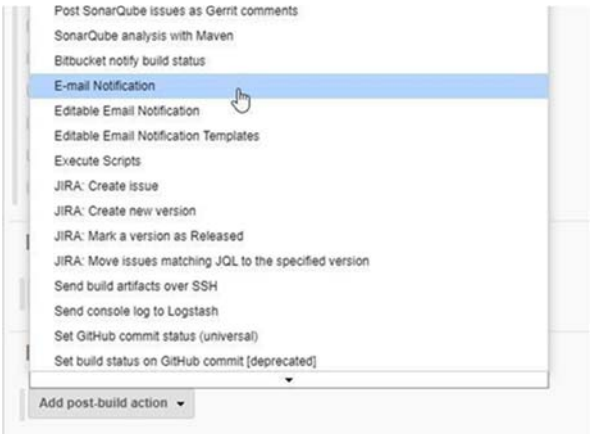


Рисунок 3.24 – Налаштування надсилання електронних сповіщень про результати виконання Jenkins Job

Щоб доповнити процес безперервної інтеграції компонентами безперервної доставки та розгортання, створюються додаткові Job, відповідальні за автоматичне розгортання вебзастосунку у тестовому (staging) та продуктивному (production) середовищах. Також додається завдання регресійного тестування, яке запускається після оновлення тестового середовища. Сукупність створених Job наведено на рисунку 3.25.


Рисунок 3.25 – Перелік Jenkins Job, що реалізують етапи CI, тестування, staging- та production-розгортання

Середовище staging інтегрується з Jenkins як додатковий агент, подібно до того, як це було зроблено для агента збірки. У конфігурації Job, відповідального за розгортання у staging, задається використання відповідного агента та тригер запуску після успішного завершення інтеграційного тестування. У секції «Build» цього Job можуть використовуватися специфічні цілі Maven або окремі скрипти розгортання, які виконують збирання, копіювання артефактів, оновлення конфігурацій і запуск вебсервера.

Аналогічним чином конфігурується Job для розгортання у продуктивному середовищі, який запускається після успішного проходження регресійних тестів у staging. У результаті формується повний ланцюжок від коміту в репозиторії до появи нової версії вебзастосунку в продуктивному середовищі.

Для підсумкового унаочнення побудованого рішення формується загальна діаграма конвеєра CI/CD, яка відображає всі етапи: отримання вихідного коду, збірку, модульне та інтеграційне тестування, регресійні перевірки, розгортання у staging та production, а також оповіщення учасників команди. Така схема наведена на рисунку 3.26.

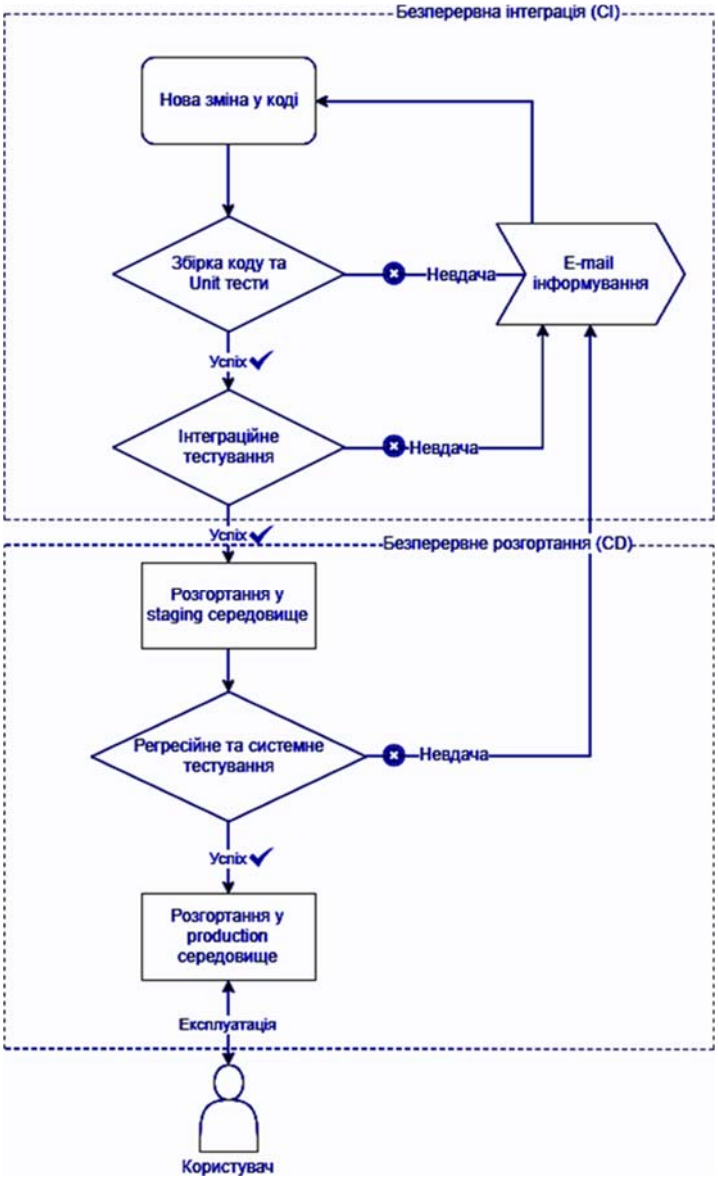


Рисунок 3.26 – Загальна схема створеного CI/CD-конвеєра для вебзастосунку на основі Jenkins

Реалізований у рамках роботи CI/CD-процес демонструє, що використання Jenkins дозволяє повністю автоматизувати основні етапи життєвого циклу вебзастосунку, скоротити час між внесенням змін у код та отриманням нової версії у продуктивному середовищі, а також зменшити ризики людських помилок завдяки систематичному застосуванню автоматизованих тестів та формалізованих процедур розгортання. Це відповідає вимогам, сформульованим у постановці задачі, та підтверджує доцільність обраного інструментарію.

## 3.4. Оцінювання результатів побудови CI/CD-конвеєра та їх візуалізація

Після розгортання Jenkins, підключення агентів, налаштування інтеграції з репозиторієм та реалізації послідовності завдань збірки, тестування і розгортання було виконано експериментальну перевірку працездатності розробленого CI/CD-конвеєра. Метою цього етапу є отримання кількісних характеристик, які відображають швидкодію процесу, його стабільність та якість контролю змін. Для цього було здійснено серію послідовних запусків конвеєра на одному й тому самому прикладному проєкті з фіксацією тривалості ключових етапів та статусу виконання. Як базові метрики обрано тривалість проходження етапів збірки, модульного тестування, інтеграційного тестування, розгортання у staging-середовище, регресійного тестування та розгортання у production-середовище. Додатково аналізувалася інтегральна тривалість проходження конвеєра та частка успішних запусків.

