МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Юрій ІСКРЕНКО
«_____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»
ЗА СПЕЦІАЛЬНІСТЮ 123 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

Тема: Системи розгортання додатків у віртуальному середовищі

Виконавець: Вадим СЕМЧУК

Керівник: Микола ГУЗІЙ

Нормоконтролер: Наталія ФОМІНА

Київ 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ЗАТВЕРДЖУЮ
Завідувач кафедри КСМ
_____ Юрій ІСКРЕНКО
«_____»_____2025 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Семчука Вадима Павловича

1. Тема кваліфікаційної роботи(проєкту): «Системи розгортання додатків у віртуальному середовищі»
Затверджена наказом ректора від «24» жовтня 2025 р. №2344/ст В
2. Термін виконання роботи: з 29.09.2025 по 31.12.2025
3. Вихідні дані до роботи: мова програмування *Python*, вебфреймворк *Flask*, система управління базами даних *SQLite*, програмний гіпервізор *VirtualBox*, інструмент автоматизації віртуальних середовищ *Vagrant*, платформа контейнеризації *Docker*, система контролю версій *Git*, сервіс безперервної інтеграції та розгортання *GitHub Actions*.
4. Зміст пояснювальної записки: вступ, аналіз теоретичних основ віртуалізації та сучасних підходів до розгортання додатків, обґрунтування вибору програмних та інфраструктурних засобів, побудова архітектури систем розгортання вебдодатків у віртуальному середовищі, реалізація тестового вебдодатка та сценаріїв ручного й автоматизованого розгортання, експериментальне дослідження ефективності методів розгортання, багатокритеріальна оцінка результатів, висновки, список використаних бібліографічних джерел.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

Представлення результатів роботи у вигляді презентації.

6. Календарний план

№ п/п	Завдання	Термін виконання етапів	Підпис керівника
1.	Затвердження теми роботи	29.09.2025	
2.	Аналіз літературних джерел та існуючих підходів	03.10.2025	
3.	Формулювання мети, завдань, об’єкта та предмета дослідження	17.10.2025	
4.	Аналіз методів і моделей розгортання	19.11.2025	
5.	Розробка та реалізація прототипу системи розгортання	21.11.2025	
6.	Проведення експерементальних досліджень	05.12.2025	
7.	Оформлення пояснювальної записки	12.12.2025	
8.	Остаточне оформлення роботи та висновків	18.12.2025	
9.	Підготовка презентації та доповіді до захисту	23.12.2025	
10.	Захист кваліфікаційної роботи	23.12.2025 – 31.12.2025	

7. Дата видачі завдання: «29» __вересня__ 2025 р.

Керівник кваліфікаційної роботи: Микола ГУЗІЙ _____
(підпис керівника)

Завдання прийняв до виконання: Вадим СЕМЧУК _____
(підпис виконавця)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Системи розгортання додатків у віртуальному середовищі»: 101 сторінока, 43 рисунка, 12 таблиць, 39 літературних джерел.

Об’єкт дослідження – системи розгортання вебдодатків у віртуальному середовищі.

Предмет дослідження – методи та технології автоматизованого функціонування систем розгортання додатків.

Мета роботи – дослідження та розробка систем автоматизованого розгортання додатків у віртуальному середовищі.

Технічні та програмні засоби – мова програмування *Python*, вебфреймворк *Flask*, система управління базами даних *SQLite*, система віртуалізації *VirtualBox,* інструмент автоматизації *Vagrant*, система контейнеризації Docker, засоби автоматизованого розгортання *CI/CD*, персональний комп’ютер під управлінням операційної системи *Linux*/*Windows*.

Основні характеристики та показники – система забезпечує повний цикл розгортання вебдодатка та підтримує різні рівні автоматизації; ефективність оцінюється за часом, стабільністю, кількістю ручних дій та інтегральним показником.

Отримані результати та їх новизна – виконано кількісне порівняльне дослідження систем розгортання з використанням статистичної обробки даних та інтегрального показника ефективності.

Рекомендації щодо використання результатів – результати можуть застосовуватись під час проєктування та експлуатації систем розгортання вебдодатків у віртуальних середовищах і в навчальному процесі.

СИСТЕМИ РОЗГОРТАННЯ, ВІРТУАЛЬНЕ СЕРЕДОВИЩЕ, ВЕБДОДАТОК, АВТОМАТИЗАЦІЯ, КОНТЕЙНЕРИЗАЦІЯ, *FLASK*, *PYTHON*.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ. ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення.

ПС – програмна система.

ВД – вебдодаток.

БД – база даних.

СУБД – система управління базами даних.

ВМ – віртуальна машина.

ВС – віртуальне середовище.

ГВ – гіпервізор.

VCS – англ. *Version Control System* – система контролю версій.

Git – англ. *Git* – розподілена система контролю версій.

CI/*CD* – англ. *Continuous Integration* / *Continuous Deployment* – безперервна інтеграція та розгортання.

IaC – англ. *Infrastructure as Code* – інфраструктура як код.

Docker – англ. *Docker* – платформа контейнеризації додатків.

Контейнер – ізольоване середовище виконання додатка з необхідними залежностями.

CPU – англ. *Central Processing Unit* – центральний процесор.

RAM – англ. *Random Access Memory* – оперативна пам’ять.

HTTP – англ. *HyperText Transfer Protocol* – протокол передачі гіпертексту.

URL – англ. *Uniform Resource Locator* – уніфікований покажчик ресурсу.

SQLite – англ. *SQLite* – вбудована реляційна система управління базами даних.

AHP – англ. *Analytic Hierarchy Process* – метод аналізу ієрархій Сааті.

CR – англ. *Consistency Ratio* – коефіцієнт узгодженості матриці парних порівнянь.

ЕОМ – електронна обчислювальна машина.

ВСТУП

Сучасні програмні системи все частіше розгортаються та експлуатуються у віртуалізованих середовищах, що зумовлено зростанням вимог до гнучкості, масштабованості та раціонального використання обчислювальних ресурсів. Віртуалізація дозволяє відокремити програмне забезпечення від апаратної платформи, спростити керування інфраструктурою та забезпечити швидке відновлення працездатності сервісів. У таких умовах процес розгортання програмних додатків перестає бути допоміжною операцією і перетворюється на один із ключових факторів стабільної та безперервної роботи інформаційних систем.

Особливого значення набуває розгортання вебдодатків, які є основою більшості сучасних інформаційних сервісів. Неефективна організація процесу розгортання призводить до збільшення часу простоїв, підвищення ризику помилок під час оновлень та суттєвої залежності від людського фактора. Традиційні ручні підходи, що тривалий час залишалися поширеними, дедалі гірше відповідають вимогам сучасних середовищ, у яких оновлення відбуваються часто, а інфраструктура змінюється динамічно.

Актуальність теми. Сучасні інформаційні системи дедалі частіше функціонують у віртуалізованих середовищах, що зумовлено потребою у гнучкості, масштабованості та ефективному використанні обчислювальних ресурсів. За таких умов процес розгортання програмних додатків набуває ключового значення, оскільки безпосередньо впливає на стабільність роботи сервісів, швидкість впровадження змін та здатність системи адаптуватися до зовнішніх і внутрішніх викликів. Неефективна організація розгортання призводить до зростання часу простоїв, підвищення ризику помилок і значної залежності від людського фактора, що є критичним для сучасних веборієнтованих сервісів.

Особливої актуальності дана проблематика набуває в умовах обмежених ресурсів і нестабільності інфраструктури, характерних для сучасного етапу

розвитку інформаційних систем в Україні. Часті оновлення програмного забезпечення, необхідність швидкого відновлення працездатності сервісів та підвищені вимоги до надійності вимагають переосмислення традиційних підходів до організації процесів розгортання. У таких умовах системний аналіз методів розгортання вебдодатків у віртуальному середовищі та обґрунтування шляхів підвищення їх ефективності стають важливим практичним завданням, спрямованим на зменшення впливу людського фактора та підвищення стійкості інформаційних систем.

У межах даного дослідження система розглядається як сукупність взаємопов’язаних процесів, методів і технічних засобів, що забезпечують приведення програмного додатка у працездатний стан у віртуальному середовищі з наперед заданими характеристиками стабільності, відтворюваності та керованості.

Мета і завдання кваліфікаційної роботи. Метою даної кваліфікаційної роботи є дослідження та порівняльна оцінка ефективності систем розгортання вебдодатків у віртуальному середовищі з різним рівнем автоматизації з метою визначення найбільш доцільного підходу для використання в умовах обмежених ресурсів і нестабільної інфраструктури.

Для досягнення поставленої мети в роботі необхідно вирішити такі завдання:

1. проаналізувати теоретичні основи віртуалізації та особливості середовищ виконання програмних систем;
2. дослідити сутність і обмеження традиційного ручного розгортання вебдодатків;
3. сформувати узагальнену модель системи розгортання вебдодатків у віртуальному середовищі;
4. обґрунтувати критерії оцінювання ефективності процесу розгортання;
5. здійснити порівняльний аналіз різних підходів до розгортання на основі кількісних показників;
6. сформулювати висновки та рекомендації щодо підвищення ефективності розгортання вебдодатків.

Об’єкт і предмет дослідження. Об’єктом дослідження є системи розгортання вебдодатків у віртуальному середовищі. Предметом дослідження є процеси та методи, що визначають функціонування й ефективність систем розгортання вебдодатків у віртуальному середовищі.

Методи дослідження. Для досягнення поставленої мети у роботі застосовано загальнонаукові та прикладні методи дослідження, зокрема аналіз і синтез для узагальнення теоретичних положень, моделювання для формалізації процесу розгортання, а також методи кількісного аналізу для оцінювання ефективності різних підходів. Використання статистичних методів дозволило забезпечити об’єктивність і достовірність отриманих результатів.

Наукова новизна отриманих результатів. Наукова новизна роботи полягає у формалізованому підході до розгляду розгортання вебдодатків як цілісної системи, що складається з взаємопов’язаних процесів і методів, а також у побудові узагальненої моделі оцінювання ефективності різних підходів до розгортання у віртуальному середовищі.

Практичне значення отриманих результатів. Практичне значення роботи полягає у можливості використання отриманих висновків і рекомендацій для підвищення ефективності розгортання вебдодатків у реальних інформаційних системах. Результати дослідження можуть бути застосовані під час проєктування та експлуатації програмних рішень у середовищах з обмеженими ресурсами, а також у навчальному процесі при підготовці фахівців у галузі програмної інженерії та інформаційних технологій.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗГОРТАННЯ ДОДАТКІВ У ВІРТУАЛЬНИХ СЕРЕДОВИЩАХ

1.1 Поняття віртуалізації та її роль у розгортанні додатків

Віртуалізація є фундаментальною технологією, що трансформувала підходи до проєктування, розгортання та експлуатації програмного забезпечення, ставши базисом для побудови сучасних центрів обробки даних та хмарних інфраструктур. У контексті розгортання додатків ця технологія визначається як процес створення програмно-апаратного прошарку абстракції, що дозволяє відокремити обчислювальні ресурси від фізичного обладнання. Такий підхід змінює традиційну парадигму, де одна операційна система монопольно керувала всіма ресурсами сервера.

Ключовим елементом цієї архітектури є монітор віртуальних машин, або гіпервізор. Це програмне забезпечення, яке емулює апаратні ресурси (процесор, пам'ять, дисковий простір, мережеві інтерфейси) і динамічно розподіляє їх між ізольованими гостьовими операційними системами. Це дозволяє декільком середовищам виконання функціонувати одночасно на одній фізичній машині, не заважаючи одне одному [2].

Для глибшого розуміння ролі віртуалізації важливо розрізняти типи гіпервізорів, оскільки вони визначають ефективність та архітектуру середовища розгортання. Існують гіпервізори першого типу («*bare-metal*»), які встановлюються безпосередньо на апаратне забезпечення, забезпечуючи високу продуктивність, та гіпервізори другого типу («*hosted*»), що працюють поверх основної операційної системи і частіше використовуються для локальної розробки та тестування.

Для наочного зіставлення класичної моделі розгортання програмного забезпечення та підходу з використанням віртуалізації доцільно порівняти їх архітектурні особливості, що наведені на рисунку 1.1.

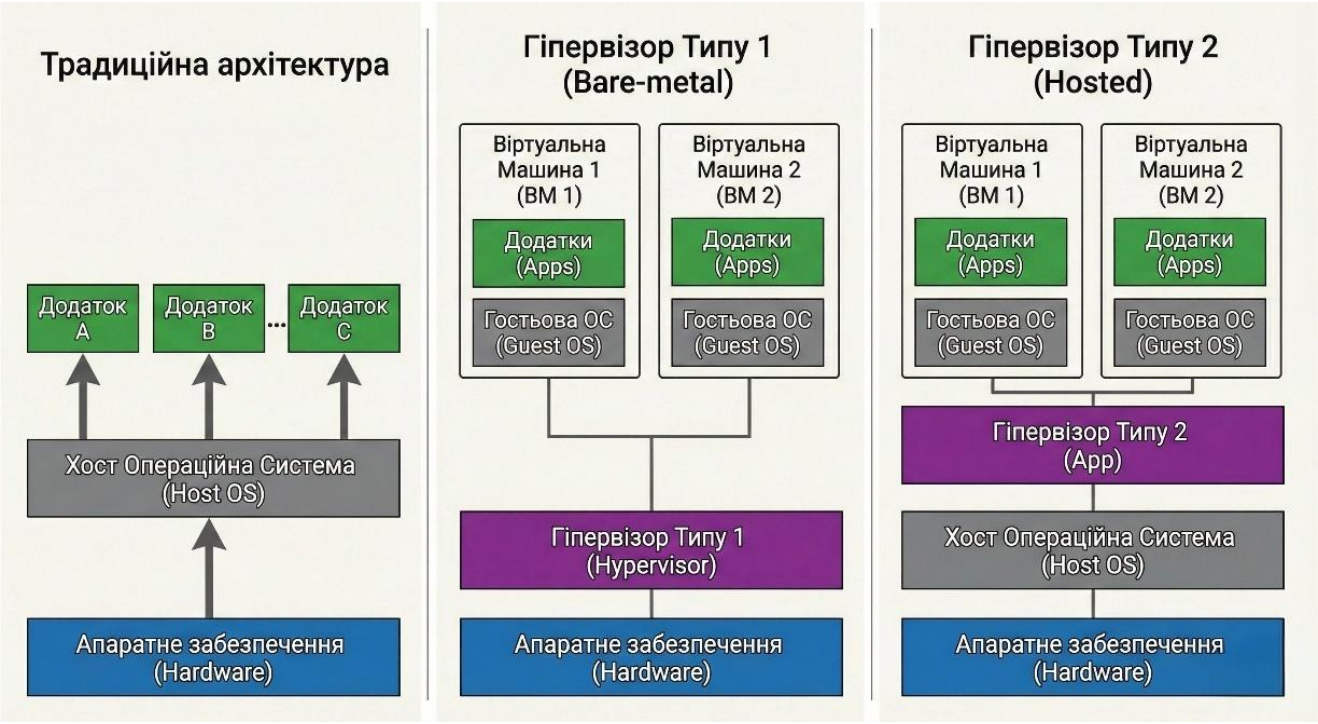

Рисунок 1.1 – Порівняння традиційної та віртуалізованої архітектур

Критичною характеристикою віртуалізації для сучасних процесів розгортання (*Deployment*) є інкапсуляція. Віртуальна машина (ВМ) по суті є повним набором віртуального обладнання та програмного забезпечення, упакованим у декілька файлів (конфігурація, образ диска, знімок стану пам'яті). Ця властивість забезпечує безпрецедентну мобільність середовищ: ВМ можна легко копіювати, переміщувати між фізичними серверами, архівувати або миттєво відновлювати. Для процесу розгортання це означає гарантію ідентичності конфігурацій на етапах розробки, тестування та промислової експлуатації, мінімізуючи ризики виникнення помилок сумісності [1].

Окрім мобільності, віртуалізація забезпечує механізм суворої ізоляції. Кожна віртуальна система функціонує у власному захищеному адресному просторі. Гіпервізор гарантує, що збій або компрометація одного додатка у ВМ не вплине на

стабільність інших сервісів чи основного хоста. Це дозволяє реалізувати концепцію безпечних «пісочниць» (*sandboxes*) для перевірки нового коду перед інтеграцією, а також ефективно утилізувати ресурси обладнання, безпечно розміщуючи різні за вимогами додатки на одному фізичному сервер.

Еволюція інфраструктурних технологій проходила декілька етапів: від розгортання програмного забезпечення безпосередньо на фізичних серверах до появи віртуальних машин, що забезпечили ізоляцію та стандартизацію середовищ. Подальший розвиток привів до контейнеризації, яка зменшила накладні витрати та дозволила запускати ізольовані застосунки на спільному ядрі операційної системи. Така послідовність еволюційних переходів сформувала сучасний стек технологій розгортання, у якому віртуалізація виступає фундаментальним шаром [32].

Механізм абстракції та ізоляції обчислювальних ресурсів, який забезпечується гіпервізором, схематично зображено на рисунку 1.2.

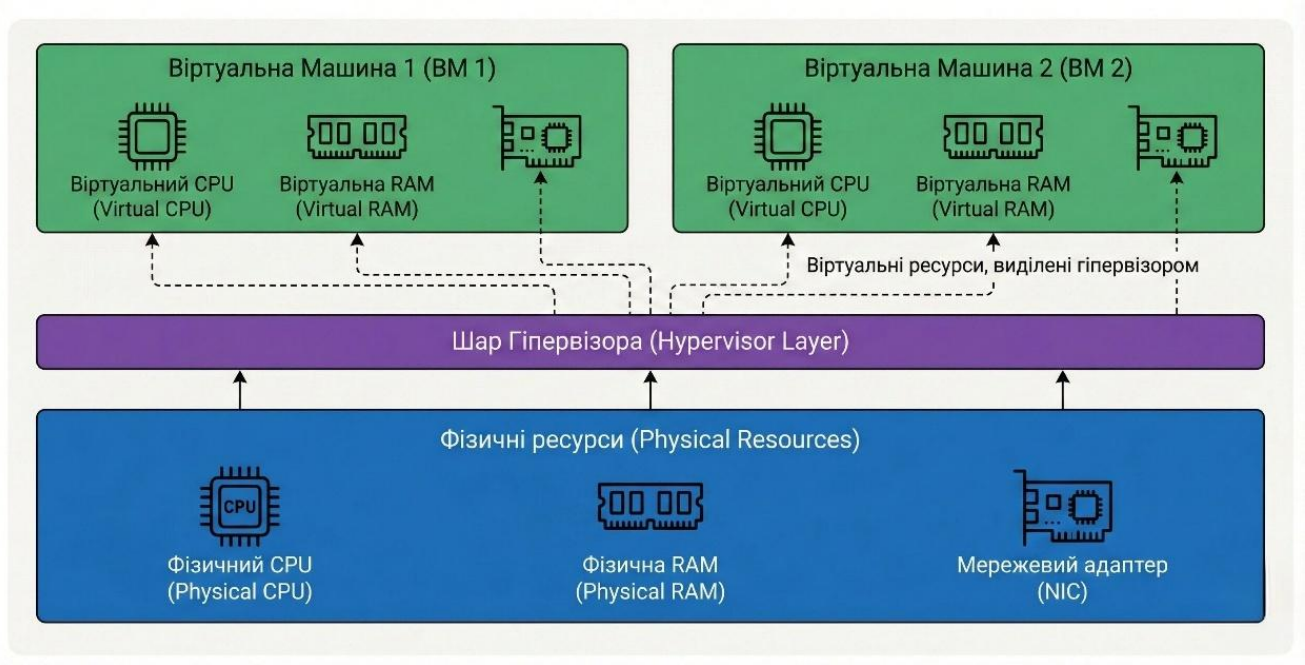

Рисунок 1.2 – Абстракція та ізоляція ресурсів за допомогою гіпервізора

Сучасні стратегії, такі як інфраструктура як код (*Infrastructure as Code - IaC*), безпосередньо спираються на можливості віртуалізації. Завдяки програмним інтерфейсам (*API*), які надають платформи віртуалізації, процеси створення,

налаштування та знищення серверів можуть бути повністю автоматизовані. Це дозволило перейти від ручного адміністрування до динамічного управління ресурсами, де нові екземпляри додатків розгортаються за лічені хвилини в залежності від навантаження. Саме ця автоматизація та абстракція уможливили появу хмарних обчислень (*Cloud Computing*), де ресурси надаються користувачеві як сервіс із можливістю еластичного масштабування. Віртуалізація є тим прихованим двигуном, що забезпечує роботу публічних та приватних хмар.[3]

Місце віртуалізації у сучасному стеку технологій розгортання програмних систем та її взаємозв'язок з іншими інфраструктурними підходами представлено на рисунку 1.3.

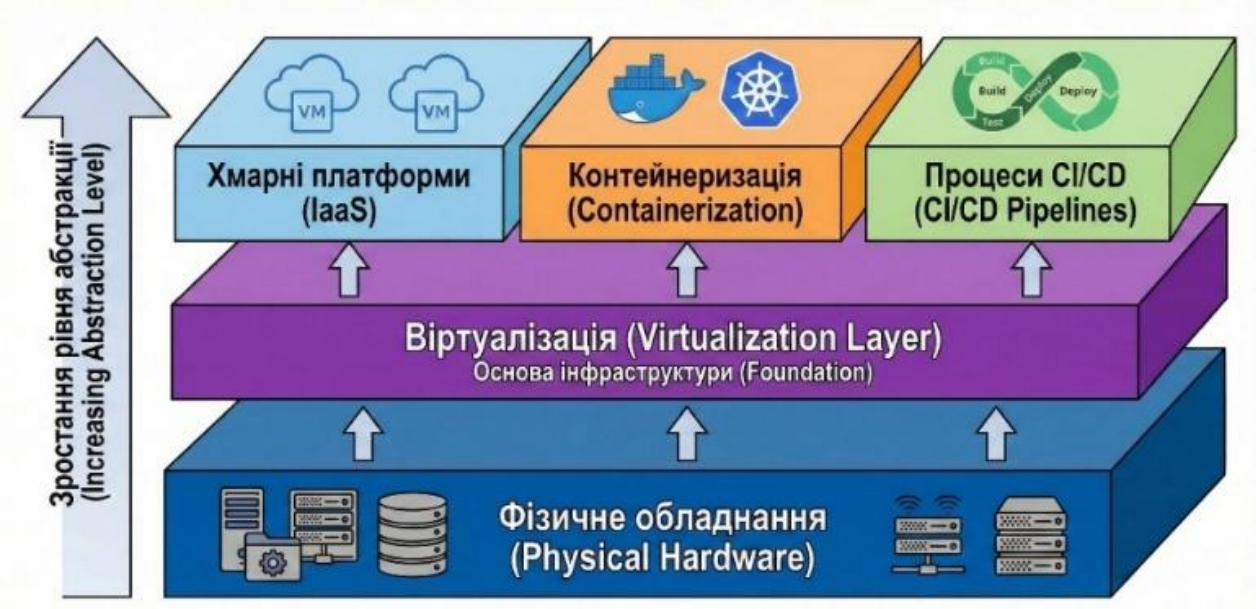

Рисунок 1.3 – Віртуалізація як фундамент сучасного стека технологій розгортання

У сучасних підходах до автоматизації віртуалізація відіграє роль базової технології, яка забезпечує відтворюваність та ізольованість середовищ. Саме на ній ґрунтуються методології *DevOps* та *CI*/*CD*, що передбачають швидке створення інфраструктури, паралельне тестування та стандартизоване розгортання у різних середовищах. Завдяки програмним *API* платформи віртуалізації стали ключовим компонентом автоматизованих конвеєрів доставки програмного забезпечення [9].