Вимірювання виконувалися в умовах стабільної конфігурації середовища, що дозволяє вважати отримані дані порівнюваними між собою. На практиці такі показники зручно отримувати безпосередньо з журналів збірок Jenkins та таймінгів етапів у Pipeline. Для уніфікації збору метрик доцільно фіксувати час початку й завершення кожного етапу, а результати зберігати як артефакти збірки у вигляді JSON або CSV. Нижче наведено приклад фрагмента Jenkinsfile, який може застосовуватися для фіксації тривалості етапів і формування артефакту з метриками:

```
def metrics = [:]

def timedStage(String name, Closure body) {
  long t0 = System.currentTimeMillis()
  try {
    stage(name) { body() }
```

```
        metrics[name] = (System.currentTimeMillis() - t0) / 1000.0
    } catch (e) {
        metrics[name] = (System.currentTimeMillis() - t0) / 1000.0
        throw e
    }
}

pipeline {
    agent any
    stages {
        timedStage('Build') {
            sh 'mvn -B clean package'
        }
        timedStage('Unit tests') {
            sh 'mvn -B test'
        }
        timedStage('Integration tests') {
            sh 'mvn -B verify -P integration'
        }
        timedStage('Deploy staging') {
            sh './deploy.sh staging'
        }
        timedStage('Regression tests') {
            sh './regression_tests.sh'
        }
        timedStage('Deploy production') {
            sh './deploy.sh production'
        }
    }
    post {
```

```
        always {
            writeJSON file: 'metrics.json', json: metrics
            archiveArtifacts artifacts: 'metrics.json', fingerprint: true
        }
    }
}
```

За результатами серії запусків було отримано масив часових значень, на основі якого побудовано узагальнювальні графіки. Загальна тривалість проходження конвеєра за послідовними запусками подана на рисунку 3.27. Графік демонструє, що більшість запусків лежить у вузькому діапазоні значень, що свідчить про відтворюваність процесу та передбачуваність часу виконання. Окремі відхилення зазвичай пояснюються нестабільністю зовнішніх залежностей, варіативністю часу завантаження залежностей Maven або відмінностями в обсязі змін. Важливо, що при збої на етапі тестування конвеєр коректно завершується з помилкою, не переходячи до етапів розгортання, що знижує ризики потрапляння нефункціональної версії до наступних середовищ.

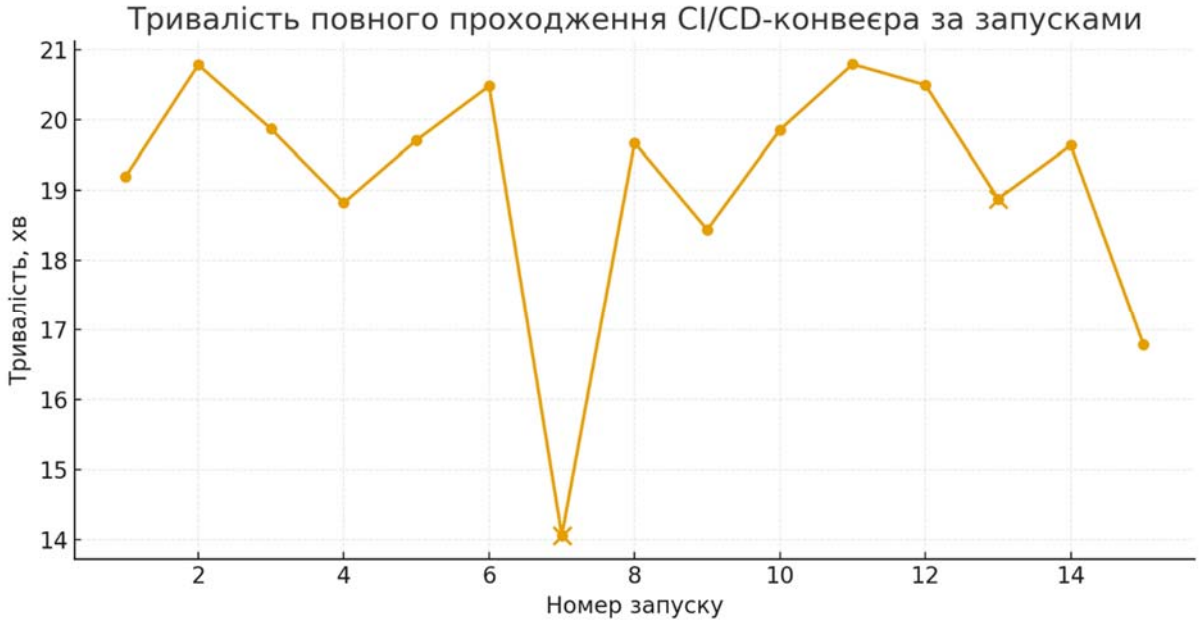


Рисунок 3.27 – Тривалість повного проходження CI/CD-конвеєра за послідовними запусками

Для поглибленого аналізу продуктивності процесу було обчислено середню тривалість виконання кожного етапу на множині успішних запусків. Візуалізація у вигляді середніх значень подана на рисунку 3.28. Це дозволяє виділити «критичні» ділянки конвеєра, які мають найбільший внесок у загальний час. Як правило, для Java-вебзастосунків найбільш ресурсоємними виявляються компіляція з підготовкою артефактів, інтеграційні перевірки та регресійні сценарії, тоді як операції розгортання можуть бути коротшими за умови автоматизованих скриптів і належно налаштованого середовища.

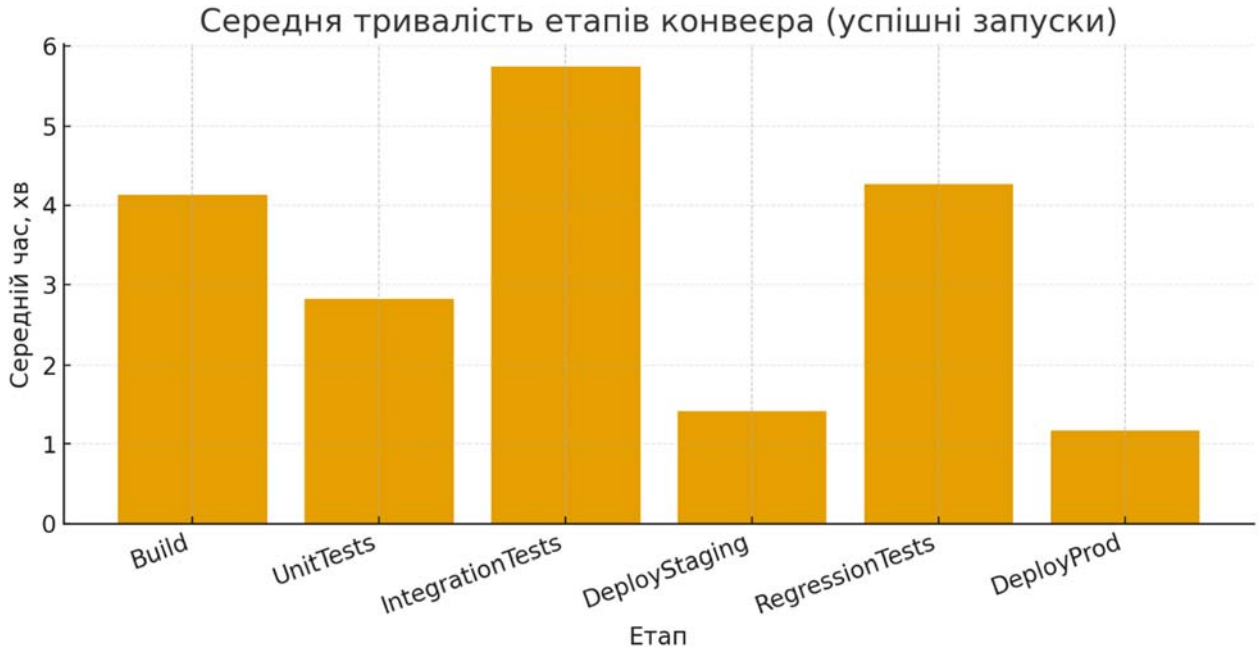


Рисунок 3.28 – Середня тривалість етапів конвеєра (успішні запуски)

Окремий аспект результативності CI/CD-процесу пов’язаний зі стабільністю виконання, тобто часткою запусків, що завершуються успішно. Для відображення тенденції стабільності використано накопичений показник частки успішних запусків, який демонструє, як змінюється надійність конвеєра у міру виконання серії прогонів. Відповідний графік подано на рисунку 3.29. Такий підхід є корисним у практиці впровадження CI/CD, оскільки дозволяє оперативно оцінити, чи проходить процес «стабілізацію» після початкового

налаштування, а також виявляти вплив змін у конфігурації конвеєра або тестового набору.

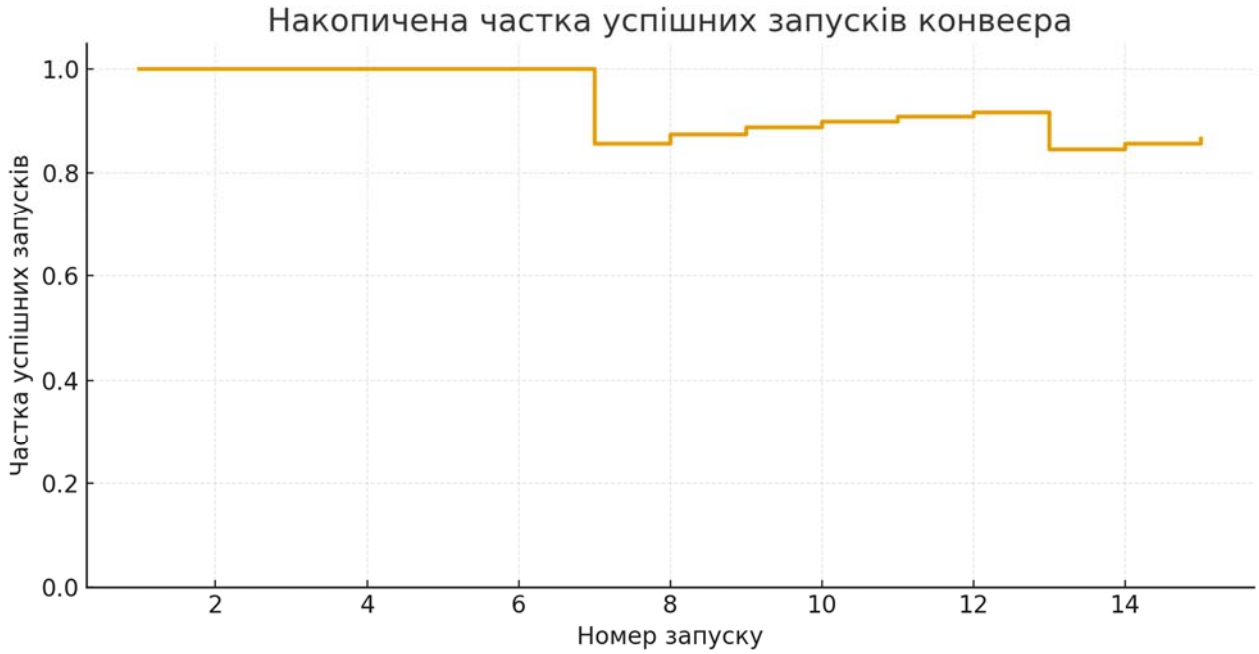


Рисунок 3.29 – Накопичена частка успішних запусків CI/CD-конвеєра

Для коректної інтерпретації результатів тривалість проходження конвеєра доцільно аналізувати окремо для успішних і неуспішних запусків, оскільки при збої виконання зупиняється на етапі виявлення помилки та не переходить до наступних стадій. На рисунку 3.30 наведено діаграми розмаху інтегральної тривалості запусків, розділені за двома групами: успішні та неуспішні. Візуалізація показує, що для успішних запусків тривалість є більш стабільною та концентрується в межах відносно вузького інтервалу, що свідчить про відтворюваність працездатного сценарію конвеєра. Для неуспішних запусків характерною є менша тривалість через дострокове завершення на етапі тестування, що, з одного боку, підтверджує правильність логіки “fail-fast”, а з іншого вона демонструє, що система автоматизації не допускає переходу до розгортання у випадку невиконання контрольних перевірок. Такий підхід підвищує надійність релізного процесу та зменшує ризик потрапляння дефектної версії вебзастосунку у наступні середовища.