Хоча сьогодні популярності набуває контейнеризація, важливо розуміти, що вона не замінює віртуалізацію, а доповнює її. У більшості промислових середовищ контейнери розгортаються всередині віртуальних машин для забезпечення додаткового рівня безпеки, кращого керування ресурсами та сумісності з існуючою інфраструктурою. Таким чином, віртуалізація залишається критично важливим базовим шаром, що забезпечує гнучкість, надійність та ефективність сучасних систем розгортання програмного забезпечення [33].

1.2 Середовища виконання програмних систем та їх особливості

Віртуалізовані середовища розгортання характеризуються значною різноманітністю, що зумовлено відмінностями в архітектурі гіпервізорів, механізмах ізоляції процесів та рівнях абстракції, що надаються адміністратору системи. Для коректного аналізу методів розгортання вебдодатків необхідно систематизувати ці середовища, оскільки саме архітектура платформи визначає доступні інструменти автоматизації, рівень безпеки та загальну стабільність програмних систем. Класифікацію доцільно проводити за двома основними напрямами: рівнем віртуалізації ресурсів та моделлю управління розгортанням [3].

За рівнем абстракції ресурсів виділяють дві фундаментальні категорії середовищ: апаратну віртуалізацію та віртуалізацію на рівні операційної системи (контейнеризацію). У класичній моделі апаратної віртуалізації ключову роль відіграє гіпервізор, який створює повноцінні віртуальні машини. Кожен такий екземпляр містить власне ядро операційної системи, повний набір системних бібліотек та драйверів. Це гарантує найвищий рівень ізоляції, оскільки процеси однієї машини жодним чином не перетинаються з процесами іншої, проте такий підхід супроводжується значними накладними витратами ресурсів на підтримку роботи гостьових ОС.

На противагу цьому, контейнеризація представляє собою полегшений тип віртуалізації, де ізоляція досягається на рівні простору користувача спільного ядра хостової операційної системи. Контейнер функціонує як ізольований процес, що

має власний файловий простір та мережевий стек, але використовує ресурси ядра спільно з іншими контейнерами. Така архітектура кардинально зменшує час запуску та споживання пам’яті, що робить її оптимальною для мікросервісних архітектур та систем, що потребують швидкого масштабування.

Узагальнену класифікацію віртуальних середовищ виконання програмних систем залежно від рівня абстракції ресурсів наведено на рисунку 1.4.

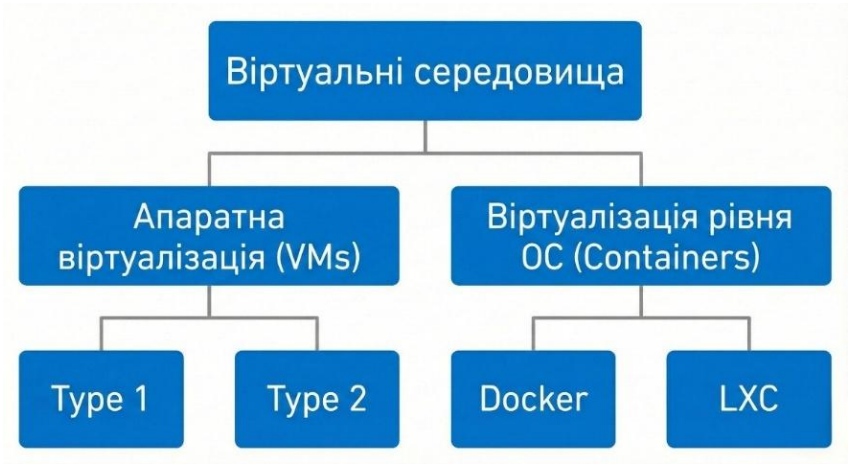

Рисунок 1.4 – Класифікація віртуальних середовищ за рівнем абстракції

Архітектурні відмінності середовищ безпосередньо формують моделі розгортання програмного забезпечення. Можна виділити два основні підходи: імперативний (традиційний) та декларативний. Традиційна модель, характерна для класичних віртуальних машин, передбачає покрокове виконання інструкцій: ручне створення ВМ, інсталяцію залежностей, редагування конфігураційних файлів. Головним недоліком цього підходу є «дрейф конфігурацій» (*configuration drift*), коли з часом стан сервера відхиляється від еталонного через ручні втручання, що створює ризики для відтворюваності системи [4].

У контейнеризованих та хмарних середовищах домінує декларативна модель. У цьому випадку адміністратор описує бажаний кінцевий стан системи (наприклад, «потрібно запустити вебсервер версії X з портом *Y*») у вигляді конфігураційного коду, а спеціалізоване програмне забезпечення автоматично приводить середовище до цього стану. Цей підхід ліг в основу концепції «Інфраструктура як код»

(*Infrastructure as Code — IaC*), яка дозволяє керувати інфраструктурою аналогічно до програмного коду: з версіонуванням, тестуванням та автоматизованим застосуванням змін [6].

Однією з ключових причин поширення контейнеризації стала проблема невідповідності середовищ, відома як «*works on my machine*». Вона виникає через різні версії бібліотек, конфігурацій та залежностей, що відрізняються між локальним середовищем розробника і продуктивним сервером. Контейнери вирішують цю проблему шляхом упаковування застосунку разом з усіма залежностями та стандартним середовищем виконання, забезпечуючи повну однаковість середовищ [13].

Концептуальні відмінності між імперативною та декларативною моделями розгортання програмного забезпечення проілюстровано на рисунку 1.5.

Рисунок 1.5 – Порівняння імперативної та декларативної моделей розгортання

Декларативний підхід до керування середовищами дозволяє описувати інфраструктуру у вигляді конфігураційного коду, що робить процес розгортання ідемпотентним та відтворюваним. Такий метод усуває розбіжності між серверами

та мінімізує вплив людського фактора, оскільки результат визначається не послідовністю ручних команд, а формальним описом кінцевого стану системи.

Окремим еволюційним щаблем є моделі безперервного розгортання (*Continuous Deployment*), які інтегрують процеси створення середовища безпосередньо в життєвий цикл розробки. Вони базуються на автоматизованих Таким чином, вибір моделі середовища виконання прямо визначає складність процесу розгортання та ступінь можливості його автоматизації. У традиційних імперативних моделях більшість операцій виконується вручну, що створює ризики дрейфу конфігурацій, тоді як декларативні та контейнерні підходи орієнтовані на стандартизацію та автоматизацію [7].

Ускладнення середовищ виконання програмних систем, зростання кількості залежностей та необхідність їх узгодження суттєво впливають на процеси розгортання вебдодатків. За відсутності спеціалізованих засобів автоматизації ці процеси, як правило, виконуються вручну, що формує окремий клас проблем, пов’язаних із трудомісткістю, відтворюваністю та стабільністю результатів [12].

1.3 Ручне розгортання вебдодатків: сутність, структура процесу та ключові обмеження

Ручне розгортання вебдодатків історично є першим і найбільш інтуїтивним підходом до введення програмних продуктів в експлуатацію. У цьому методі всі етапи підготовки інфраструктури, налаштування середовища виконання та запуску застосунку виконуються адміністратором або розробником безпосередньо через інтерфейс командного рядка (*CLI*) або графічний інтерфейс. Сутність підходу полягає в імперативному стилі управління: оператор послідовно вводить команди, кожна з яких змінює стан системи. Хоча цей метод забезпечує максимальну гнучкість у вирішенні нестандартних проблем «тут і зараз», він формує технологічний процес, цілком залежний від людського фактора, уважності та кваліфікації виконавця [6].

Процес ручного розгортання можна структурувати як лінійну послідовність дій, що починається з підготовки серверної платформи («*provisioning*»). На цьому етапі адміністратор інсталює операційну систему, налаштовує користувачів, права доступу та мережеві екрани (*firewalls*). Наступним критичним етапом є конфігурація середовища виконання (*Runtime Environment*), що включає встановлення вебсерверів (*Nginx, Apache*), інтерпретаторів мов програмування та баз даних. Лише після підготовки бази відбувається перенесення коду — зазвичай через протоколи *FTP*/*SCP* або шляхом клонування репозиторію системи контролю версій. Фінальна фаза охоплює встановлення залежностей, міграцію бази даних та запуск служб. Складність цього алгоритму полягає в тому, що помилка на будь-якому ранньому етапі може проявитися лише у фіналі, змушуючи оператора повертатися до початку ланцюжка [7].

Узагальнену алгоритмічну структуру процесу ручного розгортання вебдодатка у віртуальному середовищі наведено на рисунку 1.6.

Рисунок 1.6 – Алгоритмічна структура процесу ручного розгортання

Фундаментальною проблемою ручного методу є явище, яке у професійній літературі отримало назву «дрейф конфігурацій» (*Configuration Drift*). Це процес поступового розходження стану серверів від задокументованого еталону через незафіксовані ручні зміни, «гарячі» виправлення (*hotfixes*) та часткові оновлення пакетів. З часом це призводить до появи так званих «*Snowflake Servers*» (унікальних серверів) — систем, конфігурація яких настільки унікальна і заплутана, що їх неможливо відтворити з нуля. Згідно з дослідженнями Хамбла та Фарлі, саме унікальність серверів є головною причиною страху перед розгортанням

(*deployment fear*), коли команди уникають оновлень через ризик зруйнувати крихку рівновагу працюючої системи.

Окрім дрейфу конфігурацій, ручне розгортання страждає від проблеми залежностей, відомої як «*Dependency Hell*». Оскільки всі додатки на сервері часто використовують спільні системні бібліотеки, оновлення бібліотеки для одного проєкту може порушити роботу іншого. У ручному режимі розв'язання конфліктів версій покладається на адміністратора, що потребує глибоких знань системної архітектури та значних часових витрат. Це робить горизонтальне масштабування системи неефективним: налаштування кожного нового вузла кластера вимагає повторення всього обсягу ручної роботи, що перетворює масштабування на процес з лінійною часовою складністю $O(n)$, на відміну від $O(1)$ в автоматизованих системах [7].

Відсутність ідемпотентності у ручних процесах означає, що повторне виконання тих самих команд може призвести до інший результатів або змінити стан системи у небажаний спосіб. Це суперечить сучасним практикам автоматизації, де ідемпотентність є базовою властивістю, що забезпечує передбачуваність і стабільність розгортань.

Також варто зазначити проблему документації. При ручному підході документація (*Runbooks*) існує окремо від самого процесу налаштування. Оскільки зміни в систему вносяться динамічно, інструкції швидко стають неактуальними. Це створює ситуацію, коли знання про інфраструктуру концентруються в головах окремих співробітників («*bus factor*»), що становить загрозу для стабільності бізнес-процесів організації. Відсутність ідемпотентності — властивості, коли повторне виконання операції не змінює результат, якщо система вже у цільовому стані, — призводить до того, що повторні спроби розгортання після збою можуть дублювати дані або викликати конфлікти конфігурацій [9].

Таким чином, незважаючи на відносно низький поріг входження, ручне розгортання характеризується низькою надійністю, обмеженою масштабованістю та високою ентропією системи. Сукупність цих чинників призводить до поступового накопичення неявних залежностей, неконсистентних налаштувань і

локальних рішень, які складно відтворити або формалізувати. У результаті процес розгортання перестає бути детермінованим і все більше залежить від індивідуального досвіду конкретного адміністратора, що суттєво підвищує ризики помилок у разі зростання навантаження або зміни складу команди.

Зі збільшенням кількості середовищ або екземплярів додатка трудомісткість ручного розгортання зростає непропорційно, оскільки кожен додатковий вузол вимагає повторення повного набору конфігураційних дій. Така нелінійна залежність ускладнює масштабування системи та робить процес планування ресурсів малопередбачуваним. Водночас відсутність формалізованого опису інфраструктури унеможливлює автоматичну перевірку коректності конфігурацій і створює передумови для виникнення прихованих збоїв у продуктивному середовищі.

Для оцінювання впливу рівня автоматизації на масштабованість та трудомісткість процесу розгортання доцільно порівняти відповідні підходи, що показано на рисунку 1.7.

Рисунок 1.7 – Порівняння трудомісткості ручного та автоматизованого розгортання при масштабуванні

Для більш повного розуміння практичних наслідків описаного підходу доцільно розглянути реальний процес ручного розгортання вебдодатка у віртуальному середовищі та проаналізувати його етапи з позицій трудомісткості, передбачуваності та залежності від людського фактора.

1.4 Практичний аналіз процесу ручного розгортання вебдодатка у віртуальному середовищі

Для поглибленого розуміння обмежень ручного підходу до розгортання доцільно розглянути реальний процес впровадження вебдодатка у віртуальному середовищі. На відміну від абстрактного теоретичного опису, практичний аналіз дозволяє виявити не лише формальну послідовність дій, а й супутні труднощі, часові витрати та потенційні джерела помилок, що виникають у процесі виконання кожного етапу. Особливу увагу при цьому доцільно приділити залежності результату від початкових налаштувань середовища та ступеня формалізації виконуваних операцій [22].

Підготовка віртуального середовища для ручного розгортання зазвичай розпочинається зі створення та первинної конфігурації віртуальної машини, встановлення базової операційної системи, а також налаштування мережевих параметрів і доступу до системи. Кожен із цих кроків потребує прийняття рішень, що ґрунтуються на досвіді адміністратора, і не завжди має однозначний або стандартизований характер. Унаслідок цього вже на початкових етапах формується варіативність конфігурацій, яка згодом ускладнює відтворення середовища та порівняння результатів розгортання.

Типові початкові етапи підготовки віртуального середовища для ручного розгортання вебдодатка проілюстровано на рисунку 1.8, де відображено базові операції зі створення віртуальної машини та виконання первинних налаштувань. Наведений приклад демонструє, що навіть до початку безпосереднього розгортання прикладного програмного забезпечення значна частина часу

витрачається на підготовчі дії, які в подальшому мають бути повторені для кожного нового середовища.

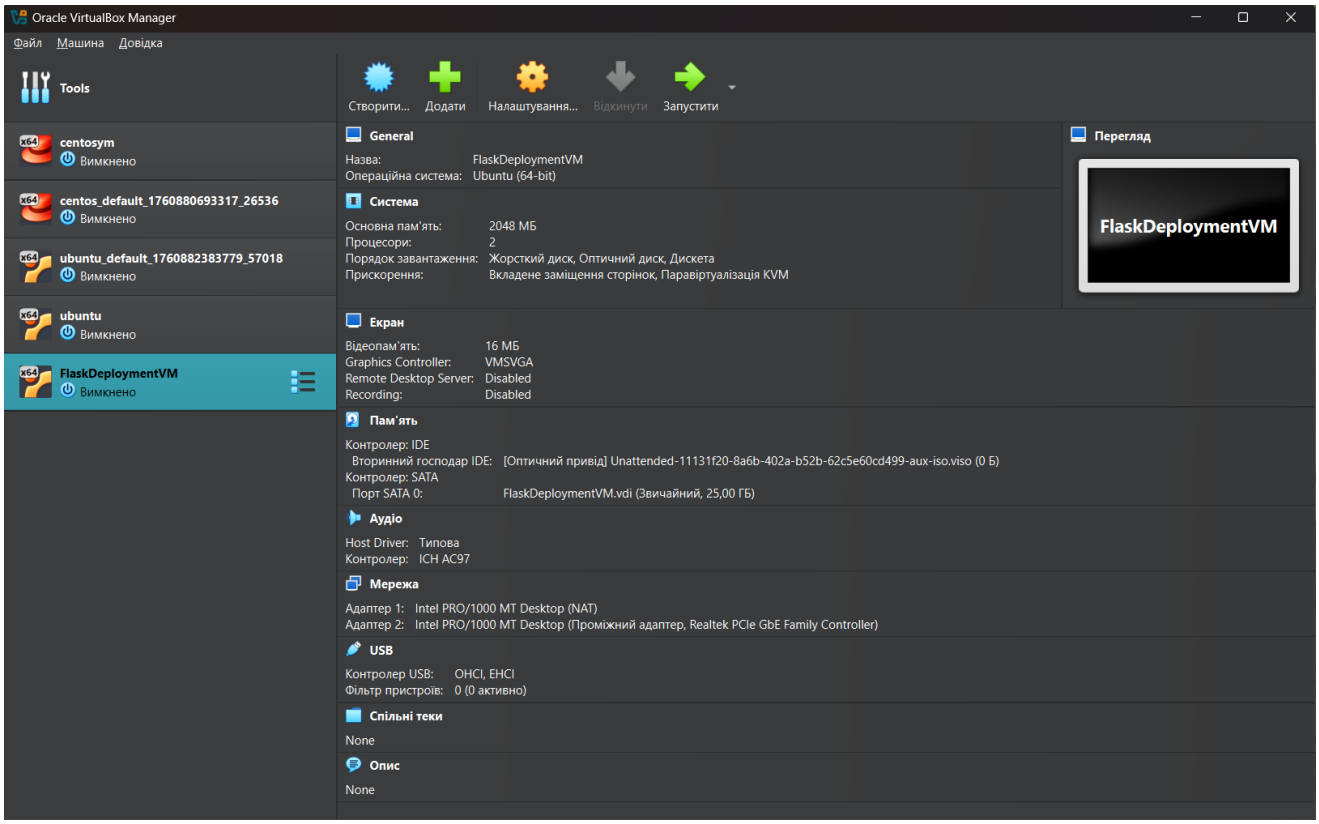

Рисунок 1.8 – Скріншот початкових етапів підготовки віртуального середовища для ручного розгортання вебдодатка

Процес ручного розгортання розпочинається з підготовки віртуального середовища, що включає створення та налаштування віртуальної машини, встановлення операційної системи та базових системних компонентів. На цьому етапі адміністратор змушений виконувати низку однотипних операцій, коректність яких безпосередньо залежить від досвіду та уважності виконавця. Навіть незначні відхилення у конфігурації можуть призвести до подальших проблем у роботі додатка або ускладнити відтворення середовища на іншій платформі.

Наступним кроком є встановлення програмного забезпечення, необхідного для роботи вебдодатка, зокрема серверних компонентів, інтерпретатора мови програмування та залежних бібліотек. Цей етап характеризується необхідністю послідовного виконання значної кількості команд, причому результат залежить від

сумісності версій програмного забезпечення та коректності їхнього налаштування. Відсутність єдиного формалізованого сценарію призводить до того, що повторення цього процесу на іншій віртуальній машині вимагає фактичного виконання всіх дій знову.

Процес встановлення та початкового налаштування програмних компонентів у ручному режимі представлено на рисунку 1.9.

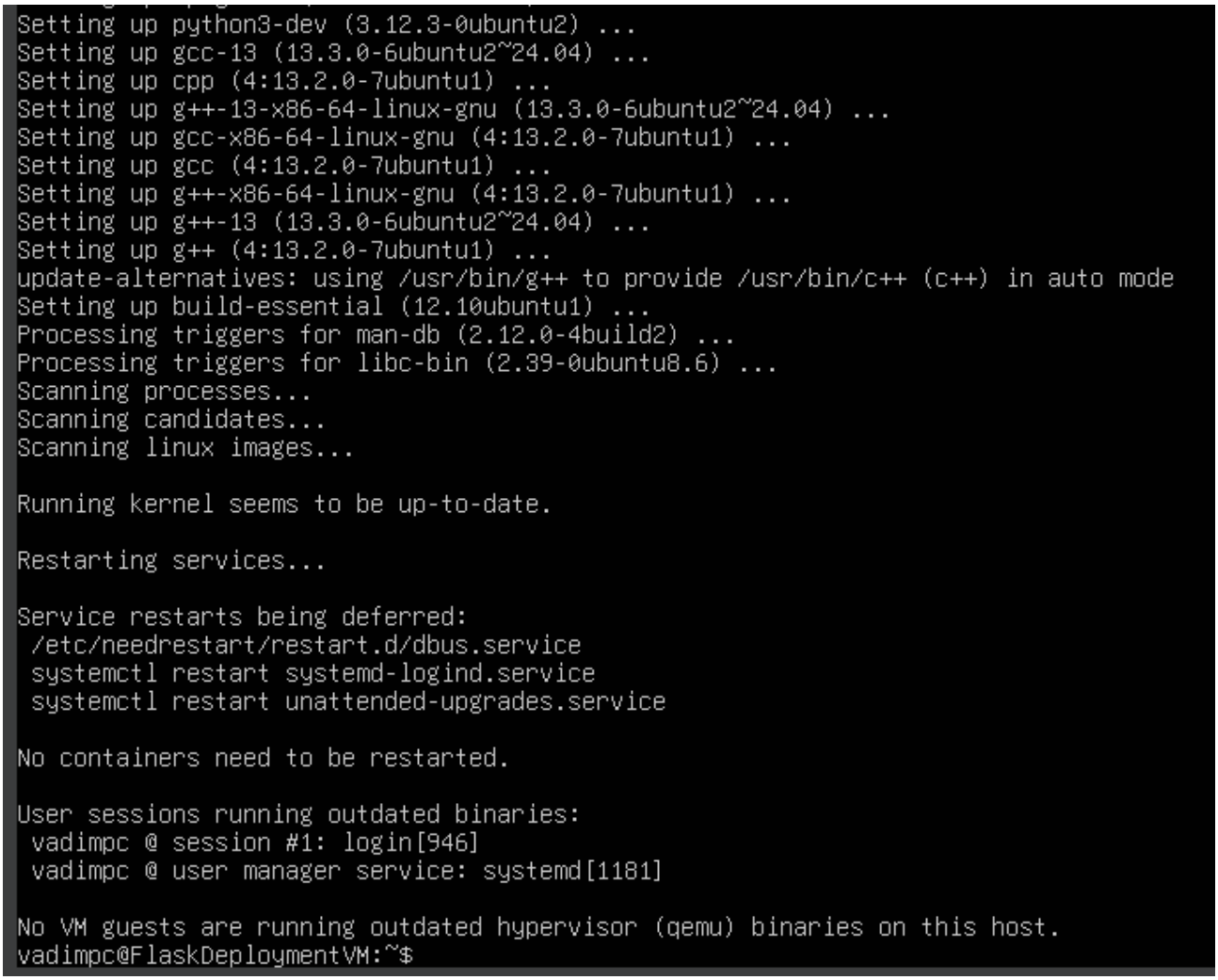

Рисунок 1.9 – Скріншот встановлення та налаштування програмних компонентів

Додатковою особливістю ручного розгортання є наявність прихованих залежностей між окремими етапами налаштування, які не завжди очевидні на перших кроках процесу. Зокрема, коректний запуск вебдодатка залежить від попередньої активації віртуального середовища та наявності коректно сформованих змінних середовища. У разі відсутності або некоректного

налаштування конфігураційного файлу із параметрами середовища виконання додаток не може бути запущений без додаткового ручного втручання. Подібні ситуації не супроводжуються автоматичними попередженнями на етапі підготовки середовища, а виявляються лише під час спроби запуску, що збільшує час розгортання та ускладнює діагностику причин помилки.

Приклад ручної конфігурації параметрів середовища виконання вебдодатка наведено на рисунку 1.10.

Рисунок 1.10 – Скріншот ручної конфігурації середовища виконання вебдодатка

Завершальним етапом є запуск вебдодатка та перевірка його працездатності. На практиці цей етап часто супроводжується необхідністю виправлення помилок, що виникли на попередніх кроках, та повторного виконання окремих операцій. Таким чином, час розгортання може істотно змінюватися залежно від кількості коригувань, що негативно впливає на передбачуваність процесу.

Таким чином, практичний аналіз процесу ручного розгортання показує, що навіть для відносно простого вебдодатка необхідно виконати значну кількість взаємопов’язаних дій, коректність яких залежить від досвіду та уважності адміністратора. Відсутність формалізованого сценарію та механізмів перевірки конфігурації призводить до зростання часу розгортання, складності відтворення середовища та підвищення ризику помилок. Завершальний етап ручного розгортання, що включає запуск вебдодатка та перевірку його працездатності, показано на рисунку 1.11.

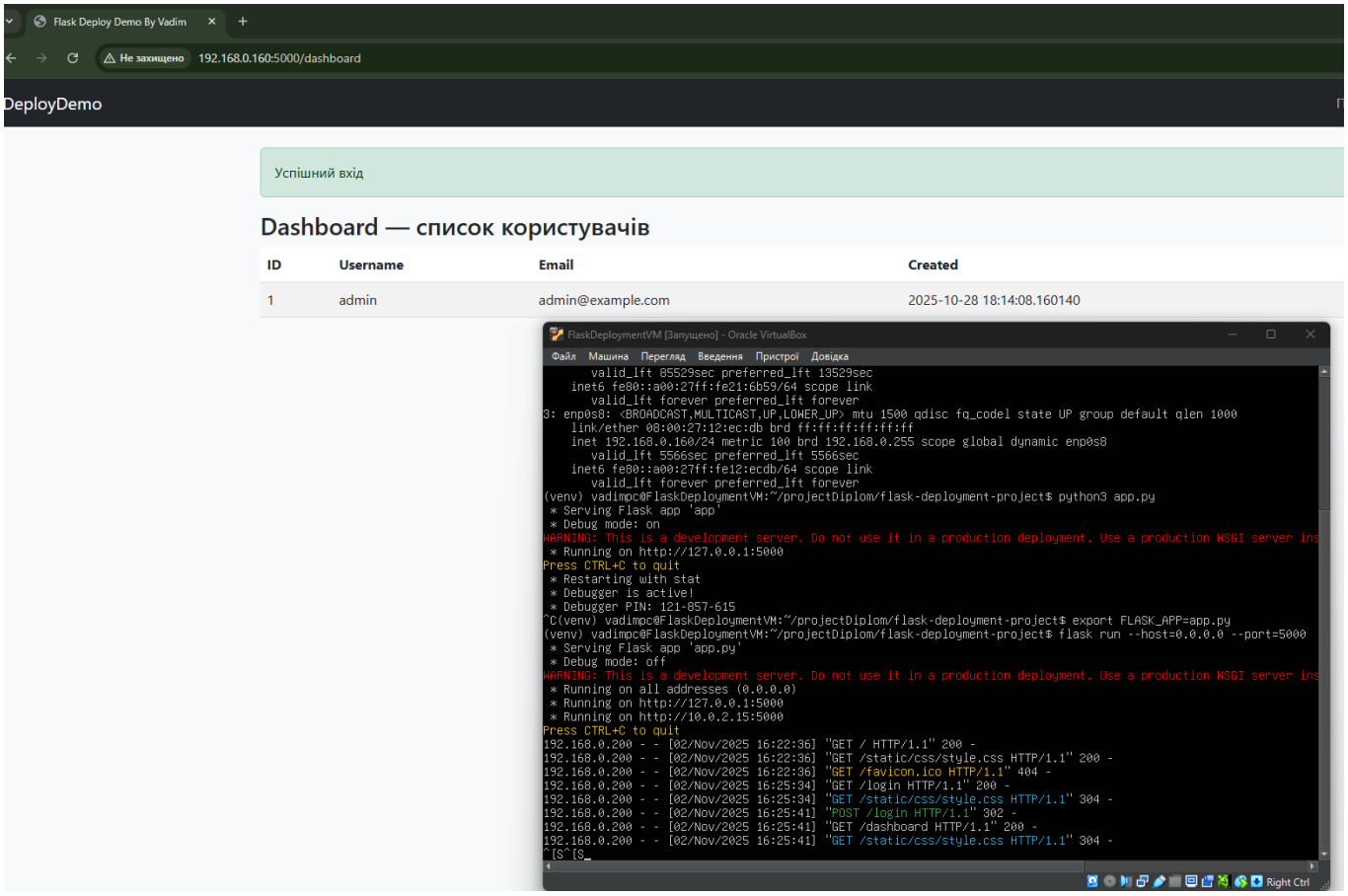

Рисунок 1.11 – Скірншот запущеного вебдодатка та перевірка його працездатності

Це підтверджує, що в умовах сучасних інформаційних систем ручний підхід не забезпечує необхідного рівня передбачуваності та стабільності. Результати практичного аналізу процесу ручного розгортання дозволяють узагальнити ключові проблеми традиційного підходу та сформулювати об’єктивні мотиви впровадження автоматизованих методів розгортання програмних систем [9].

1.5 Проблеми традиційного розгортання та мотиви впровадження автоматизації

Незважаючи на тривалу історію використання, ручне (традиційне) розгортання вебдодатків у сучасних умовах демонструє системну неспроможність відповідати вимогам динамічного ринку. Зростання складності програмних архітектур (перехід до мікросервісів), підвищення частоти релізів та суворі вимоги до аптайму (часу безперервної роботи) роблять традиційний підхід «вузьким місцем» у життєвому циклі програмного забезпечення. Проблематика ручного розгортання виходить за межі суто технічних помилок і створює організаційний конфлікт, відомий у теорії управління IT як «Стіна розбіжностей» (*Wall of Confusion*). Цей феномен ілюструє розрив між командами розробки (*Dev*) і експлуатації (*Ops*), де відсутність автоматизації призводить до хаосу в комунікації та відповідальності [5] [9].

Суть цього конфлікту полягає у протилежності цілей: розробники прагнуть якомога швидше доставити нові функції користувачеві, що вимагає частих змін у системі, тоді як експлуатаційна команда (*Operations*) відповідає за стабільність, яку найпростіше забезпечити, мінімізуючи будь-які втручання. Ручне розгортання загострює це протиріччя: оскільки процес є складним і ризикованим, адміністратори чинять опір частим оновленням, що сповільнює інноваційний цикл і збільшує обсяг змін у кожному окремому релізі (*batch size*), парадоксально підвищуючи ймовірність збою. За даними звіту *State of DevOps* 2023 від *Google Cloud*, організації з низьким рівнем автоматизації мають на 24% вищий ризик збоїв під час розгортання [28].

Критичним недоліком, що мотивує перехід до автоматизації, є феномен «племінних знань» (*Tribal Knowledge*). У ручних процесах інформація про нюанси налаштування системи часто не документується, а зберігається виключно в пам'яті окремих інженерів. Це створює високий «фактор автобуса» (*Bus Factor*) — ризик зупинки процесів у разі відсутності ключового спеціаліста (наприклад, через хворобу чи звільнення). Дослідження *Gartner* показує, що 70% IT-інцидентів у

компаніях з ручними процесами пов'язані саме з втратою таких "незафіксованих" знань. Автоматизація вирішує цю проблему шляхом перетворення знань на код (*Codification of Knowledge*): скрипти та конфігураційні файли стають живою документацією, доступною всій команді . Наприклад, інструменти на кшталт *Ansible* дозволяють описати весь процес розгортання в *YAML*-файлах, роблячи його незалежним від індивідуальних навичок [30].

Концептуальну модель «стіни розбіжностей» між командами розробки та експлуатації при традиційному підході до розгортання наведено на рисунку 1.12.

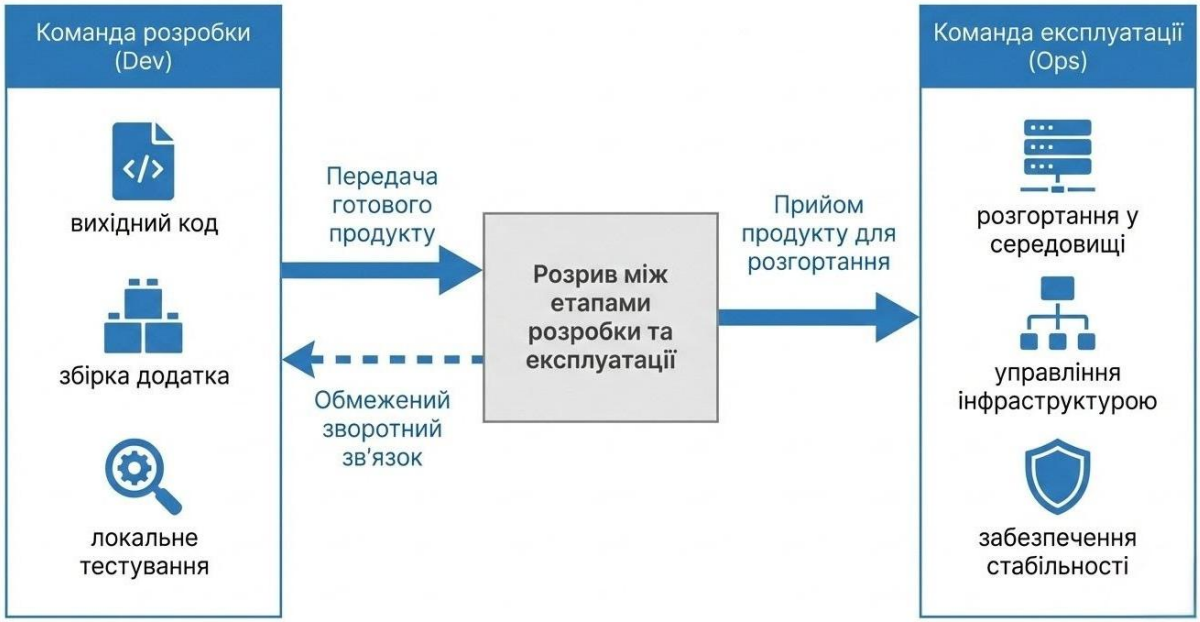

Рисунок 1.12 – «Стіна розбіжностей» між розробкою та експлуатацією при традиційному підході

Економічним мотивом відмови від ручного розгортання є висока вартість «операційної рутини» (*Toil*) — одноманітної роботи, яка не створює довгострокової цінності, але масштабується лінійно зі зростанням сервісу. Ручне відновлення системи після збою значно збільшує метрику *MTTR* (*Mean Time To Recovery* — середній час відновлення). У сучасних системах вартість простою може бути критичною: за оцінками *IDC*, середня вартість однієї години *downtime* для *Fortune* 1000 компаній становить $1–5 млн . До того ж, ручне розгортання негативно

впливає на *MTBF* (*Mean Time Between Failures* — середній час між збоями), оскільки великі пакети змін (великий *batch size*) підвищують складність тестування та ймовірність помилок. Автоматизовані системи, навпаки, дозволяють впроваджувати малі, часті оновлення, що зменшує ризик і прискорює відгук на проблеми.

Окрім того, ручне управління унеможливлює дотримання сучасних стандартів аудиту та безпеки. У регульованих галузях (фінанси, медицина) необхідно точно знати, хто, коли і яку зміну вніс у систему. Ручний доступ до серверів робить відстеження змін (*audit trail*) фрагментарним і ненадійним, що може призвести до порушень норм, таких як *GDPR* чи *HIPAA*.

Одним із ключових ефектів впровадження автоматизації у процеси розгортання є зменшення часу реакції між етапами розробки, тестування та експлуатації за рахунок скорочення петель зворотного зв’язку, що проілюстровано на рисунку 1.13.

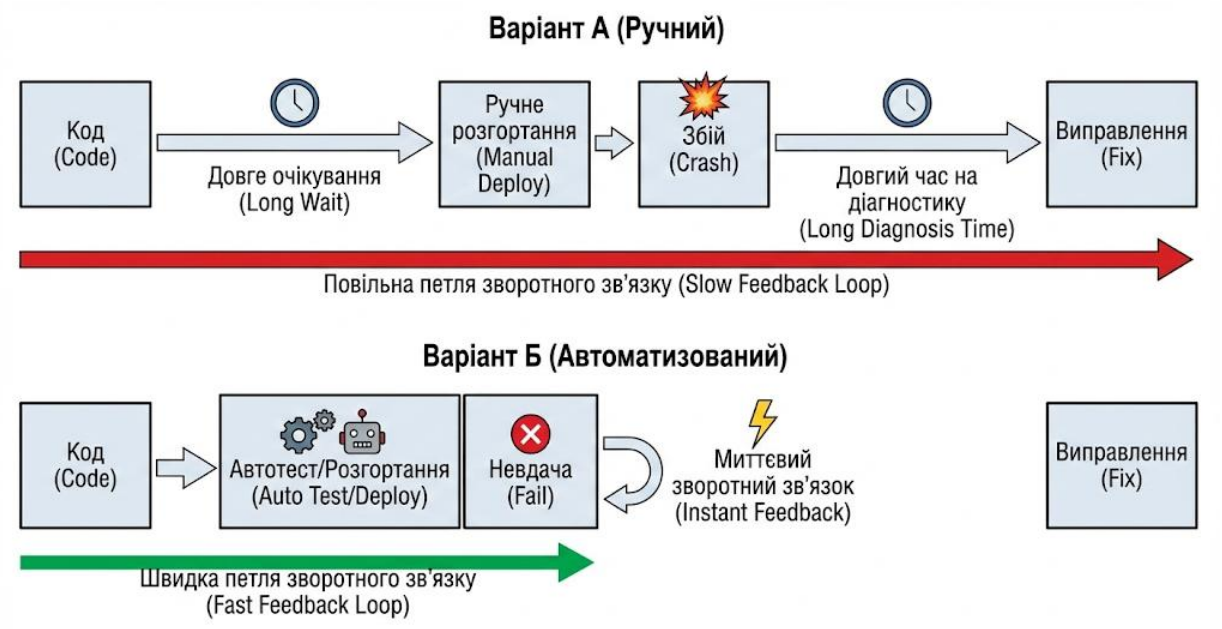

Рисунок 1.13 – Скорочення петель зворотного зв'язку при впровадженні автоматизації

За статистикою *Verizon DBIR* 2023, 82% витоків даних пов'язані з людським фактором, включаючи помилки в ручних конфігураціях . Автоматизовані конвеєри,

де всі зміни проходять через систему контролю версій (наприклад, *Git*), забезпечують повну прозорість та історичність дій за замовчуванням, а також інтегрують інструменти безпеки, як *SAST* (*Static Application Security Testing*) чи *DAST* (*Dynamic Application Security Testing*) [27].

Ще однією проблемою традиційного підходу є відсутність відтворюваності середовищ (*Environment Parity*). Ручне налаштування серверів часто призводить до "дрейфу конфігурацій" (*Configuration Drift*), коли тестове, *staging* та *production* середовища відрізняються, викликаючи помилки типу "працює в тесті, але не в проді". Це ускладнює діагностику та збільшує час на фікс помилок. Автоматизація, особливо з використанням *IaC* (*Infrastructure as Code*), забезпечує ідентичність середовищ, дозволяючи відтворювати *production* в локальному або тестовому оточенні одним скриптом.

Таким чином, мотиви впровадження автоматизації (зокрема, технологій контейнеризації та *CI/CD*) базуються на необхідності вирішення трилеми: швидкості, стабільності та безпеки. Автоматизація дозволяє розірвати пряму залежність між кількістю розгортань та кількістю збоїв, перетворюючи реліз із ризикованої події на рутинну, передбачувану операцію. Це створює фундамент для переходу до практик *DevOps*, які будуть детально проаналізовані та експериментально перевірені у наступних розділах роботи. Зрештою, впровадження автоматизації не лише оптимізує процеси, але й сприяє культурним змінам в організації, підвищуючи мотивацію команд та загальну ефективність бізнесу. За даними *Puppet State of DevOps Report* 2021, «елітні» команди з високим рівнем автоматизації випускають код у 208 разів частіше та відновлюються від збоїв у 2604 рази швидше, ніж команди з низьким рівнем [9].

1.6 Огляд методів автоматизації розгортання програмних систем

Еволюція методів розгортання програмних систем відбувалася як реакція на зростання складності програмних архітектур, перехід до мікросервісних моделей та посилення вимог бізнесу до швидкого й безпечного виведення функціональності

на ринок. У міру збільшення масштабу інфраструктур традиційні ручні або скриптові підходи перестали відповідати потребам щодо стабільності, повторюваності та швидкості розгортання. Це сприяло формуванню цілого спектра методів автоматизації, які охоплюють конфігурацію систем, підготовку інфраструктури, управління середовищами виконання та забезпечення безперервної доставки програмного забезпечення [6].

Одним із перших напрямів розвитку автоматизації стали системи конфігураційного управління, які дозволили формалізувати процес налаштування середовищ. Їх сутність полягає у переході від імперативної моделі, де адміністратор вручну виконує послідовність команд, до декларативної, у якій описується бажаний стан системи. Центральною властивістю таких моделей є ідемпотентність: повторне застосування конфігурації завжди приводить систему до одного й того самого стану, незалежно від її попереднього контексту. Завдяки цьому усувається проблема дрейфу конфігурацій — одного з головних чинників нестабільності великих інфраструктур, де відмінності між серверами можуть виникати навіть унаслідок незначних несинхронних дій оператора [13].

Наступним еволюційним етапом став підхід «інфраструктура як код», що передбачає опис інфраструктурних ресурсів — віртуальних машин, мережевих компонентів, сховищ, балансувальників — за допомогою формалізованих специфікацій. Завдяки цьому управління інфраструктурою набуває властивостей програмної розробки: її можна версіонувати, тестувати, повторно використовувати та перевіряти через механізми *code review*. Такий підхід сприяє переходу від традиційної змінної інфраструктури до моделі «незмінної інфраструктури», в якій зміна стану досягається не модифікацією існуючих екземплярів, а створенням нових, повністю відтворюваних конфігурацій. Це значно підвищує стабільність та контрольованість середовищ, особливо у великих хмарних або віртуалізованих інфраструктурах.

Для систематизації підходів до автоматизації процесів розгортання та узагальнення рівнів зрілості інфраструктурних рішень доцільно розглянути ієрархію рівнів автоматизації, наведену на рисунку 1.14.

Рисунок 1.14 – Ієрархія рівнів автоматизації інфраструктури

Істотним етапом еволюції стало поширення контейнеризації — підходу, що забезпечує ізоляцію застосунків разом із їхніми залежностями на рівні користувацького простору. Завдяки використанню механізмів операційної системи, таких як контролюючі групи ресурсів та простори імен, контейнери забезпечують легковагові та ізольовані середовища, які характеризуються високою щільністю розміщення та мінімальними накладними витратами. Контейнеризація створила можливість гарантувати бінарну ідентичність середовищ розробки, тестування та продуктивної експлуатації, що зменшує кількість помилок, спричинених відмінностями конфігурацій.

У міру зростання масштабів інфраструктур виникла потреба в автоматизованих системах керування контейнерними середовищами. Оркестраційні платформи забезпечують розміщення контейнерів на обчислювальних вузлах, контроль за їхнім життєвим циклом, балансування навантаження та автоматичне масштабування відповідно до навантаження. Однією з ключових властивостей таких платформ є самовідновлення: система здатна виявляти відмови та автоматично відтворювати потрібний стан, перезапускаючи або переміщуючи компоненти. В основі цих систем лежить принцип декларативного управління, коли користувач описує бажаний стан, а інфраструктура самостійно його підтримує. Подальшим розвитком автоматизації

інфраструктури є використання систем оркестрації, які забезпечують керування життєвим циклом контейнерів, балансування навантаження та самовідновлення компонентів, що проілюстровано на рисунку 1.15.

Рисунок 1.15 – Принцип роботи оркестрації контейнерів

Наступним ключовим елементом автоматизації є безперервні процеси інтеграції та доставки програмного забезпечення. Безперервна інтеграція передбачає регулярне об’єднання змін у спільній кодовій базі та їх автоматизовану перевірку за допомогою тестів та статичного аналізу [10].

Безперервна доставка забезпечує автоматизоване транспортування артефактів через послідовність середовищ — від розробки до продуктивної експлуатації. Цей підхід підтримує високу швидкість розробки та дозволяє своєчасно виявляти помилки, що істотно скорочує ризики, пов’язані з великими релізами.

У практиці безперервної доставки використовуються різні стратегії розгортання. Наприклад, моделі паралельного розгортання дозволяють оновлювати систему без переривання доступності, тоді як поетапні підходи забезпечують можливість поступового розгортання нової версії для частини користувачів із можливістю швидкого відкату у разі виявлення проблем. Такі стратегії не лише підвищують надійність, а й дають змогу контролювати поведінку системи у реальних умовах [18].

У межах безперервної доставки застосовуються різні стратегії розгортання, що відрізняються рівнем доступності системи та можливостями відкату змін, які узагальнено представлено на рисунку 1.16.