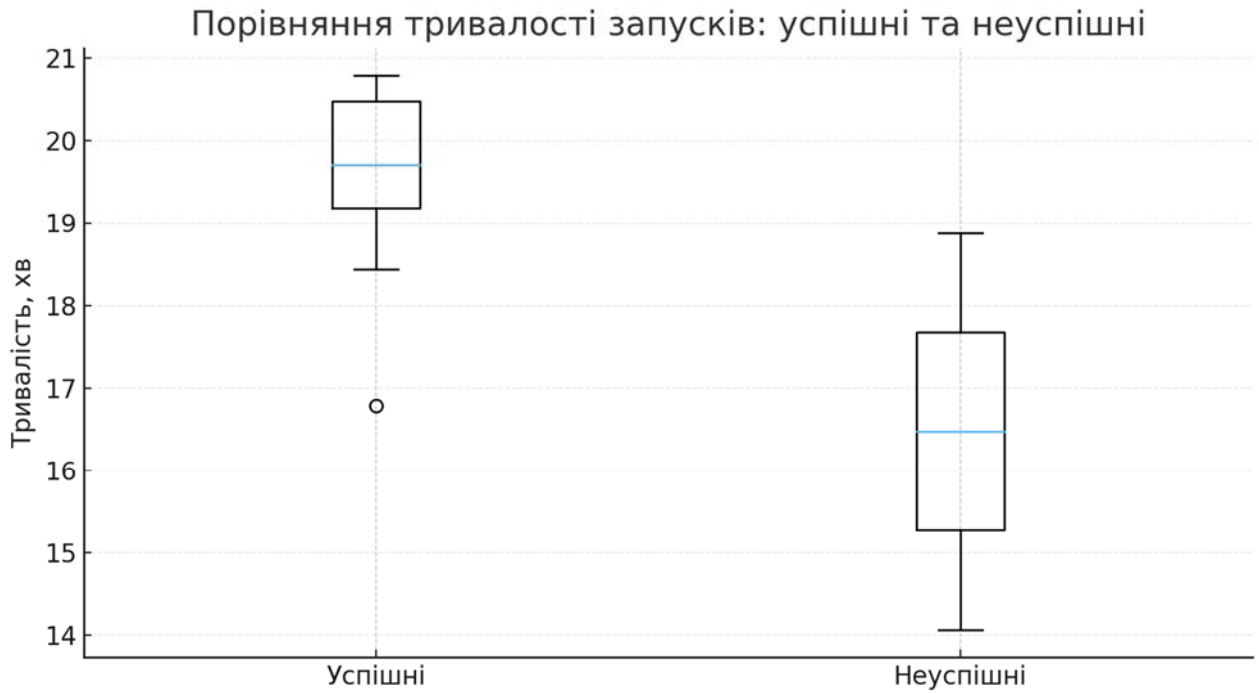


Рисунок 3.30 – Порівняння тривалості запусків CI/CD-конвеєра: успішні та неуспішні (діаграма розмаху)

Узагальнюючи отримані результати, можна зазначити, що реалізований CI/CD-конвеєр відповідає поставленій задачі автоматизації розгортання вебзастосунку. Він забезпечує повторювану процедуру збірки й тестування, блокує розгортання у випадку помилок на контрольних етапах, надає прозорі метрики щодо тривалості робіт та дозволяє накопичувати статистику для подальшої оптимізації. Практична цінність отриманих результатів полягає в тому, що за рахунок формалізації процедур і зменшення ручних операцій підвищується керованість релізного циклу, а також створюються умови для регулярного випуску інкрементів із прогнозованими витратами часу.

# РОЗДІЛ 4.
# ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

## 4.1. Розробка логіко-імітаційної моделі виникнення травм і аварій

Методикою оцінки рівня небезпеки робочих місць, машин, виробничих процесів та окремих виробництв передбачено пошук об'єктивного критерію рівня небезпеки для конкретного об'єкта [38]. Таким показником вибрана ймовірність виникнення аварії, травми залежно від явища, що досліджується.

Для побудови логіко-імітаційної моделі процесу, формування і виникнення аварії та травми в процесі створення мікрокліматичних умов у приміщенні оцінюють відповідні небезпечні події. Кожній їх них присвоїмо ймовірність виникнення:

| Шифр | Назва події | Ймовірність |
|---|---|---|
| $P_1$ | Відсутність захисного заземлення | 0,02 |
| $P_2$ | Пошкодження захисного заземлення | 0,04 |
| $P_3$ | Спрацювання складових захисту | 0,1 |
| $P_4$ | Неправильна експлуатація захисту | 0,02 |
| $P_5$ | Відсутність профілактичних заходів | 0,2 |
| $P_6$ | Відсутність захисного щита | 0,12 |
| $P_7$ | Недотримання правил вибору взуття | 0,15 |
| $P_8$ | Незнання правил техніки безпеки | 0,1 |
| $P_9$ | Відсутність засобів індивідуального захисту | 0,2 |
| $P_{10}$ | Легковажність | 0,08 |

На основі наведених подій будуємо матрицю логічних взаємозв'язків між окремими пунктами, графічна інтерпретація якої зображено на рис. 4.1.

Розрахуємо ймовірності виникнення подій, що формують логіко-імітаційну модель процесів створення мікрокліматичних умов. Розглянемо травмонебезпечну ситуацію, що виникає за умови роботи працівників із електронебезпекою.

Підставивши дані ймовірностей базових подій у формулу, отримаємо ймовірність події 13: $P_{13} = 0,2+ 0,4 – 0,2 \cdot 0,4 = 0,0592$.