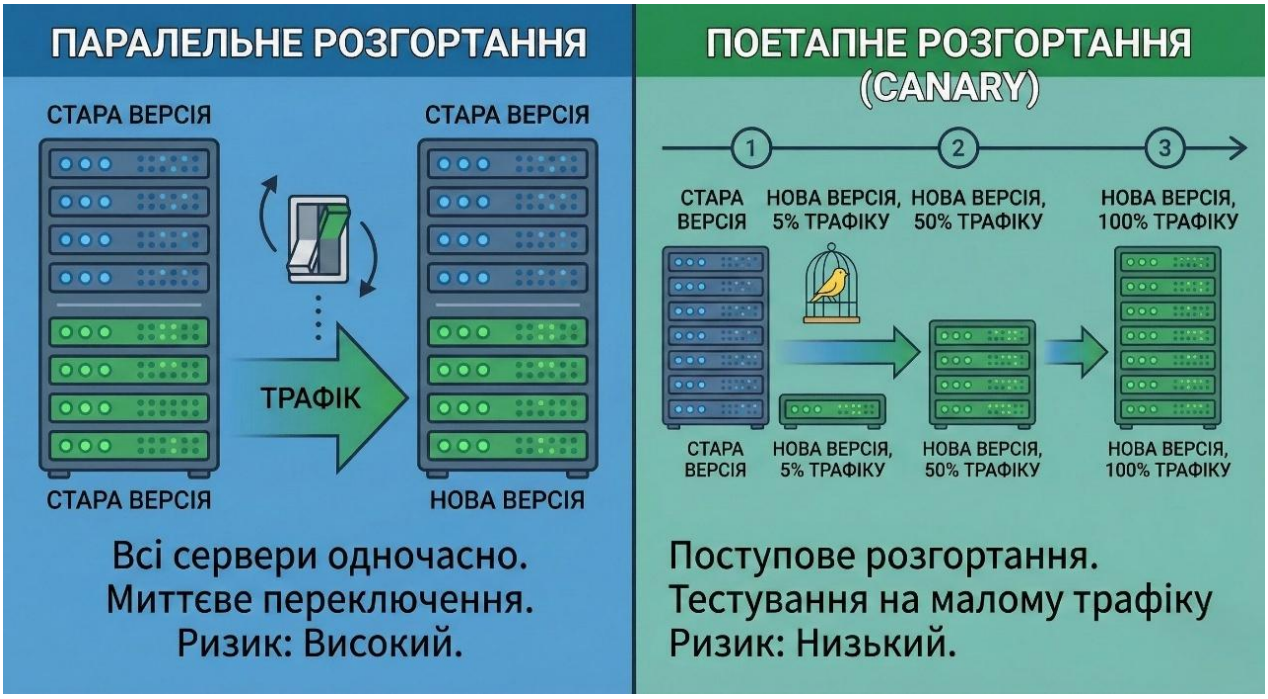

Рисунок 1.16 – Порівняння стратегій безперервного розгортання

Перспективним напрямом подальшого розвитку є безсерверні моделі обчислень, у яких інфраструктура автоматизовано створюється та масштабується провайдером. Хоча ці підходи забезпечують найвищий рівень абстракції, для значної частини веборієнтованих систем оптимальним залишається поєднання контейнеризації, декларативного управління та безперервних процесів доставки. Саме такий стек забезпечує рівновагу між гнучкістю, керованістю та стабільністю, формуючи основу для подальшого аналізу й порівняння методів автоматизованого розгортання у практичній частині роботи [14] [15].

1.7 Загальні підходи до аналізу та порівняння технічних рішень

Оцінювання та порівняння методів розгортання програмних систем є багатовимірною задачею, у якій технічні, експлуатаційні та організаційні

характеристики утворюють складний взаємозалежний простір. На відміну від задач з однозначним критерієм ефективності, аналіз інфраструктурних рішень потребує врахування численних показників, що можуть мати різну природу, масштаб та ступінь впливу. Усе це формує потребу у використанні системних підходів, здатних забезпечити об’єктивність і відтворюваність результатів.

У сфері інфраструктурних технологій особливої актуальності набуває проблема суперечливості критеріїв. Технологічне рішення, що забезпечує високу швидкість розгортання, може вимагати значних зусиль на етапі початкової конфігурації або виявитися менш стабільним у довгостроковій перспективі. Аналогічно, система, яка демонструє виняткову передбачуваність та відтворюваність, може поступатися швидкістю масштабування або потребувати більш високих ресурсних витрат. Такі конфлікти ускладнюють пряме порівняння альтернатив і роблять неможливим використання простих однокритеріальних підходів.

У цьому контексті важливого значення набуває нормування параметрів, яке дозволяє привести різнорідні характеристики до порівнюваної форми. Оскільки критерії можуть вимірюватися у різних одиницях — від секунд і мегабайтів до якісних експертних оцінок — необхідно виконати їх перетворення у єдину шкалу. Це забезпечує можливість подальших формальних операцій: агрегації, визначення вагомостей, побудови індексів ефективності. Такий підхід дає змогу усунути вплив різних масштабів і зосередитися на сутнісних відмінностях між альтернативами.

Застосування статистичних методів у цьому процесі є невід’ємною складовою об’єктивного аналізу. Показники ефективності розгортання не є сталими: вони залежать від стану віртуального середовища, рівня навантаження, поведінки ресурсів та навіть людського фактору. Тому аналіз має враховувати не лише середні значення, але й характер розподілу результатів. Дисперсія, стандартне відхилення, наявність аномальних значень та стабільність повторюваних вимірювань дозволяють оцінити передбачуваність технології та її чутливість до змін умов виконання. У практиці інфраструктурних досліджень стабільність нерідко є не менш важливою, ніж абсолютна швидкість виконання.

Окремим аспектом аналізу технічних рішень є визначення вагових коефіцієнтів, які відображають відносну значущість критеріїв у загальній оцінці. Вибір ваг є критичним, оскільки він визначає поведінку моделі та результати порівняння. У практиці дослідження складних систем застосовуються як експертні методи, що базуються на узгоджених оцінках фахівців, так і формальні процедури перевірки логічної послідовності таких оцінок. Саме на цьому етапі важливим інструментом стає метод аналізу ієрархій Сааті, який дозволяє структурувати задачу вибору у вигляді ієрархічної моделі та здійснити послідовні парні порівняння критеріїв. На основі таких порівнянь формуються ваги, що відображають пріоритетність кожного критерію, а також перевіряється узгодженість експертних суджень, що забезпечує надійність моделі.

Аналіз інфраструктурних рішень також передбачає врахування динамічного характеру програмних систем. Параметри ефективності розгортання не є статичними: на них впливають обмеження середовища, стан віртуальних ресурсів, рівень навантаження, а також людський фактор. Це робить необхідним використання методів, здатних враховувати варіації у результатах, повторюваність вимірювань та можливі флуктуації часу виконання операцій. Таким чином, аналіз має проводитися не лише за середніми значеннями, а й з урахуванням розподілу результатів, стабільності та відхилень.

Узагальнення різноманітних критеріїв часто здійснюється за допомогою інтегральних показників, що дозволяють формувати єдину числову оцінку для кожної альтернативи. Подібні моделі поєднують різнопланові характеристики у структуровану систему, де кожен критерій робить свій внесок відповідно до визначеної вагомості. Це не лише спрощує процедуру порівняння, але й забезпечує прозорість прийняття рішень, оскільки дозволяє простежити, як саме кожна характеристика вплинула на загальний результат.

Вибір технологічного рішення у сфері автоматизації розгортання завжди є компромісом між швидкістю, передбачуваністю, стабільністю, простотою налаштування та довгостроковою підтримуваністю. Тому системний підхід до аналізу, що поєднує нормування параметрів, статистичне опрацювання результатів,

визначення вагомостей критеріїв та побудову інтегральних показників, є необхідною передумовою для обґрунтованого вибору.

Для систематизації підходів до аналізу та порівняння методів розгортання використано узагальнену модель багатокритеріального оцінювання технічних рішень, наведену на рисунку 1.17.

Рисунок 1.17 – Узагальнена схема процесу багатокритеріального оцінювання технічних рішень

У наступному розділі така методологія буде формалізована через застосування методів багатокритеріального аналізу, зокрема підходу Сааті, що дозволить кількісно оцінити ефективність різних підходів до розгортання програмних систем [9].

1.8 Узагальнення теоретичних основ та постановка задачі дослідження

Узагальнюючи результати теоретичного огляду, можна відзначити, що сучасні підходи до розгортання програмних систем формуються на перетині кількох технологічних і методологічних напрямів. Ручне розгортання, яке історично було первинною практикою адміністрування, продемонструвало низьку передбачуваність, значну трудомісткість та високу залежність від людського

фактора. Зі зростанням масштабів і складності програмних систем такі властивості стали істотним обмеженням, що унеможливлює забезпечення стабільності та відтворюваності інфраструктур.

На противагу цьому сучасні методи автоматизації ґрунтуються на формалізації процесів, декларативних моделях та використанні абстракцій, що відокремлюють логіку розгортання від конкретних фізичних або віртуальних середовищ. Концепції конфігураційного управління, інфраструктури як коду, контейнеризації та оркестрації створюють умови для побудови передбачуваних систем, здатних масштабуватися, відновлюватися після збоїв і підтримувати однаковість середовищ упродовж усього життєвого циклу застосунку. У сукупності вони формують технологічний базис, на якому можливе забезпечення якісно нового рівня автоматизації.

Разом із тим, складність інфраструктурних рішень зумовлює необхідність системного підходу до їх оцінювання. Показники ефективності різних методів розгортання можуть мати різну природу та різну чутливість до умов експлуатації. Це вимагає застосування статистичних підходів, які дозволяють досліджувати варіабельність, стабільність і характер розподілу отриманих даних, а також механізмів нормування, що приводять критерії до порівнюваної форми. На цій основі можуть бути побудовані моделі багатокритеріального аналізу, здатні поєднати різнорідні параметри в узгоджену структуру оцінювання.

Метод аналізу ієрархій, що лежить в основі подальших розрахунків, надає можливість врахувати пріоритетність критеріїв і здійснити формальний вибір між альтернативами, які мають як кількісні, так і якісні характеристики. Він дозволяє зменшити суб’єктивність оцінювання і водночас забезпечити прозорість прийняття рішення, оскільки структура моделі наочно демонструє внесок кожного критерію в підсумковий результат.

У межах даної роботи особливого значення набуває зіставлення різних підходів до розгортання програмних систем з огляду на їхню практичну ефективність. Постає задача визначити, якою мірою автоматизація зменшує трудомісткість процесу, підвищує передбачуваність результатів, скорочує час

розгортання та знижує ризик помилок. Для цього необхідно поєднати методи статистичного аналізу з багатокритеріальним моделюванням, що дозволить отримати узагальнену оцінку кожної альтернативи та обґрунтувати її придатність у конкретних умовах [20].

Таким чином, підсумовуючи теоретичні положення, можна сформулювати основу подальшого дослідження: визначити, яким чином різні методи автоматизації впливають на ключові характеристики процесу розгортання та яка модель забезпечує найкраще співвідношення швидкості, стабільності та відтворюваності. Наступний розділ буде присвячений побудові такої моделі, вибору інструментів автоматизації та формуванню критеріїв оцінювання, що дозволить перейти до експериментального дослідження й кількісного порівняння технологічних рішень.

Висновоки до розділу

У першому розділі кваліфікаційної роботи було сформовано теоретичне та аналітичне підґрунтя дослідження процесів розгортання вебдодатків у віртуальному середовищі. Розглянуто поняття віртуалізації та особливості сучасних середовищ виконання програмних систем, що визначають складність та багаторівневий характер інфраструктурних рішень у сучасних інформаційних системах.

Окрему увагу приділено аналізу ручного розгортання вебдодатків як традиційного підходу до впровадження програмних систем. Було визначено його сутність, структуру процесу та ключові обмеження, пов’язані з необхідністю виконання значної кількості взаємопов’язаних дій у ручному режимі. Практичний аналіз процесу ручного розгортання у віртуальному середовищі дозволив наочно продемонструвати залежність результату від людського фактора, складність відтворення ідентичних середовищ та наявність прихованих конфігураційних залежностей, які можуть призводити до помилок на етапі запуску та експлуатації вебдодатка.

На основі отриманих результатів узагальнено основні проблеми традиційного підходу до розгортання, зокрема низьку передбачуваність часу виконання, підвищений ризик конфігураційних помилок, складність масштабування та обмежену відтворюваність середовищ. Зазначені недоліки набувають критичного значення в умовах зростання складності програмних систем і вимог до їхньої стабільності та надійності, що обумовлює доцільність переходу до автоматизованих методів розгортання.

У розділі також розглянуто загальні підходи до автоматизації розгортання програмних систем та методи багатокритеріального аналізу технічних рішень. Показано, що оцінювання ефективності інфраструктурних підходів потребує врахування сукупності кількісних і якісних показників, їх нормування та визначення вагових коефіцієнтів, що створює передумови для використання інтегральних показників і формалізованих методів порівняння.

Таким чином, у першому розділі було обґрунтовано проблематику ручного розгортання вебдодатків та сформовано методологічну основу для подальшого дослідження автоматизованих підходів. Отримані теоретичні та аналітичні висновки слугують підґрунтям для розробки і аналізу методів автоматизації розгортання, які будуть детально розглянуті у другому розділі роботи.

РОЗДІЛ 2
МОДЕЛІ ТА МЕТОДИ АВТОМАТИЗОВАНОГО РОЗГОРТАННЯ ДОДАТКІВ

2.1 Вимоги до автоматизованого процесу розгортання

Аналіз ручного розгортання вебдодатку у віртуальному середовищі, виконаний у попередньому розділі, показав, що традиційний підхід не забезпечує необхідного рівня передбачуваності, відтворюваності та масштабованості процесу. Значна кількість взаємопов'язаних ручних дій, залежність від досвіду адміністратора та наявність прихованих конфігураційних, залежностей ускладнюють експлуатацію програмних систем і підвищують ризик помилок.

У цих умовах автоматизований процес розгортання має відповідати низці вимог, спрямованих на усунення виявлених недоліків. Насамперед автоматизація повинна забезпечувати формування послідовності дій, необхідних для підготовки середовища виконання, встановлення програмних компонентів та запуску вебдодатка. Сформований сценарій дозволяє усунути неоднозначність інтерпретації етапів розгортання та зменшити вплив людського фактора на кінцевий результат [2].

Важливою вимогою є відтворюваність процесу розгортання, яка передбачає можливість багаторазового створення ідентичного середовища виконання незалежно від платформи або виконавця. Це особливо актуально в умовах використання віртуалізованих середовищ, де програмний додаток може переноситися між різними інфраструктурними конфігураціями. Автоматизований підхід має гарантувати, що кожен етап розгортання виконується однаково, без необхідності додаткового ручного втручання.

Ще однією ключовою вимогою є передбачуваність часу виконання процесу розгортання. На відміну від ручного підходу, де тривалість окремих етапів може істотно варіюватися, автоматизований процес повинен забезпечувати стабільні часові характеристики та можливість їх оцінювання. Це створює передумови для планування оновлень, масштабування системи та інтеграції процесів розгортання у більш складні життєві цикли програмних систем.

Автоматизований процес розгортання також має включати механізми контролю коректності виконання операцій та виявлення помилок на ранніх етапах. Наявність автоматичних перевірок дозволяє своєчасно виявляти некоректні конфігурації або відсутні залежності, зменшуючи кількість помилок, що проявляються лише під час запуску або експлуатації вебдодатка.

Окремої уваги заслуговує вимога підтримуваності та масштабованості автоматизованого процесу розгортання. У реальних умовах програмні системи постійно еволюціонують, що супроводжується оновленням компонентів, змінами конфігурацій та зростанням навантаження. Автоматизований підхід має забезпечувати можливість швидкого внесення змін до процесу розгортання без необхідності повного його перегляду, а також підтримувати масштабування вебдодатка без суттєвого ускладнення операційної діяльності [5] [6].

Таким чином, вимоги до автоматизованого розгортання формують основу для подальшого вибору архітектурних підходів і моделей реалізації процесу. Їх урахування є необхідною умовою для побудови ефективної системи автоматизації, здатної усунути обмеження традиційного ручного підходу та забезпечити стабільну роботу вебдодатків у віртуальному середовищі [17].

2.2 Архітектурні підходи до автоматизації розгортання програмних систем

Виконання вимог до автоматизованого процесу розгортання, сформульованих у попередньому підрозділі, неможливе без чіткого архітектурного розмежування складових інфраструктури та самого програмного додатка. Сучасні

підходи до автоматизації розгортання ґрунтуються на принципі декомпозиції складної системи на окремі логічні рівні, кожен з яких відповідає за власну зону відповідальності та може автоматизуватися незалежно від інших. Для узагальнення архітектурного підходу до автоматизації процесу розгортання програмних систем доцільно розглянути рівневу архітектуру, що відображає декомпозицію системи за зонами відповідальності, наведену на рисунку 2.1.

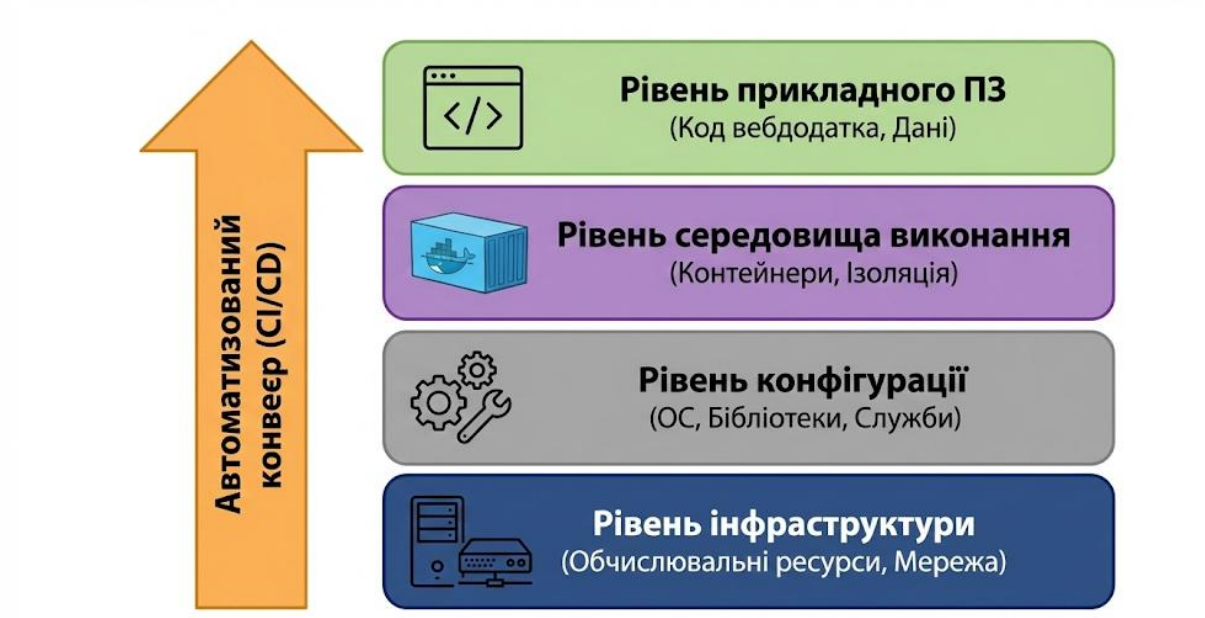

Рисунок 2.1 – Рівнева іархітектура системи автоматизованого розгортання

Узагальнена рівнева архітектура автоматизованого розгортання відображає логіку декомпозиції процесу на окремі рівні відповідальності — від інфраструктури до прикладного програмного забезпечення [17].

У загальному випадку архітектура автоматизованого розгортання розглядається як багаторівнева модель, у межах якої виокремлюють рівень інфраструктури, рівень конфігурації, рівень середовища виконання та рівень прикладного програмного забезпечення. Такий поділ дозволяє зменшити зв’язність між компонентами системи та локалізувати зміни, що виникають у процесі розвитку програмного продукту. Зміни на одному рівні не повинні призводити до необхідності повного перегляду всієї системи розгортання, що є критично важливим з точки зору підтримуваності.

Рівень інфраструктури відповідає за підготовку обчислювальних ресурсів, мережевих з’єднань та базових параметрів віртуального середовища. Архітектурно цей рівень відокремлюється від прикладної логіки та розглядається як фундамент, на якому будуються всі наступні етапи розгортання. Формалізація інфраструктурного рівня дозволяє досягти відтворюваності середовища та забезпечити однакові початкові умови для розгортання вебдодатків у різних середовищах.

Наступним є рівень конфігурації, який визначає стан операційної системи та допоміжного програмного забезпечення, необхідного для функціонування вебдодатка. На цьому рівні описуються правила встановлення залежностей, налаштування сервісів та параметрів середовища. Архітектурне відокремлення конфігурації від інших рівнів дозволяє уникнути неконтрольованого «дрейфу» налаштувань і забезпечити ідемпотентність процесу розгортання [5].

Окрему роль відіграє рівень середовища виконання, який забезпечує ізоляцію вебдодатка разом із його залежностями. Виділення цього рівня дозволяє абстрагуватися від конкретної платформи виконання та зменшити вплив відмінностей між середовищами розробки, тестування та продуктивної експлуатації. Архітектурно це створює передумови для уніфікації процесу розгортання та підвищення його передбачуваності.

Завершальним рівнем є рівень прикладного програмного забезпечення, який охоплює безпосередньо розгортання та оновлення вебдодатка. У рамках автоматизованої архітектури цей рівень не розглядається ізольовано, а інтегрується у загальний процес розгортання, що дозволяє забезпечити узгодженість змін та контроль життєвого циклу програмного продукту.

Взаємодія між окремими архітектурними рівнями автоматизованого розгортання повинна будуватися на принципах слабкої зв’язаності та чітко визначених інтерфейсів. Кожен рівень виконує власні функції та передає результат наступному рівню у формалізованому вигляді, що дозволяє зменшити каскадний вплив змін і спростити супровід системи. Такий підхід особливо важливий у

складних середовищах, де оновлення інфраструктури, конфігурацій або прикладного коду відбуваються з різною частотою та за різними сценаріями.

Архітектурна декомпозиція процесу розгортання також сприяє підвищенню прозорості та керованості системи. Чітке розмежування відповідальності між рівнями дозволяє локалізувати причини помилок і збоїв, спрощуючи їх діагностику та усунення.

У разі виникнення проблем стає можливим визначити, на якому саме етапі процесу відбулося відхилення від очікуваного стану, без необхідності аналізу всієї системи розгортання в цілому.

Важливою характеристикою архітектурних підходів до автоматизації є можливість поетапного впровадження автоматизації. У практичних умовах автоматизація рідко реалізується одномоментно для всіх рівнів інфраструктури. Натомість вона може починатися з окремих етапів, поступово охоплюючи конфігурацію середовища, управління залежностями та процеси розгортання прикладного програмного забезпечення. Архітектура, побудована з урахуванням багаторівневості, дозволяє здійснювати таке поетапне впровадження без порушення цілісності системи.

Крім того, багаторівневий архітектурний підхід створює передумови для інтеграції процесу розгортання у ширший життєвий цикл програмної системи. Формалізований та автоматизований процес може бути пов’язаний із системами контролю версій, тестування та управління змінами, що забезпечує узгодженість розгортання з іншими етапами розробки та експлуатації. Таким чином, автоматизація розгортання перестає бути ізольованою операційною задачею і стає складовою цілісної інженерної системи.

Узагальнюючи, архітектурні підходи до автоматизації розгортання програмних систем ґрунтуються на багаторівневій організації процесу, чіткому розмежуванні зон відповідальності та мінімізації зв’язаності між компонентами.

Інтеграцію автоматизованого процесу розгортання у загальний життєвий цикл програмної системи та його взаємозв’язок з етапами розробки й експлуатації ілюструє схема DevOps-пайплайна, наведена на рисунку 2.2.