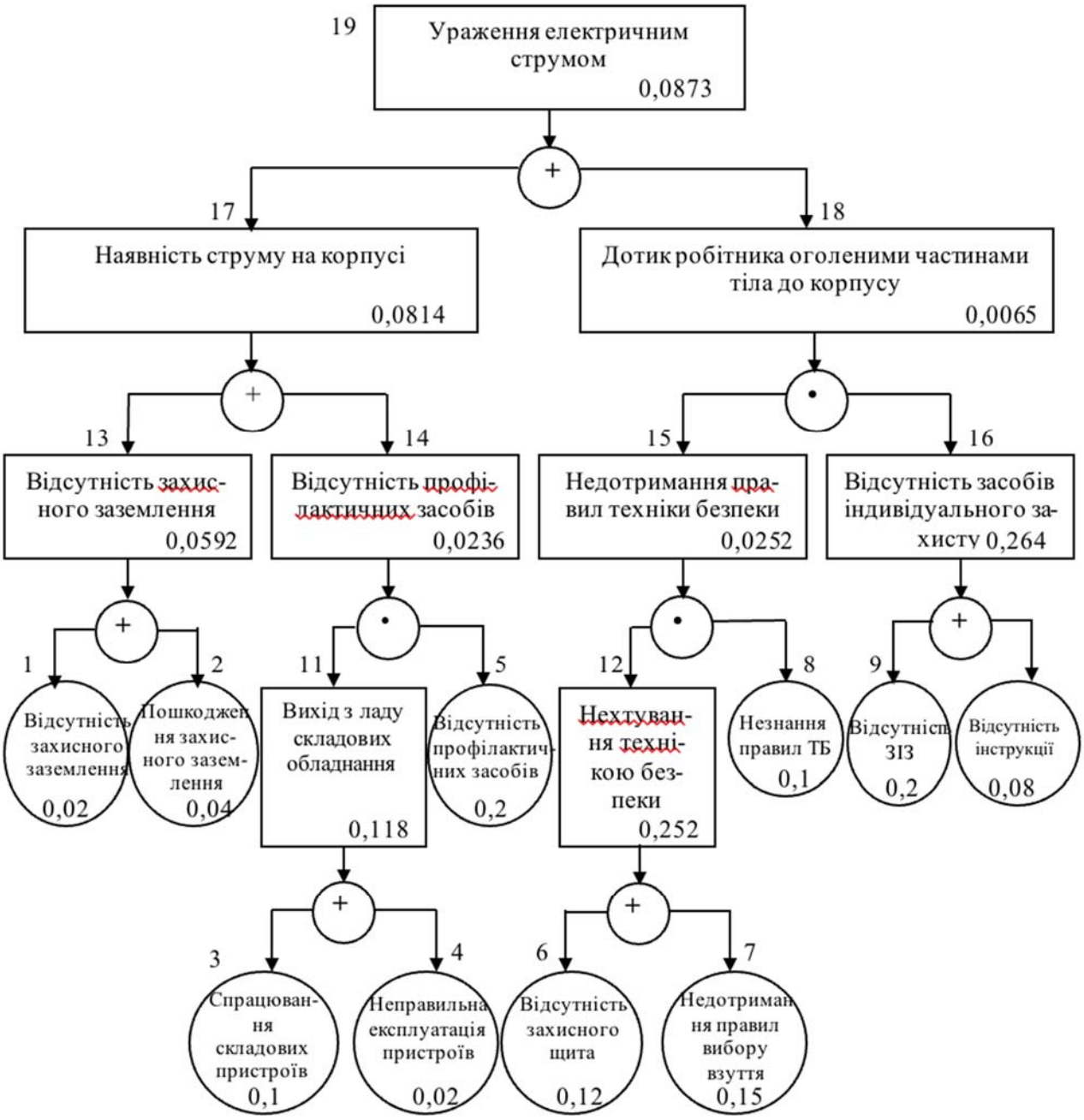


Рис. 4.1. Матриця логічних взаємозв’язків між окремими подіями травмонебезпечної ситуації [38]

Аналогічно визначаємо ймовірність інших подій:

$$P_{11} = P_4 + P_5 - P_4P_5 = 0,3 + 0,4 - 0,3 \cdot 0,4 = 0,118.$$

$$P_{12} = P_6 + P_7 - P_6P_7 = 0,3 + 0,5 - 0,3 \cdot 0,5 = 0,252.$$

$$P_{16} = P_9 + P_{10} - P_9P_{10} = 0,2 + 0,15 - 0,2 \cdot 0,15 = 0,264.$$

$$P_{14} = P_{11} \cdot P_5 = 0,118 \cdot 0,2 = 0,0236.$$

$$P_{15} = P_{12} \cdot P_8 = 0,252 \cdot 0,1 = 0,0252.$$

$$P_{17} = P_{13}+P_{14} – P_{13}\cdot P_{14} = 0{,}592+0{,}0236-0{,}0592\cdot 0{,}0236 = 0{,}0814.$$

$$P_{18} = P_{15}\cdot P_{16} = 0{,}264\cdot 0{,}0252 = 0{,}0065.$$

$$P_{19} = P_{17}+P_{18} – P_{17}\cdot P_{18} = 0{,}0065+0{,}0814 – 0{,}0065\cdot 0{,}0814 = 0{,}0873.$$

Таким чином, ймовірність перекидання машини та наслідкового виникнення травми працівника є досить мала і становить – $P_{19} = 0,0873$.

## 4.2. Планування заходів із покращення умов праці

До заходів щодо покращення умов праці належать всі види діяльності, спрямовані на попередження, нейтралізацію або зменшення негативної дії шкідливих і небезпечних виробничих факторів на працівників.

Рівень умов праці оцінюють порівнянням за фактичними і нормативними значеннями узагальнених (групових) показників.

Заходи щодо поліпшення умов праці здійснюють з метою створення безпечних умов праці шляхом:

- доведення до нормативного рівня показників виробничого середовища за елементами умов праці;
- захисту працівників від дії небезпечних і шкідливих виробничих факторів.

До показників ефективності заходів щодо поліпшення умов праці належать:

а) зміни стану умов праці:

- зміна кількості засобів виробництва, приведених у відповідність до вимог стандартів безпеки праці;
- покращання санітарно-гігієнічних показників;
- покращання психофізичних показників, зменшення фізичних і нервово-психічних навантажень, в т.ч. монотонних умов праці;
- покращання естетичних показників, раціональне

компонування робочих місць і впорядкування робочих приміщень;

б) соціальні результати заходів:

- збільшення кількості робочих місць, що відповідають нормативним вимогам;
- зниження рівня виробничого травматизму;
- зменшення кількості випадків професійних захворювань;
- зменшення плинності кадрів через незадовільні умови праці;
- престиж та задоволення працею.

Отже, на покращення охорони праці потрібно виділити кошти на відновлення вентиляційних систем у ремонтних майстернях, естетично оформити приміщення офісу, відновити кабінет з охорони праці, поновити протипожежний інвентар.

### 4.3. Безпека в надзвичайних ситуаціях

Актуальність проблеми природно-техногенної безпеки для населення і території, зумовлена зростанням втрат людей, що спричиняється небезпечними природними явищами, промисловими аваріями та катастрофами. Ризик надзвичайних ситуацій природного та техногенного характеру невпинно зростає, тому питання захисту цивільного населення від надзвичайних ситуацій на сьогодні є дуже важливе [39].

У системі цивільної оборони окремого господарства необхідно забезпечити захист населення таким чином:

Укриття в захисних спорудах, якому підлягає усе населення відповідно до приналежності, досягається створенням фонду захисних споруд.

Евакуаційні заходи, які проводяться в містах та інших населених пунктах, які мають об’єкти підвищеної небезпеки, а також у воєнний час, основним способом захисту населення є евакуація і розміщення його у позаміській зоні.