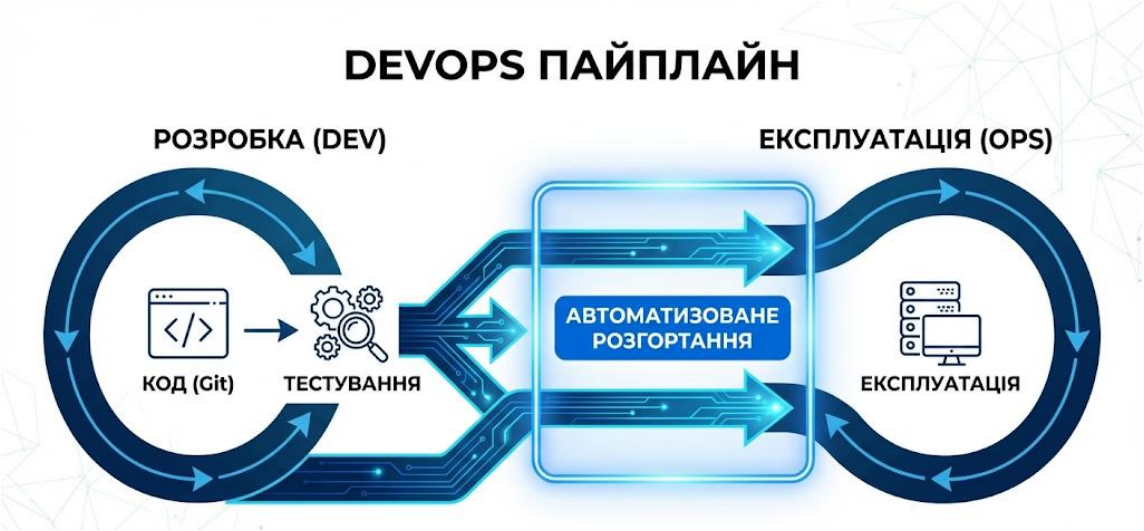

Рисунок 2.2 – Місце автоматизованого розгортання в *DevOps*-пайплайні

Така організація дозволяє формалізувати взаємодію між інфраструктурою, середовищем виконання та прикладним програмним забезпеченням, забезпечуючи керованість, відтворюваність і масштабованість процесу розгортання. Реалізація цих принципів потребує побудови узагальненої моделі автоматизованого розгортання, яка відображає послідовність та взаємозв’язок ключових етапів процесу.

2.3 Модель автоматизованого розгортання вебдодатка

Побудова автоматизованої моделі розгортання вебдодатка є необхідним кроком для формалізації архітектурних підходів, описаних у попередньому підрозділі. На відміну від узагальненого архітектурного опису, модель розгортання має відображати конкретну послідовність дій, взаємодію компонентів та логіку переходів між етапами процесу. Така модель виступає проміжною ланкою між абстрактними принципами автоматизації та їх практичною реалізацією у віртуальному середовищі.

Автоматизоване розгортання розглядається як керований процес, у межах якого початковий стан інфраструктури послідовно приводиться до цільового стану, необхідного для коректної роботи вебдодатка. Кожен етап цього процесу має чітко

визначений вхідний та вихідний стан, що дозволяє формалізувати логіку виконання і забезпечити детермінованість результату. Такий підхід є принципово відмінним від ручного розгортання, де значна частина дій виконується імпровізаційно та залежить від контексту конкретного середовища [6].

Важливою властивістю моделі автоматизованого розгортання є її орієнтація на повторюваність та контрольованість. Модель повинна передбачати можливість багаторазового виконання процесу без накопичення побічних ефектів, а також містити механізми перевірки коректності виконання окремих етапів. Це дозволяє не лише автоматизувати розгортання вебдодатка, але й використовувати модель як основу для аналізу, порівняння та подальшої оптимізації процесу.

Для формального опису автоматизованого процесу розгортання доцільно представити його у вигляді структурованої моделі, що відображає основні етапи, їх взаємозв’язок та послідовність виконання. Такий підхід дозволяє перейти від абстрактних архітектурних принципів до чіткого уявлення про логіку функціонування процесу розгортання як цілісної системи. Модель у цьому контексті виконує не лише описову, а й аналітичну функцію, оскільки дає змогу ідентифікувати критичні точки процесу та потенційні джерела помилок.

Структурно модель автоматизованого розгортання вебдодатка може бути подана як послідовність взаємопов’язаних етапів, кожен з яких відповідає за приведення системи з одного визначеного стану до іншого. Початковим станом є підготовлене або порожнє віртуальне середовище, тоді як кінцевим — повністю працездатний вебдодаток, готовий до експлуатації. Між цими станами розміщуються етапи ініціалізації інфраструктури, налаштування середовища виконання, підготовки залежностей та запуску прикладного програмного забезпечення.

Наведена модель відображає логіку поетапного переходу між станами системи та підкреслює залежність кожного наступного кроку від результатів попереднього. Візуальне подання процесу дозволяє наочно продемонструвати, що автоматизоване розгортання є не набором ізольованих операцій, а цілісним

конвеєром, у якому порушення одного етапу впливає на працездатність усього процесу.

Для формалізованого опису процесу автоматизованого розгортання вебдодатка доцільно використати узагальнену модель, що відображає послідовність етапів та переходи між станами системи, наведену на рисунку 2.3.

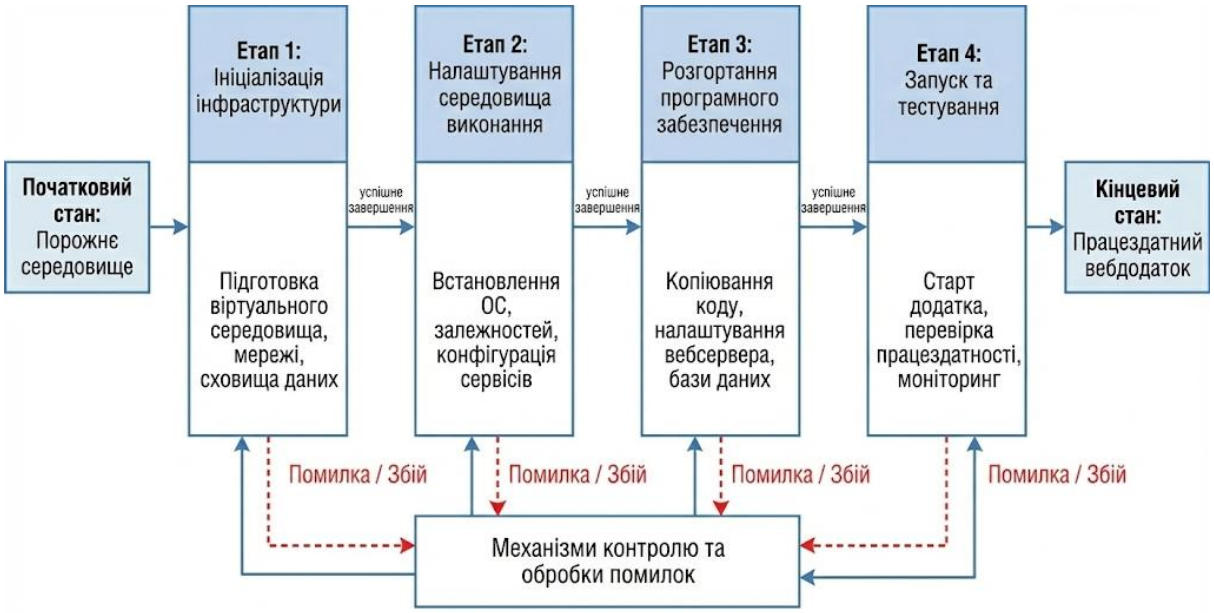

Рисунок 2.3 – Узагальнена модель автоматизованого розгортання вебдодатка

Кожен етап моделі має чітко визначені вхідні дані та очікуваний результат, що забезпечує можливість контролю коректності виконання. Наприклад, завершення етапу налаштування середовища виконання є передумовою для встановлення залежностей, а помилки на цьому етапі можуть бути виявлені ще до запуску вебдодатка. Такий підхід дозволяє локалізувати збої та зменшити витрати часу на діагностику проблем.

Важливою особливістю моделі є її універсальність щодо конкретної реалізації. Структура процесу залишається незмінною незалежно від використовуваних інструментів або платформ, що дозволяє застосовувати модель як основу для різних сценаріїв автоматизації. Це робить модель придатною не лише для реалізації конкретного проєкту, а й для порівняльного аналізу альтернативних підходів до розгортання, що буде використано у подальших розділах роботи.

Функціонування моделі автоматизованого розгортання ґрунтується на послідовному виконанні етапів, кожен з яких переводить систему з одного визначеного стану до іншого. Початковою фазою процесу є підготовка базового середовища, у межах якої формується мінімально необхідна інфраструктурна основа для подальших дій. На цьому етапі визначаються параметри віртуального середовища, мережеві налаштування та початкові умови виконання, що створює фундамент для стабільної роботи наступних компонентів процесу розгортання.

Після формування базового середовища виконується етап конфігурації, який полягає у приведенні операційної системи та допоміжних сервісів до заданого цільового стану. Саме на цьому етапі усувається більшість потенційних джерел нестабільності, пов’язаних із відмінностями середовищ або неконсистентністю налаштувань. Формалізація конфігураційних дій дозволяє забезпечити однаковий результат незалежно від кількості запусків процесу, що є ключовою умовою для досягнення ідемпотентності розгортання.

Наступним кроком моделі є підготовка середовища виконання вебдодатка, у межах якої забезпечується ізоляція прикладного коду та його залежностей. Цей етап зменшує вплив зовнішніх факторів, таких як версії бібліотек або специфіка платформи, та сприяє уніфікації процесу запуску додатка у різних середовищах. Наявність чітко визначеного середовища виконання дозволяє розглядати вебдодаток як самодостатню одиницю, що значно спрощує його розгортання та супровід.

На завершальному етапі відбувається безпосередній запуск вебдодатка та перевірка його працездатності. У рамках автоматизованої моделі цей етап не обмежується лише стартом процесів, а передбачає виконання базових перевірок коректності роботи системи. Такий підхід дозволяє виявити помилки ще на етапі розгортання, до того як вебдодаток стане доступним для кінцевих користувачів, і запобігти переходу системи у некоректний або частково працездатний стан.

Важливою складовою моделі є наявність контрольних точок між етапами процесу. Кожен перехід між станами супроводжується перевіркою успішності виконання попередніх дій, що забезпечує детермінований характер розгортання. У

разі виникнення помилки процес може бути зупинений або повернутий до попереднього стабільного стану, що підвищує надійність розгортання та зменшує ризики порушення працездатності системи [9].

Узагальнюючи, запропонована модель автоматизованого розгортання описує процес як послідовність формалізованих етапів із чітко визначеними станами та умовами переходу. Така організація забезпечує передбачуваність результатів, спрощує діагностику помилок і створює основу для подальшої реалізації автоматизованого розгортання у віртуальному середовищі. Реалізація цієї моделі у практичному сценарії буде розглянута далі, де наведено опис конкретного механізму автоматизації.

2.4 Автоматизація інфраструктури та середовища розгортання за допомогою *Vagrant, Docker* і *CI/CD*

Практична реалізація автоматизованого розгортання вебдодатка потребує використання інструментів, здатних формалізувати кожен із етапів моделі та забезпечити їх узгоджену роботу. У межах даної роботи автоматизація охоплює створення інфраструктури, підготовку середовища виконання та доставку змін у рамках єдиного керованого процесу. Такий підхід дозволяє перейти від фрагментарних ручних дій до відтворюваної та контрольованої системи розгортання.

Першим кроком реалізації є автоматизація інфраструктурного рівня за допомогою підходу *Infrastructure as Code* (*IaC*), який у роботі реалізовано із застосуванням *Vagrant*. Конфігураційний файл *Vagrantfile* формалізує параметри віртуальної машини, включаючи вибір базового образу, налаштування мережевих інтерфейсів та базові характеристики середовища. Виконання однієї команди дозволяє створити стандартизовану віртуальну машину, яка відтворюється однаковим чином незалежно від хост-системи, що усуває значну частину рутинних операцій і знижує вплив людського фактора. Попри те, що автоматизація

інфраструктури забезпечує стабільні початкові умови, вона не гарантує повної ізоляції середовища виконання вебдодатка.

Реалізація підходу *Infrastructure as Code* для автоматизації створення інфраструктури у межах дослідження здійснюється за допомогою *Vagrant* [15], загальний робочий процес якого представлено на рисунку 2.4.

Рисунок 2.4 – Загальний робочий процес *Vagrant*

У межах однієї операційної системи можуть виникати конфлікти версій бібліотек або накопичення змін конфігурації, що негативно впливає на передбачуваність результату. Для усунення цієї залежності у роботі застосовано контейнеризацію, яка дозволяє відокремити середовище виконання додатка від внутрішнього стану операційної системи. Для усунення залежності вебдодатка від внутрішнього стану операційної системи застосовується контейнеризація, принцип ізоляції якої схематично зображено на рисунку 2.5.

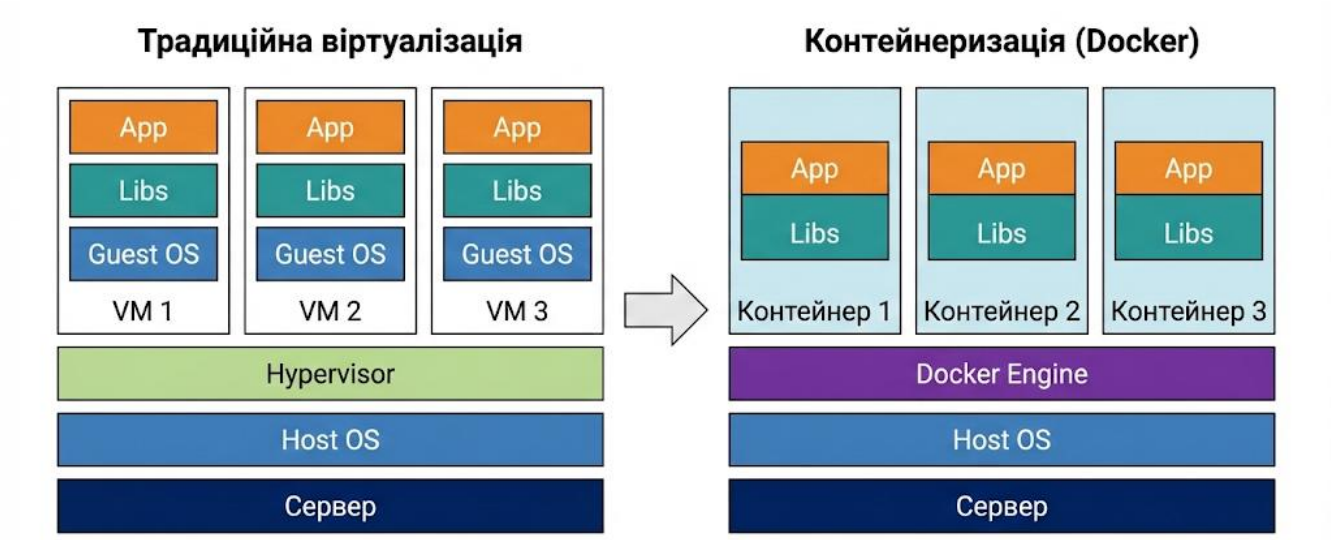

Рисунок 2.5 – Ізоляція через контейнеризацію

Контейнеризація реалізується за допомогою *Docker*, який забезпечує створення автономного середовища виконання на основі формалізованого опису. Файл *Dockerfile* визначає послідовність встановлення залежностей, копіювання програмного коду та запуск сервісів, що забезпечує однакову поведінку контейнера під час кожного розгортання. Такий підхід відповідає принципам ідемпотентності та повторюваності, закладеним у модель автоматизованого розгортання, і усуває типові проблеми, характерні для ручного та частково автоматизованого підходів [16].

Разом із цим, навіть за умови використання контейнерів, процес доставки оновлень може залишатися напівручним, якщо не забезпечено узгодженого виконання всіх етапів розгортання. Для повної автоматизації життєвого циклу вебдодатка буде застосовано підхід безперервної інтеграції та доставки (*CI*/*CD*). У межах цього підходу кожна зміна програмного коду автоматизовано проходить через визначені етапи перевірки, збирання та розгортання, що дозволяє мінімізувати участь оператора та забезпечити стабільність процесу оновлення.[20]

Узагальнена логіка реалізованого автоматизованого процесу розгортання наведена на рисунку 2.6, де показано взаємодію інструментів *Vagrant, Docker* та *CI*/*CD* [19] у межах єдиного механізму. Така інтеграція дозволяє забезпечити відповідність практичної реалізації архітектурним принципам і моделі автоматизованого розгортання, сформульованим раніше.

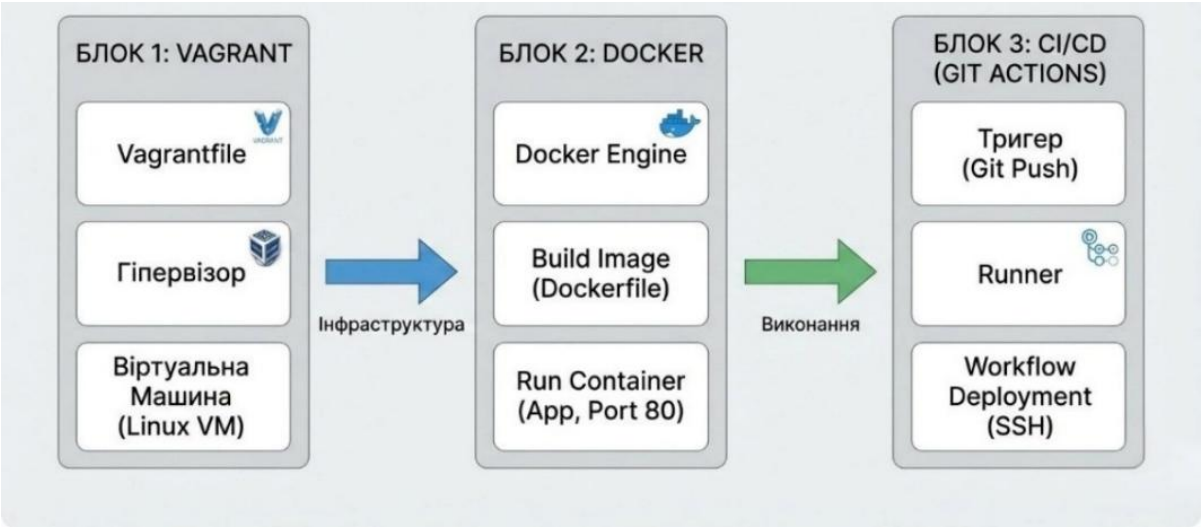

Рисунок. 2.6 – Узагальнена модель автоматизованого розгортання вебдодатка

Для наочності функціональний внесок кожного інструмента у загальний процес автоматизації подано в таблиці 2.1. У ній відображено відповідність між рівнями системи та інструментами, що використовуються для їх реалізації.

Таблиця 2.1 – Ролі інструментів у моделі автоматизованого розгортання

Рівень системи	**Інструмент**	**Функція у процесі розгортання**
Інфраструктура	*Vagrant*	Створення стандартизованої віртуальної машини
Середовище виконання	*Docker*	Ізоляція додатка та формування стабільного середовища
Життєвий цикл розробки	*CI/CD*	Автоматична доставка змін та оновлення середовища

Узгоджене застосування зазначених інструментів формує цілісне автоматизоване середовище, у якому процес розгортання набуває керованості, передбачуваності та високого ступеня повторюваності [20]. За таких умов стає можливим не лише автоматично відтворювати необхідні умови виконання вебдодатка, а й проводити об’єктивне порівняння різних підходів до розгортання за єдиною логікою. Це створює підґрунтя для подальшого аналізу ефективності застосованих методів і формалізації системи критеріїв оцінювання.

2.5 Обґрунтування критеріїв ефективності та визначення їх ваг методом аналізу ієрархій

Реалізація автоматизованого розгортання вебдодатка, описана раніше, дозволяє суттєво зменшити вплив випадкових факторів і привести процес до керованого та відтворюваного стану. Однак наявність формалізованого процесу сама по собі ще не дає відповіді на питання про його ефективність у порівнянні з альтернативними підходами. Для цього необхідно перейти від опису реалізації до

кількісного аналізу результатів, який дозволяє здійснювати обґрунтоване порівняння різних методів розгортання [39].

Особливість задачі оцінювання ефективності процесів розгортання полягає у багатовимірному характері результатів. Жоден окремий показник не може повною мірою відобразити якість процесу. Зменшення часу розгортання може супроводжуватися зниженням стабільності, високий рівень автоматизації — збільшенням початкової складності налаштування, а мінімізація ручних дій — підвищеним споживанням обчислювальних ресурсів. У таких умовах використання однокритеріальних підходів є методологічно некоректним і не дозволяє зробити об’єктивні висновки.

Для подолання цієї проблеми доцільно застосувати багатокритеріальний підхід, у межах якого ефективність кожного методу розгортання розглядається як сукупний результат впливу кількох незалежних, але взаємопов’язаних характеристик. Формально це може бути подано у вигляді інтегрального показника ефективності, який агрегує окремі критерії в одну узагальнену числову оцінку. У загальному вигляді такий показник може бути представлений наступною залежністю:

$$E = \omega_1 K_1 + \omega_2 K_2 + \cdots + \omega_n K_n \tag{2.1}$$

де K_i — окремі критерії ефективності, а ω_i — відповідні вагові коефіцієнти, що відображають відносну важливість кожного критерію у загальній оцінці.

На даному етапі формула має абстрактний характер і не містить конкретних значень ваг. Її призначення полягає у формалізації підходу до оцінювання та визначенні структури інтегрального показника. Подальшим кроком є формування переліку критеріїв ефективності та визначення їх вагових коефіцієнтів з урахуванням специфіки процесів розгортання вебдодатків у віртуальному середовищі.

Саме визначення вагових коефіцієнтів є ключовою складністю багатокритеріального аналізу, оскільки воно потребує врахування інженерної

доцільності, практичного досвіду та взаємного впливу критеріїв. Для розв’язання цієї задачі використовується метод аналізу ієрархій, який дозволяє формалізувати експертні судження та отримати кількісні оцінки ваг критеріїв.

Формування системи критеріїв ефективності є ключовим етапом багатокритеріального аналізу, оскільки саме від вибору критеріїв залежить коректність подальшого порівняння методів розгортання. Критерії мають відображати не лише технічні характеристики процесу, але й його експлуатаційні властивості, які проявляються під час регулярного використання системи у реальному середовищі. Вибір показників здійснювався з урахуванням особливостей ручного та автоматизованого розгортання, а також вимог до стабільності та керованості інфраструктури [37].

Одним із базових критеріїв ефективності є стабільність розгортання, яка характеризує здатність процесу завершуватися успішно за повторних запусків у однакових умовах. Стабільність відображає передбачуваність результату та безпосередньо пов’язана з ризиком виникнення збоїв під час оновлення системи. У контексті автоматизованого розгортання цей критерій набуває особливої ваги, оскільки навіть незначні відхилення у конфігурації або послідовності дій можуть призводити до повної непрацездатності вебдодатка.