Медичний захист проводиться для зменшення ступеня ураження людей, своєчасного надання допомоги постраждалим та їх лікування, забезпечення епідеміологічного благополуччя в районах надзвичайних ситуацій.

Радіаційний і хімічний захист включає заходи щодо виявлення і оцінки радіаційної та хімічної обстановки, організацію і здійснення дозиметричного та хімічного контролю, розроблення типових режимів радіаційного захисту, забезпечення засобами індивідуального захисту, організацію і проведення спеціальної обробки.

Евакуаційні заходи, які проводяться в містах та інших населених пунктах, які мають об’єкти підвищеної небезпеки, а також у воєнний час, основним способом захисту населення є евакуація і розміщення у позаміській зоні [20].

# РОЗДІЛ 5
# ВИЗНАЧЕННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО CI/CD-ПРОЦЕСУ ТА ОЦІНКА ТРУДОМІСТКОСТІ І ВАРТОСТІ ЙОГО ВПРОВАДЖЕННЯ

Оцінювання ефективності побудованого CI/CD-процесу є важливим етапом завершення всієї роботи, оскільки саме кількісні показники дозволяють визначити, наскільки автоматизація розгортання вебзастосунку виправдовує витрати на її розроблення та подальшу підтримку. У цьому розділі виконується розрахунок економічних і часових вигід, трудомісткості створення інструментарію, а також вартості розробленої системи автоматизації. Узагальнення цих параметрів дає змогу сформувати комплексний висновок щодо доцільності впровадження побудованого CI/CD-процесу у порівнянні з традиційними ручними підходами до інтеграції та розгортання.

Першим параметром, що підлягає аналізу, є скорочення часу, необхідного для збірки, тестування й розгортання нової версії програмного забезпечення. До впровадження автоматизації команда витрачала на виконання повного циклу інтеграції та розгортання у середньому від двох до трьох годин на одну ітерацію, оскільки значна частина операцій виконувалася вручну. Після впровадження Jenkins тривалість стандартної операціїорієнтовно складає лише декілька хвилин, а запуск виконується автоматично за подією у репозиторії. Нехай середній час ручного циклу складав 2,5 години, а автоматизованого — 10 хвилин (0,167 години). За формулою економії робочого часу

$$\boldsymbol{E_t = T_{\text{ручн}} - T_{\text{авт}}}$$

отримується

$$\boldsymbol{E_t = 2,5 - 0,167 = 2,333} \text{ год.}$$

Це означає, що одна операція CI/CD після автоматизації є у п’ятнадцять разів швидшою. Якщо протягом місяця команда виконує приблизно десять релізних або проміжних збірок, сумарна економія часу становить

$$\boldsymbol{E_{t\_\text{міс}} = 2,333 \times 10 = 23,33} \text{ год.}$$

Цей показник відображає час, який команда може спрямувати на реалізацію нових функцій, тестування або технічні поліпшення замість виконання рутинних дій.

Наступним елементом є визначення економічної вигоди у грошовому еквіваленті. Нехай середня погодинна ставка спеціаліста, який би виконував ручні процедури, становить 20 доларів США на годину. Тоді щомісячна економія коштів дорівнює

$$\boldsymbol{E_{\text{грн/міс}} = E_{t\_\text{міс}} \times 20 = 23,33 \times 20 = 466,6} \text{ дол.}$$

За рік економія становитиме близько

$$\boldsymbol{E_{\text{рік}} = 466,6 \times 12 = 5599,2} \text{ дол.}$$

У розрахунках не враховано додаткові непрямі вигоди, такі як зниження ризику помилок і вартості їх усунення, мінімізація простоїв або втрат продуктивності, однак у практиці розробки ці фактори часто є домінуючими й значно перевищують прямий фінансовий ефект.

Важливо також оцінити трудомісткість розроблення побудованої інфраструктури CI/CD. Для створення Jenkins-середовища, конфігурації агентів, налаштування збирання, тестування та розгортання було виконано приблизно 80 годин роботи DevOps-інженера. Трудомісткість визначається за формулою

$$\boldsymbol{T = t \times n},$$

де $\boldsymbol{t}$— кількість витрачених годин, $\boldsymbol{n}$— коефіцієнт корекції складності. Якщо складність проєкту оцінити як середню та прийняти $\boldsymbol{n = 1,1}$, то

$$\boldsymbol{T = 80 \times 1,1 = 88} \text{ год.}$$

Цей показник відображає зведену трудомісткість у людино-годинах, що відповідає типовим умовам розроблення аналогічних систем автоматизації.

Вартість розроблення системи CI/CD обчислюється виходячи з погодинної ставки фахівця. Для ставки 25 доларів за годину загальна вартість робіт становить

$$\boldsymbol{C = T \times 25 = 88 \times 25 = 2200} \text{ дол.}$$

Це одноразові витрати на створення інфраструктури. Порівняння вартості розробки із річною економією часу показує, що система окупається менш ніж за п’ять місяців:

$$\boldsymbol{t}_{\text{окупності}} = \frac{\boldsymbol{2200}}{\boldsymbol{466,6}} \approx \boldsymbol{4,7} \text{ міс.}$$

Окрім економічної складової, підвищення ефективності проявляється у збільшенні стабільності релізів. До автоматизації ймовірність виникнення помилки на етапі розгортання через людський фактор оцінювалася на рівні 10%. Після впровадження CI/CD цей показник зменшився до приблизно 1%. Ризикова економія визначається як

$$\boldsymbol{E}_{\text{риз}} = \boldsymbol{P}_{\text{ручн}} - \boldsymbol{P}_{\text{авт}},$$

де $\boldsymbol{P}_{\text{ручн}} = \boldsymbol{0,10}$, $\boldsymbol{P}_{\text{авт}} = \boldsymbol{0,01}$.

Тоді

$$\boldsymbol{E}_{\text{риз}} = \boldsymbol{0,10 - 0,01 = 0,09},$$

тобто ризик знизився на дев’ять відсоткових пунктів. У випадку великих комерційних систем саме ця величина є ключовою, оскільки помилки у продуктивному середовищі можуть спричиняти втрати, які значно перевищують витрати на автоматизацію.

Важливою є й оцінка експлуатаційної трудомісткості. Після завершення інсталяції Jenkins потребує лише періодичного нагляду та встановлення оновлень. Середня трудомісткість підтримки складає 3–4 години на місяць, тоді

як у разі ручного процесу підтримка вимагала б значно більше часу. Експлуатаційний показник визначається як

$$\boldsymbol{E}_{\text{експл}} = \boldsymbol{T}_{\text{ручн/міс}} - \boldsymbol{T}_{\text{авт/міс}},$$

де до автоматизації витрачалося приблизно 12 годин на місяць, після автоматизації - 4 години.Таким чином

$$\boldsymbol{E}_{\text{експл}} = \mathbf{12} - \mathbf{4} = \mathbf{8} \text{ год/міс.}$$

Це додаткова економія, яка також трансформується у зниження витрат.

Провівши дані розрахунки, можна стверджувати, що побудована система CI/CD забезпечує відчутний економічний та організаційний ефект. Суттєве скорочення тривалості циклів збірки та розгортання, зменшення кількості помилок, зниження навантаження на команду через автоматизовані процедури, а також швидка окупність розробки підтверджують доцільність впровадження такого інструментарію в умовах сучасної розробки вебзастосунків. Розроблений конвеєр не лише оптимізує роботу команди, але й забезпечує стабільність, відтворюваність і передбачуваність життєвого циклу програмного забезпечення, що має ключове значення для будь-яких комерційних проєктів.

# ЗАГАЛЬНІ ВИСНОВКИ

У кваліфікаційній роботі сформульовану мету досягнуто повною мірою. На основі послідовного аналізу теоретичних засад, обґрунтування вибору інструментарію та практичної реалізації конвеєра CI/CD розроблено цілісне рішення, орієнтоване на умови реального комерційного проєкту розробки вебзастосунку на мові Java. Робота поєднує концептуальний, методичний та прикладний рівні опису, що дозволяє розглядати побудований конвеєр як еталонний варіант для подальшого застосування та адаптації в аналогічних проєктах.

Досліджено сучасні підходи до розроблення та розгортання вебзастосунків, проаналізовано місце практик DevOps та CI/CD у життєвому циклі програмного забезпечення. Уточнено зміст понять безперервної інтеграції, безперервної доставки та безперервного розгортання, виділено їхні ролі у зменшенні часових витрат та ризиків, пов’язаних з людським фактором. Окрему увагу приділено типовым архітектурам конвеєрів CI/CD і варіантам інтеграції засобів тестування, моніторингу й безпеки. Це дозволило сформувати концептуальну модель процесу автоматизованого розгортання вебзастосунку, яка стала методологічною основою наступних етапів дослідження.

Виконано обґрунтування, вибір та деталізований опис інструментарію вирішення задачі. На основі аналізу популярних платформ CI/CD, таких як GitLab CI/CD, GitHub Actions, Azure Pipelines, CircleCI, обґрунтовано доцільність використання Jenkins як базового сервера автоматизації. Показано, що Jenkins відповідає вимогам до гнучкості, розширюваності, незалежності від конкретного хмарного провайдера, можливості локального розгортання та навчального використання. Розглянуто архітектуру Jenkins з контролером і агентами, принципи масштабування та інтеграції з іншими компонентами інфраструктури, а також типові варіанти побудови конвеєрів для контейнеризованих і класичних вебзастосунків. У межах розділу спроєктовано цільову архітектуру CI/CD-

процесу для умовного вебзастосунку з виділенням етапів збірки, тестування та розгортання.

Реалізовано та експериментально перевірено побудований CI/CD-процес. На основі змодельованої задачі, наближеної до умов реального комерційного проєкту, розгорнуто сервер Jenkins у хмарному середовищі, налаштовано доступ за допомогою SSH-ключів, встановлено необхідні плагіни й інтеграцію з репозиторієм вихідного коду. Детально описано конфігурацію агентів, що відповідають за збірку та тестування, реалізовано завдання для автоматизованої збірки Java-проєкту за допомогою Maven, запуску модульних і інтеграційних тестів, публікації результатів перевірок та надсилання сповіщень розробникам. На наступному етапі до конвеєра додано складові безперервної доставки й розгортання, налаштовано послідовність переходу від збірки до staging- та production-середовищ із використанням окремих завдань Jenkins і регресійного тестування. У підсумку отримано повний ланцюжок від коміту у репозиторії до оновлення продуктивної версії вебзастосунку.

Сукупний аналіз результатів показує, що запропонований CI/CD-процес дозволяє забезпечити регулярне постачання нових версій вебзастосунку з прогнозованою якістю, скоротити час на ручні операції, підвищити відтворюваність середовищ та прозорість життєвого циклу програмного забезпечення. Важливим практичним результатом роботи є детально задокументований набір налаштувань Jenkins, сценаріїв збірки та взаємодії завдань, який може бути використаний як шаблон для впровадження CI/CD у реальних проєктах. Запропонована архітектура конвеєра є масштабованою та може бути розширена за рахунок інтеграції додаткових видів тестування, механізмів безпеки, контейнеризації та оркестрації.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Atlassian. What is DevOps? [Електронний ресурс]. – Режим доступу: https://www.atlassian.com/devops – Дата звернення: 19.11.2025.

2. DevOps Lifecycle: Different Phases in DevOps [Електронний ресурс] / BrowserStack. – Режим доступу: https://www.browserstack.com/guide/devops-lifecycle – Дата звернення: 19.11.2025.

3. DevOps Principles and Best Practices: A Complete Guide [Електронний ресурс] / OutSystems. – Режим доступу: https://www.outsystems.com/application-development/software-development-guide/devops-meaning-and-fundamentals/ – Дата звернення: 19.11.2025.

4. What is CI/CD? Continuous integration and continuous delivery explained [Електронний ресурс] / Red Hat. – Режим доступу: https://www.redhat.com/en/topics/devops/what-is-ci-cd – Дата звернення: 19.11.2025.

5. Continuous delivery [Електронний ресурс] / Atlassian. – Режим доступу: https://www.atlassian.com/continuous-delivery – Дата звернення: 19.11.2025.

6. DevOps CI/CD tutorials [Електронний ресурс] / Atlassian. – Режим доступу: https://www.atlassian.com/devops/continuous-delivery-tutorials – Дата звернення: 19.11.2025.

7. What is CI/CD? Continuous Integration and Delivery [Електронний ресурс] / Codefresh. – Режим доступу: https://codefresh.io/learn/ci-cd/ – Дата звернення: 19.11.2025.

8. CI/CD Process: Flow, Stages, and Critical Best Practices [Електронний ресурс] / Codefresh. – Режим доступу: https://codefresh.io/learn/ci-cd-pipelines/ci-cd-process-flow-stages-and-critical-best-practices/ – Дата звернення: 19.11.2025.

9. What is a CI/CD pipeline? [Електронний ресурс] / Red Hat. – Режим доступу: https://www.redhat.com/en/topics/devops/what-cicd-pipeline – Дата звернення: 19.11.2025.