Час розгортання є другим важливим критерієм, що визначає оперативність впровадження змін та швидкість відновлення працездатності системи у разі аварійних ситуацій. У сучасних вебсервісах швидкість доставки оновлень напряму впливає на безперервність надання послуг і рівень доступності системи. Водночас надмірне скорочення часу розгортання без належного контролю може негативно позначитися на стабільності процесу, що підкреслює необхідність розглядати цей критерій у взаємозв’язку з іншими показниками.

Рівень автоматизації процесу розгортання відображає ступінь формалізації та самостійності виконання операцій без участі людини. Високий рівень автоматизації зменшує залежність від досвіду оператора, підвищує відтворюваність результатів і створює передумови для масштабування

інфраструктури. Разом з тим, автоматизація не є самоціллю, а розглядається як інструмент підвищення загальної ефективності процесу розгортання.

Кількість ручних дій є показником трудомісткості процесу та опосередковано характеризує ймовірність виникнення помилок, зумовлених людським фактором. Чим більша кількість операцій виконується вручну, тим складніше забезпечити однаковий результат при повторних запусках і тим вищі вимоги до кваліфікації персоналу. Зменшення частки ручних дій є одним із основних мотивів впровадження автоматизованих підходів до розгортання.

Окремим критерієм ефективності є споживання обчислювальних ресурсів, яке характеризує навантаження на процесор, оперативну пам’ять та дискову підсистему під час розгортання. Хоча цей показник не завжди є визначальним, його значення зростає у середовищах із обмеженими ресурсами або при високій щільності розгортань. У таких умовах надмірне використання ресурсів може обмежувати масштабування або призводити до деградації продуктивності суміжних сервісів.

Таким чином, для подальшого аналізу було сформовано систему з п’яти критеріїв ефективності: стабільність розгортання, час розгортання, рівень автоматизації, кількість ручних дій та споживання ресурсів. Обрані критерії охоплюють як технічні, так і експлуатаційні аспекти процесу та дозволяють комплексно оцінити ефективність різних методів розгортання. Наступним кроком є визначення відносної важливості цих критеріїв, що потребує застосування формалізованого методу зважування.

Для визначення відносної важливості сформованих критеріїв ефективності було застосовано метод аналізу ієрархій (*Analytic Hierarchy Process, AHP*), запропонований Т. Сааті. Даний метод дозволяє формалізувати експертні судження щодо значущості окремих критеріїв шляхом їх попарного порівняння та подальшого обчислення вагових коефіцієнтів. Перевагою *AHP* є можливість врахування як кількісних, так і якісних характеристик, що є особливо важливим для оцінювання процесів розгортання, де не всі аспекти можуть бути безпосередньо виміряні.

Застосування методу *AHP* передбачає побудову ієрархічної структури задачі, на верхньому рівні якої знаходиться мета оцінювання — визначення інтегральної ефективності методів розгортання. Нижчий рівень ієрархії утворюють критерії ефективності, сформовані у попередній частині підрозділу. У межах цього рівня здійснюється попарне порівняння критеріїв за ступенем їх впливу на досягнення поставленої мети.

Попарні порівняння виконуються за дев'ятибальною шкалою Сааті [39], де значення 1 відповідає однаковій важливості критеріїв, а значення 9 — абсолютній перевазі одного критерію над іншим. Проміжні значення використовуються для відображення різного ступеня домінування. Така шкала дозволяє не лише визначити напрям переваги, а й кількісно оцінити її інтенсивність, що є принципово важливим для подальших розрахунків. Відповідна матриця попарних порівнянь наведена в таблиці 2.2.

Таблиця 2.2 – Матриця попарних порівнянь за методом Сааті

Критерій	**Стабільність**	**Час**	**Автоматизація**	**Ручні дії**	**Ресурси**
Стабільність	1	3	5	6	8
Час	1/3	1	4	5	7
Автоматизація	1/5	1/4	1	3	5
Ручні дії	1/6	1/5	1/3	1	4
Ресурси	1/8	1/7	1/5	1/4	1

Отримана матриця попарних порівнянь відображає практичну логіку розгортання вебдодатків у віртуальному середовищі. Так, стабільність процесу має перевагу над часом розгортання, оскільки нестабільний процес може вимагати повторних запусків і ручного втручання, що в підсумку призводить до збільшення загального часу впровадження. Перевага часу розгортання над рівнем автоматизації пояснюється тим, що навіть добре автоматизований, але повільний процес не забезпечує необхідної оперативності в умовах частих оновлень. Високі значення у порівнянні стабільності зі споживанням ресурсів відображають

інженерну доцільність пріоритету надійності над тимчасовим перевитрачанням ресурсів.

На основі матриці попарних порівнянь було виконано її нормалізацію та обчислено власний вектор пріоритетів, який і визначає вагові коефіцієнти критеріїв ефективності. У результаті нормалізації матриці попарних порівнянь та обчислення власного вектора пріоритетів було визначено вагові коефіцієнти критеріїв ефективності, які відображають їх відносну значущість у формуванні інтегральної оцінки. Отримані значення наведено в таблиці 2.3.

Таблиця 2.3 – Ваги критеріїв ефективності, визначені методом *AHP*

Критерій	**Вага**
Стабільність	0.475
Час розгортання	0.295
Рівень автоматизації	0.146
Кількість ручних дій	0.059
Споживання ресурсів	0.025

Аналіз отриманих результатів свідчить про домінування критерію стабільності, на який припадає найбільша частка загальної оцінки. Значну роль також відіграє час розгортання, що відповідає сучасним вимогам до швидкості доставки змін у вебсистемах. Рівень автоматизації має середній ступінь впливу, оскільки його ефект проявляється опосередковано — через зменшення кількості ручних дій та підвищення стабільності. Найменші ваги отримали критерії кількості ручних дій та споживання ресурсів, проте їх вплив не є нульовим і може зростати у специфічних умовах експлуатації.

Для підвищення наочності отримані вагові коефіцієнти можуть бути представлені у графічному вигляді, що дозволяє швидко оцінити співвідношення впливу окремих критеріїв на загальний результат. Для підвищення наочності

результатів визначення вагових коефіцієнтів критеріїв ефективності за методом аналізу ієрархій їх графічне подання наведено на рисунку 2.7.

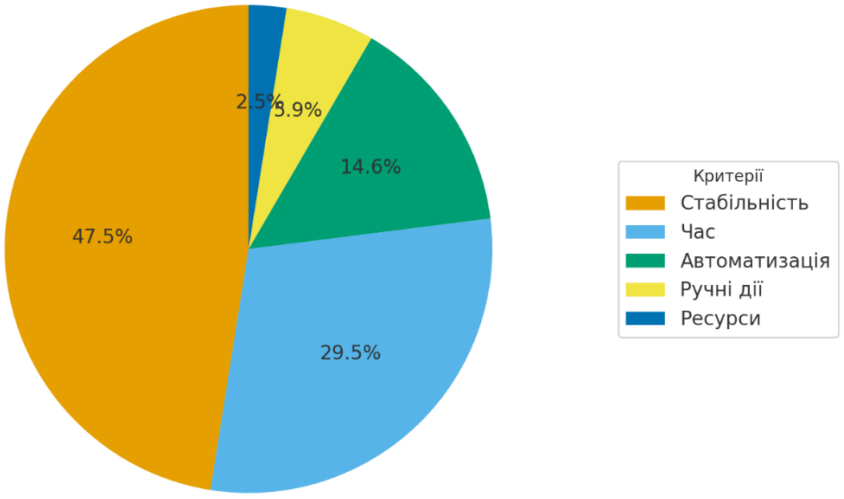

Рисунок 2.7 – Ваги критеріїв ефективності, визначені методом аналізу ієрархій

Коректність побудованої матриці попарних порівнянь підтверджується значенням показника узгодженості, яке для даної матриці становить $CR = 0.0502$, що є нижчим за граничне значення 0.1. Це означає, що експертні оцінки є логічно узгодженими, а отримані вагові коефіцієнти можуть бути використані для подальших розрахунків без втрати достовірності результатів.

Після визначення вагових коефіцієнтів критеріїв ефективності стає можливим завершити формалізацію інтегрального показника, який використовуватиметься для кількісного порівняння методів розгортання вебдодатків.

На попередньому етапі інтегральна модель була представлена у загальному вигляді без конкретних числових значень, що дозволило окреслити логічну структуру оцінювання та визначити взаємозв’язки між обраними критеріями. Такий підхід є доцільним на етапі формування моделі, оскільки дає змогу зосередитися на змісті показників і їхній ролі у загальній системі оцінювання без прив’язки до конкретних експериментальних даних.

Отримані за методом аналізу ієрархій вагові коефіцієнти дозволяють перейти від абстрактної постановки задачі до практичного інструмента кількісного аналізу. Визначені значення ваг відображають відносну важливість кожного критерію з урахуванням експертних оцінок та забезпечують можливість інтеграції різнорідних показників в єдиний узагальнений індикатор. Таким чином, інтегральна модель набуває прикладного характеру та може бути використана для об'єктивного порівняння різних підходів до розгортання за сукупністю параметрів.

З урахуванням визначених вагових коефіцієнтів інтегральний критерій ефективності набуває наступного вигляду:

$$E = 0.475S + 0.295T + 0.146A + 0.059H + 0.025R \qquad (2.2)$$

де S – стабільність, T – час розгортання, A – рівень автоматизації, H – кількість ручних дій, R – споживання ресурсів.

Отримана формула відображає внесок кожного критерію у загальну оцінку ефективності відповідно до їхньої відносної важливості. Найбільшу вагу має стабільність розгортання, що відповідає вимогам до надійності та передбачуваності інфраструктурних процесів. Значний вплив також має час розгортання, який визначає оперативність впровадження змін та відновлення працездатності системи. Інші критерії мають менші ваги, однак їхній вплив не є другорядним, оскільки вони формують загальний контекст експлуатаційної ефективності процесу.

Сформований інтегральний критерій дозволяє привести різнорідні показники до єдиної числової шкали та забезпечує можливість прямого порівняння альтернативних методів розгортання. За рахунок цього стає можливим оцінювати не окремі характеристики процесу, а його узагальнену ефективність з урахуванням взаємного впливу критеріїв. Такий підхід особливо важливий у випадках, коли методи демонструють протилежні переваги за різними показниками [38].

Використання інтегрального критерію також створює передумови для аналізу стабільності результатів та виявлення закономірностей у поведінці різних підходів до розгортання. Оскільки значення критеріїв можуть змінюватися залежно

від умов експлуатації та навантаження, застосування узагальненої оцінки дозволяє оцінювати тенденції та порівнювати методи у стандартизованому вигляді.

Таким чином, побудова інтегрального критерію ефективності завершує теоретико-методичний етап аналізу методів розгортання. Отримана модель слугує основою для експериментального дослідження, у межах якого будуть зібрані емпіричні дані, обчислені значення окремих критеріїв та визначена узагальнена ефективність кожного з розглянутих підходів. Це забезпечує логічний перехід до експериментальної частини роботи, присвяченої практичному порівнянню методів розгортання вебдодатків у віртуальному середовищі.

Висновки до розділу

У другому розділі було сформовано методологічну та архітектурну основу дослідження систем розгортання вебдодатків у віртуальному середовищі. Основну увагу зосереджено на переході від інтуїтивних та ручних процедур до формалізованого автоматизованого підходу, що відповідає сучасним вимогам до керованості, відтворюваності та масштабованості інфраструктурних рішень.

У межах розділу визначено ключові вимоги до автоматизованого процесу розгортання, зокрема формалізацію дій, відтворюваність результатів, передбачуваність часових характеристик, зменшення впливу людського фактора та підтримуваність системи в умовах її еволюції. На основі цих вимог розглянуто архітектурні підходи до автоматизації розгортання, що базуються на багаторівневому розмежуванні відповідальності між інфраструктурою, середовищем виконання та механізмами доставки змін.

Окремо було описано узагальнену модель автоматизованого розгортання, у якій поєднуються інструменти автоматизації інфраструктури, контейнеризації та безперервної доставки. Такий підхід дозволяє усунути більшість обмежень ручного розгортання та створює цілісну керовану систему, придатну для подальшого аналізу та експериментального дослідження.

Важливою особливістю розділу є те, що всі підходи до автоматизації розглядалися не абстрактно, а в контексті конкретного вебдодатка, який виступає єдиним об’єктом розгортання для всіх методів. Це дозволило забезпечити концептуальну єдність дослідження та створило передумови для коректного порівняння різних підходів за однакових умов. Саме наявність одного й того самого програмного продукту дає змогу відокремити вплив методу розгортання від особливостей прикладної логіки.

Окрему увагу у розділі приділено формуванню системи критеріїв ефективності та обґрунтуванню їх відносної важливості за допомогою методу аналізу ієрархій Сааті. Побудова багатокритеріальної моделі дозволила перейти від якісних міркувань до формалізованої оцінки, у якій стабільність, час розгортання, рівень автоматизації, кількість ручних дій та споживання ресурсів поєднуються в єдиному інтегральному показнику. Це створює методологічну основу для об’єктивного порівняння методів і усуває суб’єктивність у виборі «кращого» рішення.

Таким чином, другий розділ сформував завершену концептуальну модель автоматизованого розгортання вебдодатків та підготував інструментарій для кількісного аналізу. Запропоновані архітектурні рішення, обґрунтовані критерії ефективності та визначені вагові коефіцієнти створюють необхідні передумови для проведення експериментальних досліджень. У наступному розділі ці теоретичні положення використовуються для практичної оцінки ефективності ручного та автоматизованих методів розгортання на основі реальних вимірювань і статистичного аналізу.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА ЕФЕКТИВНОСТІ МЕТОДІВ РОЗГОРТАННЯ

3.1 Підготовка та організація експериментального середовища

Експериментальне дослідження ефективності методів розгортання виконувалося на прикладі вебдодатка власної розробки, який використовується як контрольний об’єкт для порівняння різних підходів до розгортання у віртуальному середовищі. Використання одного й того самого програмного продукту для всіх сценаріїв дозволяє виключити вплив прикладної логіки на результати та зосередитися виключно на характеристиках процесу розгортання.

Вебдодаток реалізовано на основі серверного фреймворку *Flask* [23] із використанням мови програмування *Python* та локальної бази даних *SQLite* [24]. Архітектура застосунку відповідає класичній моделі клієнт–серверної взаємодії та побудована за шаблоном *MVC*, що забезпечує чітке розмежування логіки обробки даних, представлення та взаємодії з користувачем. Такий підхід є типовим для вебзастосунків середнього рівня складності та добре підходить для експериментального аналізу процесів розгортання.

Функціональні можливості вебдодатка свідомо обмежені базовим набором операцій, достатніх для перевірки коректності запуску, роботи серверної частини та взаємодії з базою даних.

Це дозволяє уникнути впливу сторонніх сервісів і зосередити дослідження на інфраструктурних та організаційних аспектах розгортання, а не на продуктивності або функціональній насиченості застосунку.

На рисунку 3.1 наведено узагальнену схему експериментального об’єкта та середовищ його розгортання, яка ілюструє використання одного вебдодатка в різних сценаріях: у віртуальній машині з ручним налаштуванням, у середовищі з автоматизованим створенням інфраструктури, у контейнеризованому середовищі та в рамках *CI*/*CD*-конвеєра. Така схема дозволяє наочно відобразити принцип дослідження та взаємозв’язок між експериментальним об’єктом і методами його розгортання [20].

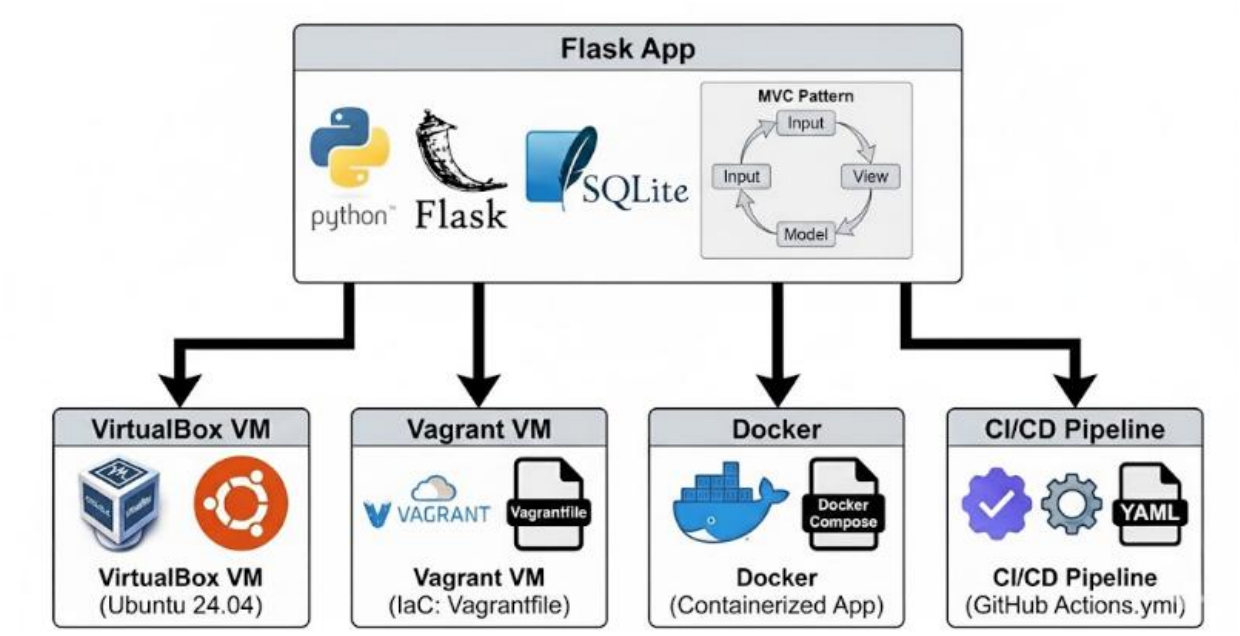

Рисунок 3.1 – Узагальнена схема взаємодії технологій, використаних у процесі розгортання вебдодатка.

Вихідний код цього проєкту було розміщено у віддаленому репозиторії *GitHub*, який став єдиним джерелом істини для всіх подальших експериментів. Саме з нього код завантажувався на віртуальну машину під час ручного розгортання, використовувався у *Vagrant*-сценаріях, включався до складу *Docker*-образу та слугував основою для *CI*/*CD*-конвеєра [16]. Така організація дала змогу усунути розбіжності між різними варіантами розгортання й гарантувала, що всі методи працюють із однією й тією самою версією програмного забезпечення. Загальний вигляд структури репозиторію з файлами конфігурації, *Dockerfile* та службовими каталогами подано на рисунку 3.2.

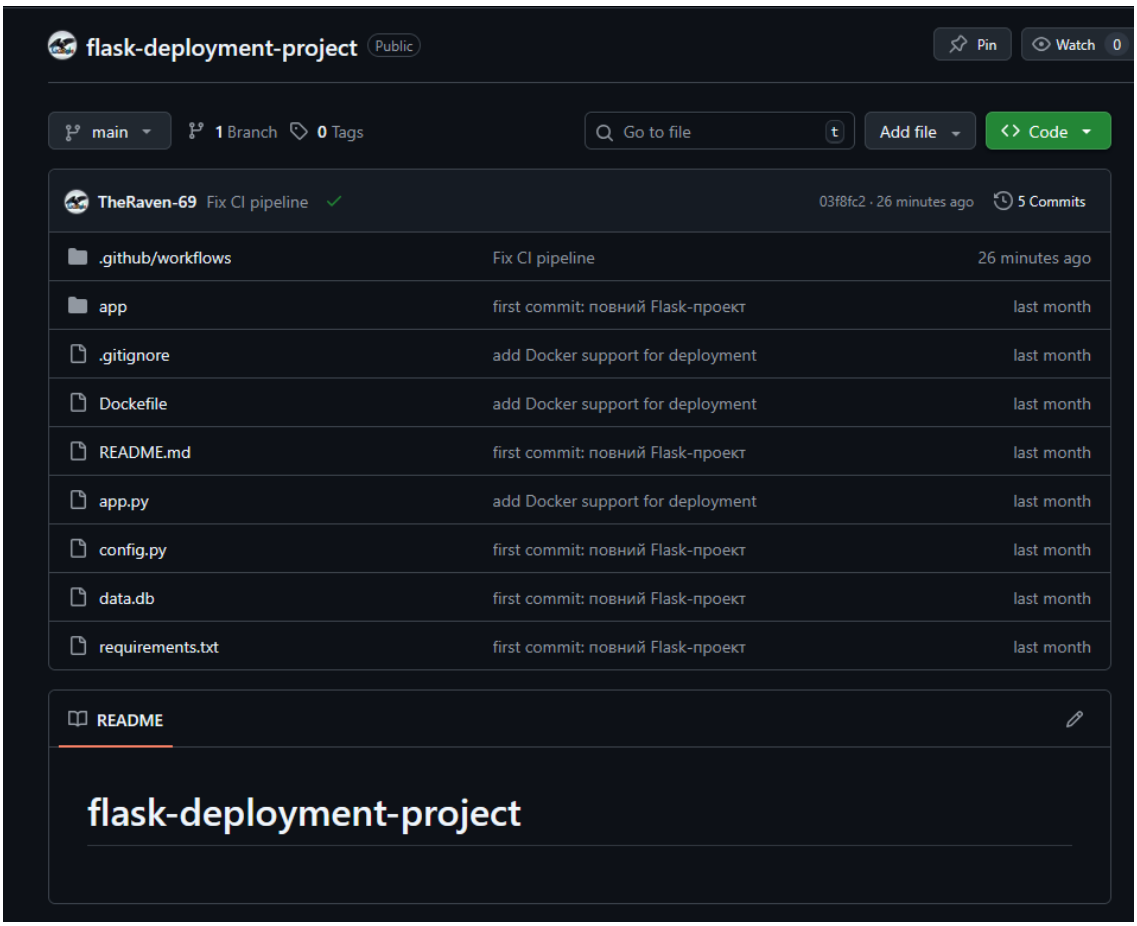

Рисунок 3.2 – Скріншот структури репозиторію дипломного проєкту на *GitHub*,

Ручний метод і підхід на основі *Vagrant* використовували класичну схему серверного розгортання у віртуальному середовищі. Спочатку на базі *VirtualBox* була підготовлена віртуальна машина з *Linux*-середовищем, до якої оператор підключався через консоль. Для ручного методу вихідний код отримували безпосередньо з *GitHub* за допомогою команди *git clone* або *git pull*, після чого виконувалася послідовність операцій із встановлення залежностей, міграції бази даних та запуску вебсервера. У випадку *Vagrant* частина цих кроків переносилася у сценарій провізіонування, однак джерелом коду залишався той самий репозиторій, а кінцевою точкою все одно була запущена у віртуальній машині інстанція *Flask*-додатка. Таким чином, обидва підходи репрезентують традиційну модель розгортання на окремому сервері, але з різним ступенем автоматизації підготовчих кроків.

Для *Docker*-методу було побудовано контейнерне середовище, яке спирається на файл *Dockerfile* у корені проєкту. У цьому файлі описано послідовність дій: вибір базового образу *python*:3.13-*slim*, копіювання файлу *requirements.txt*, встановлення залежностей, перенесення всієї структури проєкту у робочий каталог контейнера та запуск застосунку через *app.py*. Під час первинної збірки *Docker* завантажував базовий образ і всі необхідні шари, розпаковував їх та встановлював пакети з *PyPI*. Лог даного процесу наведено на рисунку 3.3 і демонструє поетапне завантаження шарів, формування файлової системи контейнера та виконання інсталяції залежностей, необхідних для коректної роботи вебдодатка. Послідовність повідомлень у консолі відображає детермінований характер процесу збірки, у межах якого кожен крок виконується відповідно до інструкцій, визначених у Dockerfile. Повна первинна збірка зайняла кілька десятків секунд, що зумовлено необхідністю одноразової підготовки базового середовища та завантаження всіх залежностей.

Варто зазначити, що повторні збірки за відсутності змін у конфігурації виконуються значно швидше завдяки механізму кешування шарів, що є однією з ключових переваг контейнеризації. Це дозволяє суттєво скоротити час повторного розгортання та підвищує передбачуваність процесу при внесенні змін до програмного коду.

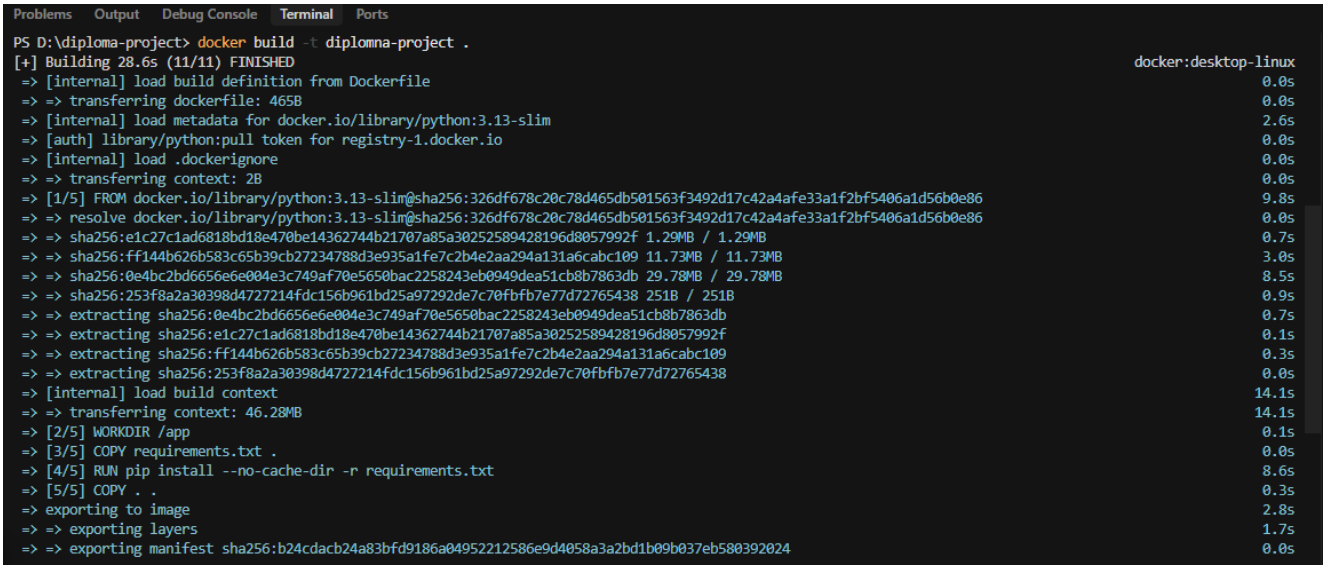

Рисунок 3.3 – Скріншот логу первинної збірки *Docker*-образу

Після формування Docker-образу контейнер було запущено у локальному середовищі з пробросом порту 5000 на хостову систему, що забезпечило доступ до вебдодатка безпосередньо з браузера. На рисунку 3.4 показано фрагмент консолі з повідомленнями Flask-сервера, який функціонує всередині контейнера та коректно обробляє вхідні HTTP-запити. Наявність стандартних лог-повідомлень свідчить про успішний запуск додатка та підтверджує працездатність сформованого контейнерного середовища.

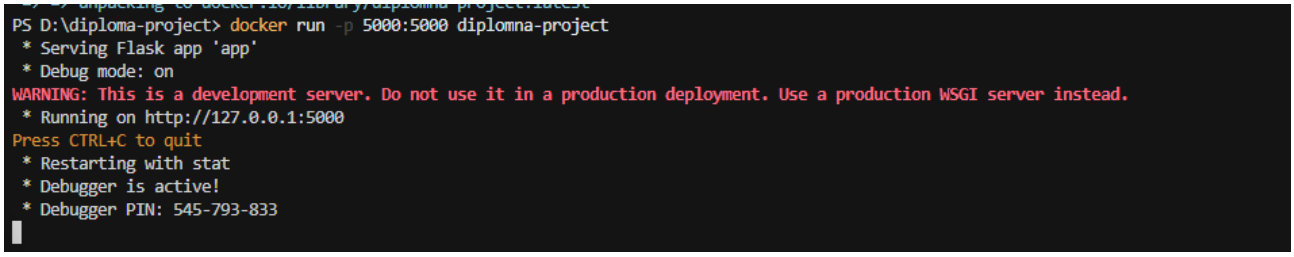

Рисунок 3.4 – Скріншот успішного запуску контейнера з вебдодатком у *Docker*

Додатково на рисунку 3.5 подано інтерфейс Docker Desktop, де відображено стан контейнера з образом дипломного проєкту, який перебуває у режимі *running*. Інтерфейс дозволяє наочно простежити активні порти контейнера, а також оцінити споживання обчислювальних ресурсів, яке у процесі роботи залишається мінімальним.

Це свідчить про ефективне використання ресурсів хостової системи та відсутність надмірних накладних витрат, характерних для повноцінних віртуальних машин.

Особливу практичну цінність контейнеризації демонструє поведінка системи при повторних збірках. За умови, що базові шари образу вже кешовані, загальний час підготовки контейнера істотно скорочується і становить лише кілька секунд.

Така особливість забезпечує високу швидкодію процесу розгортання при багаторазових деплоях і є критично важливою для сценаріїв активної розробки та тестування, де часті оновлення програмного коду є нормою.

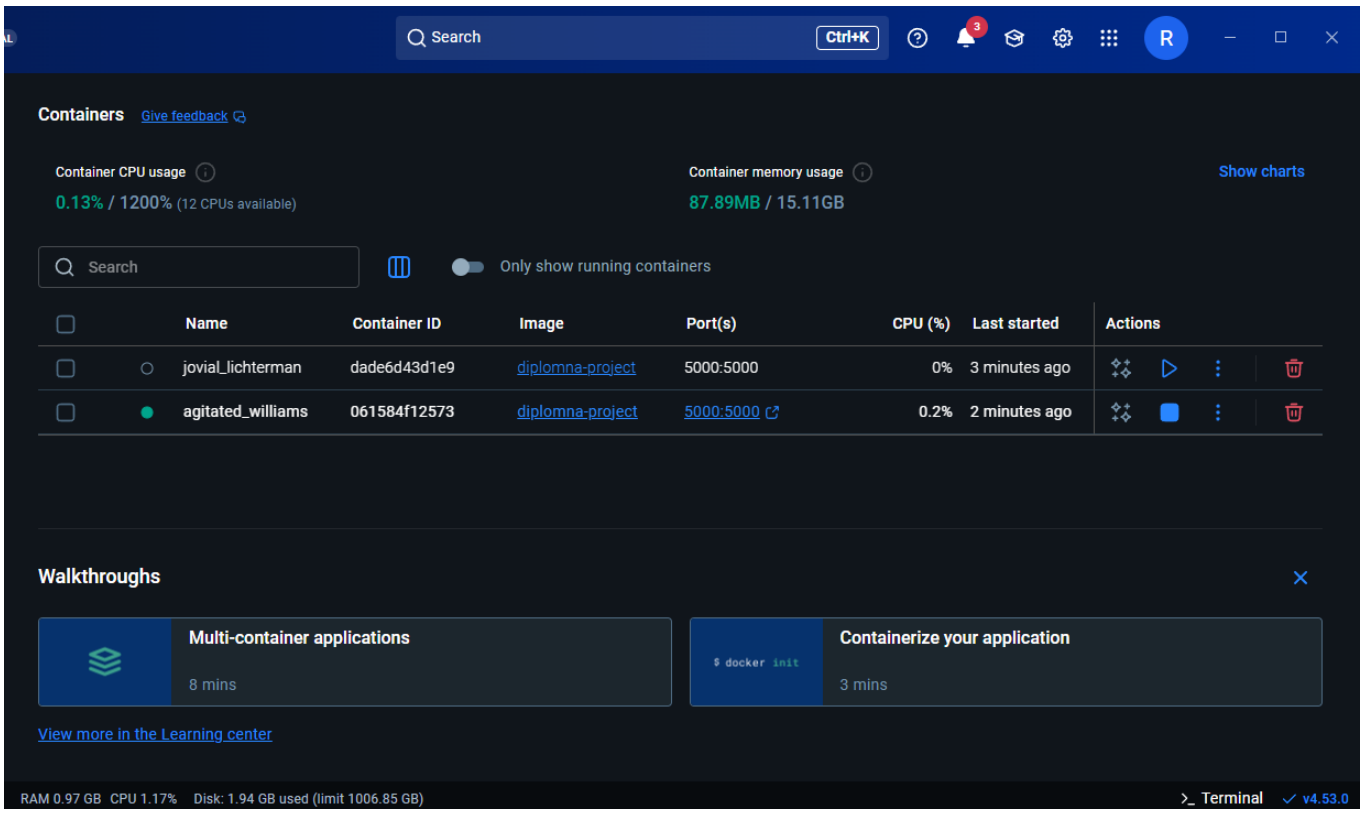

Рисунок 3.5 – Скріншот інтерфейсу *Docker Desktop* із активним контейнером

Автоматизований метод розгортання на основі CI/CD був реалізований із використанням платформи GitHub Actions. У репозиторії проєкту створено конфігураційний файл *workflow*, який формалізує послідовність дій, що виконуються автоматично при кожному оновленні гілки *main*. Такий підхід дозволяє перетворити процес розгортання з разової або напівручної операції на стандартизований і повторюваний конвеєр, керований системою контролю версій.

Після виконання коміту система ініціює запуск конвеєра, в межах якого автоматично завантажується актуальна версія вихідного коду, розгортається середовище Python необхідної версії, встановлюються залежності з файлу *requirements.txt* та виконується тестовий сценарій, що перевіряє базову працездатність застосунку. Наявність автоматичної перевірки на цьому етапі дозволяє виявляти помилки ще до фактичного розгортання та зменшує ймовірність потрапляння некоректних змін у продуктивне середовище.

На рисунку 3.6 наведено загальний вигляд сторінки *Actions* з інтерфейсу GitHub, де зафіксовано успішне виконання всіх етапів конвеєра. Відображення

статусу виконання у реальному часі забезпечує прозорість процесу розгортання та надає розробнику оперативний зворотний зв’язок щодо результатів кожного оновлення коду.

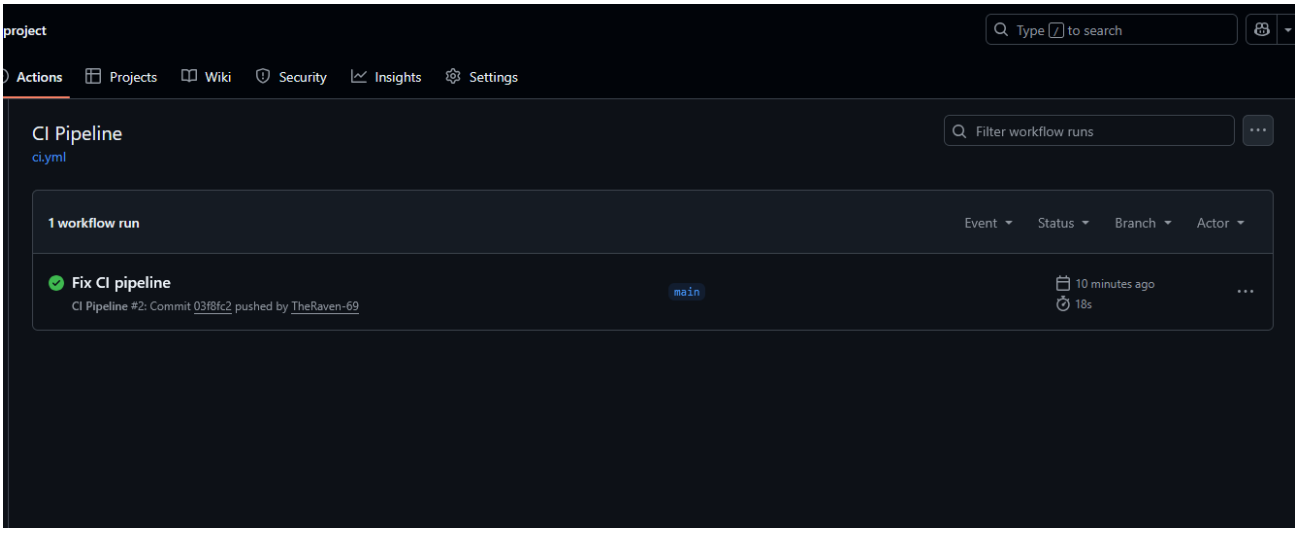

Рисунок 3.6 – Скірншот вкладки *GitHub Actions* із зафіксованим успішним запуском конвеєра

На рисунку 3.7 показано лог кроку *Run test script*, у якому міститься повідомлення *«CI/CD pipeline executed successfully»*, що підтверджує коректне завершення всього сценарію без втручання оператора.

Рисунок 3.7 – Скріншот фрагменту логу етапу «*Run test script*»

Усі чотири методи розгортання були прив’язані до одного й того самого репозиторію та функціонували в узгодженому, відтворюваному середовищі. Окремим етапом підготовки експерименту стало визначення єдиної методики, за якою здійснювався збір усіх кількісних показників. Це дозволило забезпечити відтворюваність та коректність подальшого порівняння методів розгортання. Передусім було визначено порядок фіксації часу виконання. Усі заміри проводились за однаковим принципом: відлік часу починався у момент запуску відповідної команди (*python app.py*, *vagrant up*, *docker run*, старт *workflow* у *CI/CD*) і завершувався тоді, коли вебдодаток ставав доступним за адресою *http*://127.0.0.1:5000. Для уникнення похибок кожен метод запускали десять разів, після чого з отриманих значень обчислювалось середнє та стандартне відхилення.

Оцінювання навантаження на ресурси здійснювалося за допомогою стандартних інструментів операційної системи. Для методів, заснованих на віртуальній машині (ручний підхід та *Vagrant*), навантаження фіксувалось через системний монітор *Windows* (*Resource Monitor*), який дозволяє відстежувати активність конкретних процесів у режимі реального часу.

На рисунку 3.8 наведено приклад такого вимірювання: видно поведінку процесу *VirtualBoxVM.exe*, який відповідає за роботу гостьової операційної системи, а також характерні коливання завантаження *CPU* у момент запуску серверної частини вебдодатка.

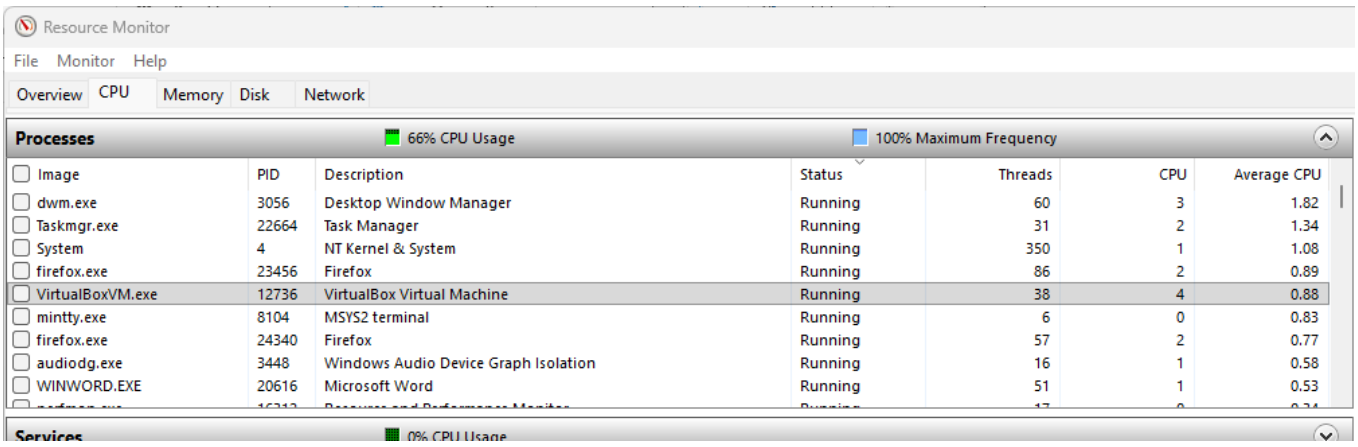

Рисунок 3.8 – Скріншот навантаження процесу *VirtualBoxVM.exe* під час роботи віртуальної машини

Окремо фіксувалась кількість ручних дій, необхідних для запуску кожного методу. Під «ручними діями» розумілись усі кроки, які потребували прямої участі оператора: відкриття терміналу, отримання вихідного коду, встановлення залежностей, виконання службових команд та контроль проміжних станів. Для *Vagrant* частина цих процедур була автоматизована сценаріями провізіонування, а *Docker* та *CI*/*CD* мінімізували або повністю усували ручну роботу, що створило суттєвий контраст між підходами.

Стабільність запуску визначалась як частка успішних спроб із десяти повторних виконань кожного методу. Запуск вважався успішним, якщо додаток коректно стартував, не завершувався з помилкою і був доступний протягом перших секунд після ініціалізації. Такий підхід дав змогу оцінити вплив людського фактору на ручне розгортання, передбачуваність *Vagrant*-сценаріїв, а також конструктивну надійність контейнеризації та *CI*/*CD*.

Завдяки такій методиці кожен метод розгортання тестувався в уніфікованих умовах, що дозволило сформувати репрезентативний експериментальний набір даних та забезпечило можливість коректного математичного аналізу в подальшому.

3.2 Збір експериментальних даних

Для отримання об’єктивної характеристики ефективності різних методів розгортання було проведено серію експериментів, у межах яких кожен підхід тестувався десять разів у однакових умовах. Такий формат дозволив зібрати не лише середні значення, а й зафіксувати коливання, які виникають у реальному робочому середовищі. Усі вимірювання виконувалися на заздалегідь підготовленій конфігурації, описаній раніше, а кожен запуск включав повний процес розгортання — від початкової ініціалізації до перевірки успішного запуску вебдодатка. Під час тестування фіксувалися ключові показники: час виконання, стабільність, кількість ручних дій та пікове споживання ресурсів, що дозволило сформувати детальне уявлення про поведінку кожного методу.

Під час вимірювань стали помітними особливі типи помилок, що виникають у практичних сценаріях. Наприклад, при ручному розгортанні кілька разів виникала типова ситуація, коли застосунок не запускався через неактивоване віртуальне середовище, у якому зберігалися змінні конфігурації, ключі доступу та параметри підключення. Також зустрічалися помилки, пов’язані з конфліктом порту під час запуску вебсервера або неправильним шляхом до конфігураційного файлу. Кожна така помилка вимагала додаткових ручних дій: повторного запуску окремих команд, відновлення залежностей або перевірки параметрів середовища. Унаслідок цього саме ручний метод продемонстрував найбільшу мінливість результатів. Типові помилки, що виникали під час ручного розгортання вебдодатка, зокрема пов’язані з некоректним налаштуванням середовища виконання, проілюстровано на рисунку 3.9.

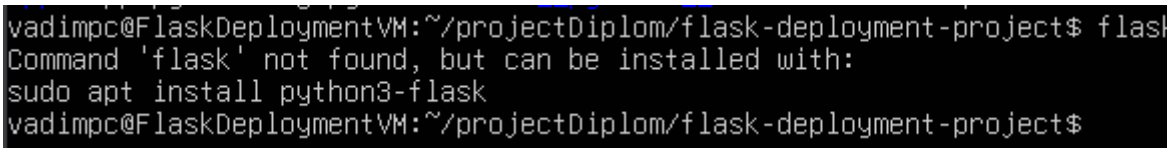

Рисунок 3.9 – Скріншот помилки через неактивоване віртуальне середовище

Сценарії з *Vagrant* виявилися більш передбачуваними, однак через необхідність запускати повноцінну віртуальну машину та виконувати *provisioning* інколи виникали затримки, пов’язані з роботою пакетного менеджера системи або некоректним ініціалізуванням мережевого адаптера у *VirtualBox*. У таких випадках розгортання потребувало повторного запуску чи відновлення окремих кроків, що також впливало на загальний час.

Робота *Docker* у більшості запусків була стабільною, однак кілька разів спостерігалося блокування порту, що вимагало зміни налаштувань або повторного запуску контейнера. У поодиноких випадках система потребувала повного очищення та перебудови образу через некоректно кешовані шари. Незважаючи на це, контейнеризація загалом демонструвала нижчий час розгортання та мінімальну кількість ручних втручань.

Найменш проблемним виявився метод автоматизованого розгортання за допомогою *CI*/*CD*. Усі дії виконувалися рутинно, а відхилення виникали лише тоді, коли мережева затримка або короткочасна недоступність *runner*'*a* вимагала повторного запуску *workflow*. Подібні події майже не впливали на тривалість, проте інколи збільшували кількість ручних дій на один додатковий крок.

Зібрані під час експерименту значення показують, як кожен метод поводився в умовах повторних запусків. Щоб було простіше простежити різницю між підходами, усі результати були зведені в одну велику таблицю — кожен рядок відповідає одному методу, а стовпчики відображають окремі спроби. Такий формат дозволяє побачити і середні значення, і випадкові коливання, які виникали у процесі. Результати експериментальних вимірювань часу розгортання вебдодатка для кожного з досліджуваних методів, отримані внаслідок багаторазового повторення експерименту, узагальнено та подано в таблиці 3.1.

Таблиця 3.1 – Час розгортання (10 запусків, с)

Метод	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Ручне	1310	1465	1390	1180	1500	1255	1420	1335	1475	1210
Vagrant	615	540	685	590	630	710	655	575	690	605
Docker	29	33	27	35	31	28	26	34	30	32
CI/CD	21	18	22	19	23	17	21	20	18	22

Окрім часу виконання, у процесі експериментального дослідження фіксувалася стабільність процесу розгортання, тобто здатність відповідного методу завершувати всі етапи без помилок і потреби у повторному запуску. Даний показник є принципово важливим, оскільки навіть незначні збої у процесі розгортання можуть призводити до додаткових часових витрат, зниження передбачуваності результатів та зростання навантаження на виконавця.