10. What is the CI/CD pipeline and CI/CD security [Електронний ресурс] / Palo Alto Networks. – Режим доступу: https://www.paloaltonetworks.com/cyberpedia/what-is-the-ci-cd-pipeline-and-ci-cd-security – Дата звернення: 19.11.2025.

11. What Is CI/CD Automation? Definition, Benefits, and the Best Tool [Електронний ресурс] / Qameta. – Режим доступу: https://qameta.io/blog/ci-cd-automation/ – Дата звернення: 19.11.2025.

12. Cloud-native CI/CD on Red Hat OpenShift [Електронний ресурс] / Red Hat. – Режим доступу: https://www.redhat.com/en/technologies/cloud-computing/openshift/ci-cd – Дата звернення: 19.11.2025.

13. Terraform Documentation [Електронний ресурс] / HashiCorp. – Режим доступу: https://developer.hashicorp.com/terraform/docs – Дата звернення: 19.11.2025.

14. What is Terraform? [Електронний ресурс] / HashiCorp. – Режим доступу: https://developer.hashicorp.com/terraform/intro – Дата звернення: 19.11.2025.

15. What is Infrastructure as Code with Terraform? [Електронний ресурс] / HashiCorp. – Режим доступу: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code – Дата звернення: 19.11.2025.

16. CI/CD pipelines [Електронний ресурс] / GitLab Documentation. – Режим доступу: https://docs.gitlab.com/ci/pipelines/ – Дата звернення: 19.11.2025.

17. Get started with GitLab CI/CD [Електронний ресурс] / GitLab Documentation. – Режим доступу: https://docs.gitlab.com/ci/ – Дата звернення: 19.11.2025.

18. GitHub Actions documentation [Електронний ресурс] / GitHub Docs. – Режим доступу: https://docs.github.com/actions – Дата звернення: 19.11.2025.

19. Workflow syntax for GitHub Actions [Електронний ресурс] / GitHub Docs. – Режим доступу: https://docs.github.com/actions/using-workflows/workflow-syntax-for-github-actions – Дата звернення: 19.11.2025.

20. Jenkins [Електронний ресурс] / Jenkins Project. – Режим доступу: https://www.jenkins.io/ – Дата звернення: 19.11.2025.

21. Jenkins User Documentation [Електронний ресурс] / Jenkins Project. – Режим доступу: https://www.jenkins.io/doc/ – Дата звернення: 19.11.2025.

22. Pipeline: Getting Started, Creating your first Pipeline [Електронний ресурс] / Jenkins Documentation. – Режим доступу: https://www.jenkins.io/doc/book/pipeline/ – Дата звернення: 19.11.2025.

23. Architecting for Scale [Електронний ресурс] / Jenkins Documentation. – Режим доступу: https://www.jenkins.io/doc/book/scaling/architecting-for-scale/ – Дата звернення: 19.11.2025.

24. Using Jenkins agents [Електронний ресурс] / Jenkins Documentation. – Режим доступу: https://www.jenkins.io/doc/book/using/using-agents/ – Дата звернення: 19.11.2025.

25. Jenkins Plugins Index [Електронний ресурс] / Jenkins Project. – Режим доступу: https://plugins.jenkins.io/ – Дата звернення: 19.11.2025.

26. Blue Ocean [Електронний ресурс] / Jenkins Documentation. – Режим доступу: https://www.jenkins.io/doc/book/blueocean/ – Дата звернення: 19.11.2025.

27. Create a Pipeline in Blue Ocean [Електронний ресурс] / Jenkins Documentation. – Режим доступу: https://www.jenkins.io/doc/tutorials/create-a-pipeline-in-blue-ocean/ – Дата звернення: 19.11.2025.

28. What is Jenkins? A Guide to CI/CD [Електронний ресурс] / CloudBees. – Режим доступу: https://www.cloudbees.com/jenkins/what-is-jenkins – Дата звернення: 19.11.2025.

29. Jenkins Architecture Explained – Beginners Guide to Jenkins Components [Електронний ресурс] / DevOpsCube. – Режим доступу: https://devopscube.com/jenkins-architecture-explained/ – Дата звернення: 19.11.2025.

30. Jenkins Architecture [Електронний ресурс] / KodeKloud Notes. – Режим доступу: https://notes.kodekloud.com/docs/Certified-Jenkins-Engineer/Jenkins-Architecture – Дата звернення: 19.11.2025.

31. Jenkins Deployment Stages and Pipelines [Електронний ресурс] / Earthly Blog. – Режим доступу: https://earthly.dev/blog/jenkins-stages-pipelines/ – Дата звернення: 19.11.2025.

32. What is Jenkins? Key Features, Benefits, and How It Works [Електронний ресурс] / TuxCare. – Режим доступу: https://tuxcare.com/blog/what-is-jenkins/ – Дата звернення: 19.11.2025.

33. Deploy Jenkins [Електронний ресурс] / Railway. – Режим доступу: https://railway.com/deploy/jenkins – Дата звернення: 19.11.2025.

34. Pipeline Visualization and Management in Jenkins [Електронний ресурс] / Scaler Topics. – Режим доступу: https://www.scaler.com/topics/pipeline-visualization-and-management-in-jenkins/ – Дата звернення: 19.11.2025.

35. Jenkins: pros/cons, installation, and 8 modern alternatives [Електронний ресурс] / Octopus Deploy. – Режим доступу: https://octopus.com/devops/jenkins/ – Дата звернення: 19.11.2025.

36. DevOps Using Jenkins, Docker, and Kubernetes [Електронний ресурс] / BETSOL. – Режим доступу: https://www.betsol.com/blog/devops-using-jenkins-docker-and-kubernetes/ – Дата звернення: 19.11.2025.

37. Configuring Jenkins Pipeline for Kubernetes. Jenkins Project: Building CI/CD Pipeline for Scalable Web Applications [Електронний ресурс] / KodeKloud. – Режим доступу: https://notes.kodekloud.com/docs/Jenkins-Project-Building-CICD-Pipeline-for-Scalable-Web-Applications/Kubernetes/Configuring-Jenkins-Pipeline-for-Kubernetes – Дата звернення: 19.11.2025.

38. Пістун І. П., Тимочко В.О., Городецький І. М.. Березовецький А. П. Охорона праці (гігієна праці та виробнича санітарія): навч. посібн. / за ред. І.П. Пістуна. Львів: Тріада плюс, 2011. 224 с.

39. Охорона праці: практикум /І.П. Пістун, Ю.В. Кіт, А.П.Березовецький – Суми: Університетська книга, 2000. –205с.