У ході тестування було зафіксовано окремі сценарії, в яких процес розгортання переривався через помилки конфігурації, нестабільність середовища або залежність від ручних дій. Наявність таких збоїв безпосередньо впливала на

підсумкову оцінку методів, оскільки вимагала повторного виконання окремих етапів або додаткового втручання з боку користувача. У таблиці 3.2 наведено результати запусків із зазначенням успішних та проблемних випадків, що дозволяє наочно оцінити надійність кожного з розглянутих підходів.

Таблиця 3.2 – Стабільність роботи (1 — успіх, 0 — збій)

Метод	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Ручне	1	1	1	0	1	1	1	1	0	1
Vagrant	1	1	1	1	1	1	0	1	1	1
Docker	1	1	1	1	1	1	1	1	1	1
CI/CD	1	1	1	1	1	1	1	1	1	1

Також ще було зафіксовано кількість ручних дій, необхідних для завершення розгортання. Саме цей показник часто демонструє “вартість” методу з точки зору навантаження на розробника. Ручні дії включали повторні команди, відкриття логів, перезапуск сервісів, оновлення пакетів та інші ручні операції, необхідні для завершення процесу. Дані для кожного з методів наведені в таблиці 3.3.

Таблиця 3.3 – Кількість ручних дій (10 запусків)

Метод	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Ручне	17	19	21	16	24	18	20	17	23	16
Vagrant	8	8	11	8	9	12	8	9	8	8
Docker	3	3	2	3	4	3	2	4	3	3
CI/CD	1	1	1	2	1	1	1	1	2	1

Останнім вимірюваним параметром було пікове споживання ресурсів системи — *CPU* та оперативної пам’яті. Цей критерій особливо важливий у випадках, коли розгортання виконується на обмежених за ресурсами системах, або

коли розглядається масштабування. У таблиці 3.4 наведені середні значення, отримані під час запусків.

Таблиця 3.4 – Пікове споживання ресурсів (*CPU*/*RAM*)

Метод	***CPU, %***	***RAM, MB***
Ручне	52	420
Vagrant	68	2200
Docker	34	310
CI/CD	29	265

Для підвищення наочності отримані результати були представлені також у графічній формі. На рисунку 3.10 наведено динаміку часу розгортання для кожного методу в межах десяти запусків. Така форма подання дає змогу побачити характер зміни тривалості та відстежити відхилення у повторюваних сценаріях.

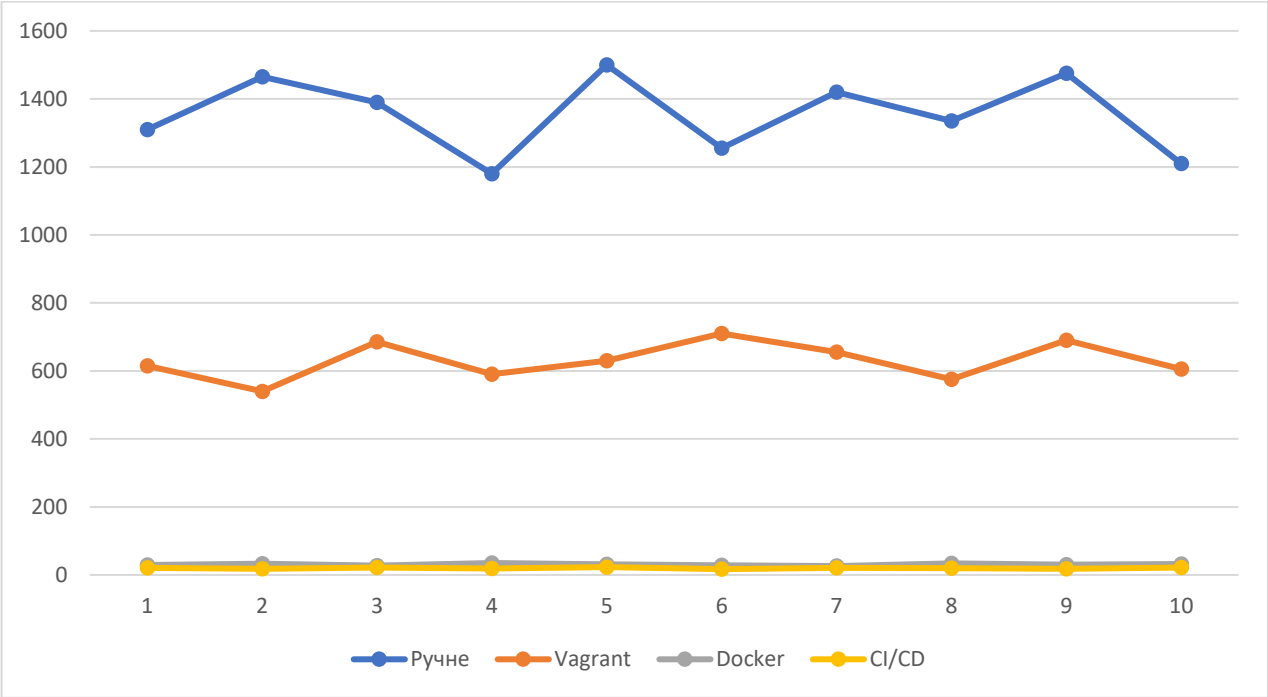

Рисунок 3.10 – Динаміка часу розгортання (10 запусків)

Потребу в ручних діях було виділено в окремий показник і подано на рисунку 3.11. Його аналіз дозволяє оцінити не лише загальну кількість додаткових операцій, а й характер взаємодії користувача з кожним із розглянутих методів розгортання. Зменшення частки ручних втручань свідчить про вищий рівень автоматизації процесу та зниження залежності результату від людського фактора.

Отримані дані дають змогу простежити, наскільки рівномірною була кількість ручних операцій між окремими запусками та як змінювалася трудомісткість взаємодії з методом у процесі розгортання. Методи з більшою кількістю ручних дій демонструють підвищену варіативність результатів і, як наслідок, нижчу передбачуваність, що негативно впливає на загальну ефективність процесу.

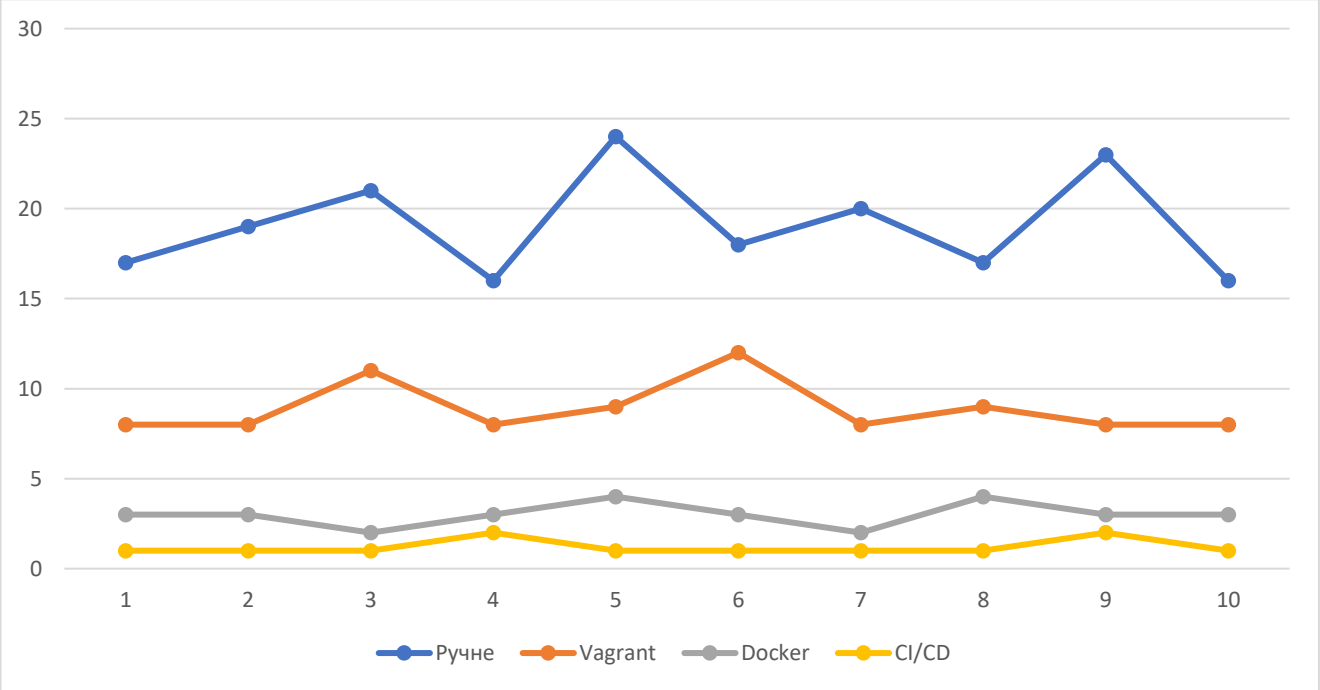

Рисунок 3.11 – Кількість ручних дій під час розгортання

Стабільність роботи, тобто кількість успішно завершених спроб, показано на рисунку 3.12. Ця візуалізація дає загальне уявлення про те, як кожен метод поводився під час повторних запусків, не заглиблюючись у деталізацію причин можливих помилок.

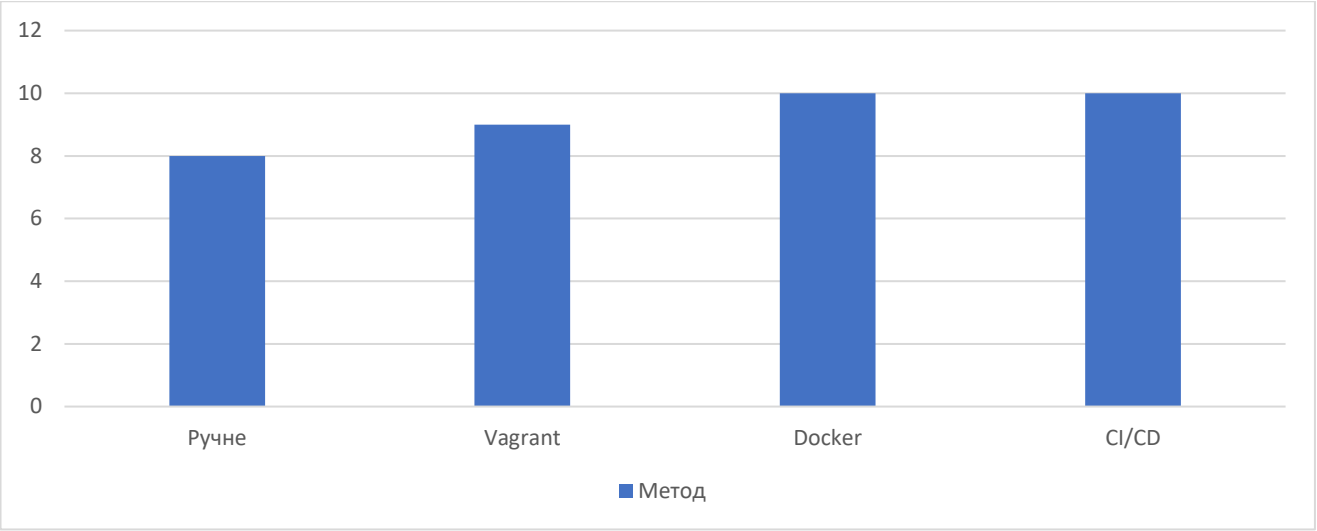

Рисунок 3.12 – Стабільність методів розгортання

Для доповнення загальної картини було побудовано графік пікового навантаження на ресурси системи — центральний процесор та оперативну пам’ять (див. рисунок 3.13). Він ілюструє ресурсоємність кожного підходу та допомагає оцінити їх придатність у середовищах з обмеженими ресурсами.

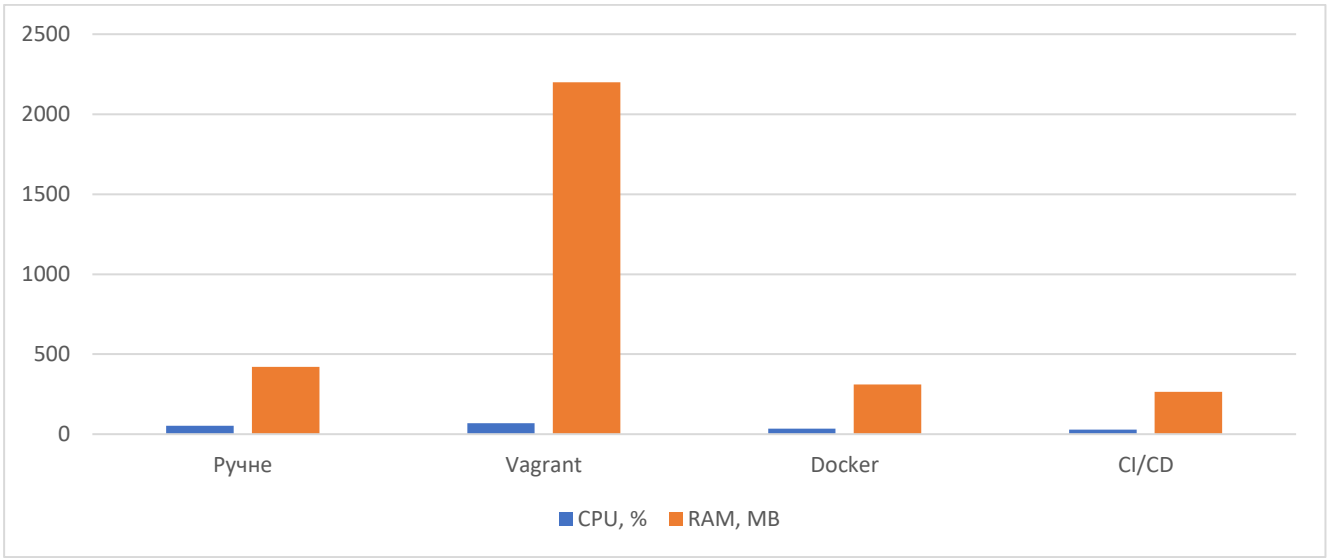

Рисунок 3.13 – Пікове споживання ресурсів *CPU*/*RAM*

Узагальнюючи результати, можна відзначити суттєві відмінності між методами за всіма критеріями. Ручне розгортання продемонструвало найбільшу мінливість часу та залежність від людського фактора, що робить його найменш

передбачуваним. *Vagrant* виявився стабільнішим, але загальний час виконання залишався значно вищим через необхідність запускати повноцінну віртуальну машину. *Docker* показав високу ефективність та швидкість, а *CI*/*CD* підтвердив свою перевагу завдяки повній автоматизації та мінімальній кількості відхилень.

Зібрані експериментальні дані створюють достатню основу не лише для фіксації окремих значень часу чи стабільності, а й для глибшого розуміння того, як поводяться різні методи в умовах повторюваних запусків. Самі по собі числові показники дають лише статичний зріз процесу, тоді як для повної оцінки важливо побачити їхню динаміку, внутрішні закономірності та характерні відхилення.

Саме тому наступним кроком стало статистичне опрацювання результатів — визначення взаємозв'язків між параметрами, оцінка варіацій та побудова узагальнених моделей, які дозволяють описати поведінку методів у стійкому режимі. Такий підхід робить можливим не лише пояснення відмінностей між ручними та автоматизованими способами розгортання, а й прогнозування їхньої ефективності при збільшенні кількості запусків або зміні умов середовища.

3.3 Аналіз отриманих експериментальних даних

Аналіз експериментальних даних дає змогу перейти від окремих вимірювань до розуміння загальної поведінки кожного з досліджуваних методів. Уже на етапі первинного огляду стало помітно, що різні підходи формують відміні закономірності коливань: ручний спосіб характеризується значною розбіжністю між окремими запусків, тоді як автоматизовані методи демонструють майже однорідні результати в усіх спробах. Побудовані графіки виконання підтверджують цю закономірність — у ручному та *Vagrant*-орієнтованому підходах точки розташовані нерівномірно, тоді як у *Docker* та *CI*/*CD* вони практично утворюють компактну область з мінімальними відхиленнями.

Такі особливості підтверджуються базовою статистичною обробкою даних. Середні значення часу для кожного методу демонструють суттєвий розрив між підходами: найдовшим виявився процес ручного розгортання, тоді як *Docker* та

CI/*CD* забезпечили на порядок швидший результат. Стандартні відхилення також відображають характер поведінки системи. Для ручного підходу вони були найбільшими, що свідчить про значну залежність від неконтрольованих факторів: повторні дії користувача, випадкові затримки або помилки під час налаштування середовища. У випадку *Vagrant* відхилення все ще залишалися помітними, але були меншими майже вдвічі, що свідчить про більшу передбачуваність. *Docker* та *CI*/*CD*, навпаки, показали мінімальні відхилення, які можна вважати природним фоновим шумом системи, а не закономірними коливаннями процесу.

Щоб підсумувати основні статистичні характеристики, було сформовано компактну таблицю 3.5, яка відображає середні значення ключових параметрів та їх варіації.

Таблиця 3.5 – Узагальнені середні значення та стандартні відхилення

Метод	**Середній час (с)**	***SD* часу**	**Середня кількість ручних дій**	**Стабільність**	***CPU* (%)**	***RAM* (*MB*)**
Ручне	1354	114.04	19	0.8	52	420
Vagrant	629.5	54.99	9	0.9	68	2200
Docker	30.5	3.03	3	1.0	34	310
CI/CD	20.1	2.02	1	1.0	29	265

Узагальнені дані демонструють чітку закономірність: рівень автоматизації прямо впливає на швидкість та стабільність процесу. Ручний метод, який покладається на великий обсяг взаємодій із користувачем, виявився найбільш непередбачуваним. Висока кількість ручних дій не лише збільшує час, але й створює умови, за яких окремі етапи можуть завершитися помилкою, що узгоджується з отриманими значеннями стабільності. *Vagrant* частково розв’язує цю проблему, але через запуск віртуальної машини й виконання додаткових *provisioning*-кроків загальний час залишається значним, а варіативність усе ще достатньо помітною.

Docker та *CI/CD* побудовані за принципово іншою логікою. Ізольованість контейнерів та детермінованість конфігурацій роблять кожен запуск близьким до ідентичного. У *Docker* час розгортання практично не залежить від стану системи, а *CI/CD* усуває навіть необхідність виклику процедури вручну, завдяки чому коливання часу стають мінімальними. Це також корелює зі споживанням ресурсів: *Docker* і *CI/CD* працюють у більш легкому середовищі, що знижує пікові навантаження та сприяє стабільнішому виконанню [27].

Таким чином, аналіз експериментальних даних показує послідовний та відчутний зв’язок між рівнем автоматизації та передбачуваністю розгортання. Чим більше етапів виконуються без участі користувача, тим менший вплив мають випадкові фактори, і тим ближчою до детермінованої стає поведінка системи в умовах повторюваних запусків. Цей висновок формує основу для подальшої багатокритеріальної оцінки методів, де результати будуть проаналізовані не лише на рівні окремих критеріїв, але й у комплексному інтегральному вигляді.

3.4 Інтегральний показник ефективності методів розгортання

Оцінювання методів розгортання лише за окремими параметрами дозволяє розглянути їхню роботу з різних боків, проте не дає чіткої відповіді на запитання, який із підходів є найбільш ефективним у комплексному застосуванні. Час розгортання показує швидкість, стабільність — надійність, кількість ручних дій — трудомісткість, а ресурси — поведінку системи під навантаженням. Усі ці характеристики важливі, але жодна з них окремо не формує повноцінне уявлення про загальну якість методу. Саме тому на цьому етапі формується інтегральний показник ефективності — зведена величина, що враховує всі критерії одночасно та дозволяє порівняти методи між собою у єдиній системі координат.

Основою для побудови інтегрального показника став метод аналізу ієрархій Сааті, який дозволяє визначити відносну важливість кожного критерію. На попередніх етапах дослідження була проведена парна порівняльна оцінка

критеріїв, у результаті якої було встановлено їх ваги графічне подання яких наведено на рисунку 3.14.

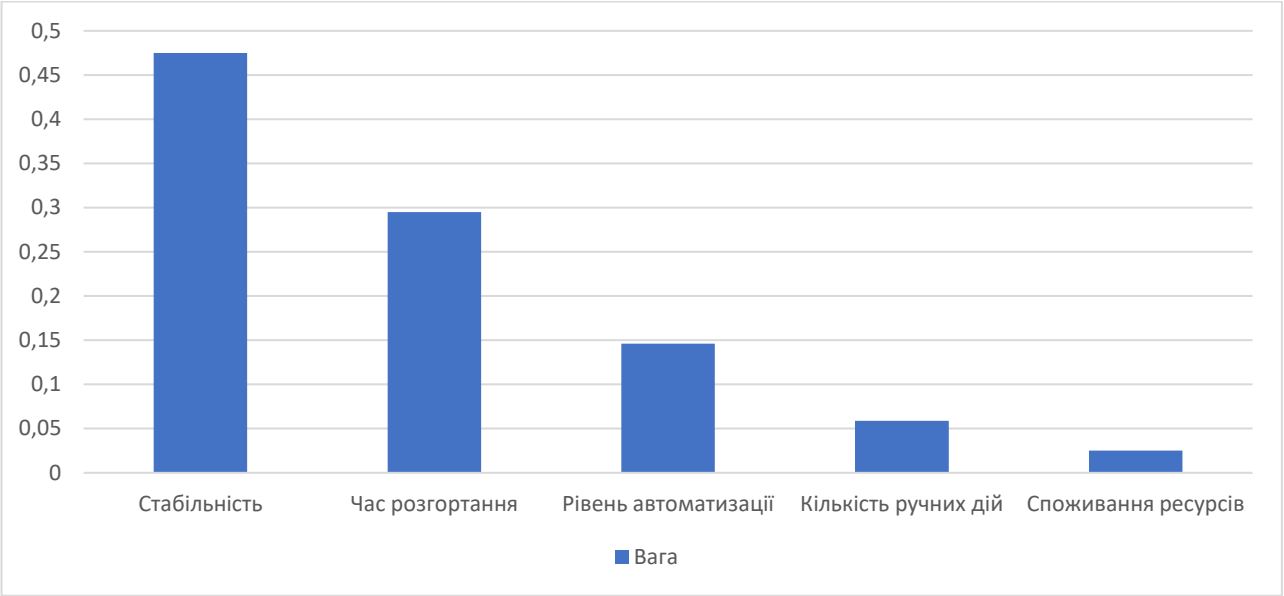

Рисунок 3.14 Ваги критеріїв ефективності

Найважливішим параметром виявилася стабільність розгортання (вага 0.475). Це цілком природно: незалежно від швидкості чи способу автоматизації, метод не може вважатися ефективним, якщо він не забезпечує передбачуване й стабільне виконання. На другому місці за значущістю опинився час розгортання (0.295), адже швидкість є критичною для систем, що потребують частих оновлень чи оперативного реакційного циклу. Далі йдуть рівень автоматизації (0.146) та трудомісткість у вигляді кількості ручних дій (0.059). Хоча ці критерії й мають менші ваги, вони безпосередньо впливають на людський фактор та ризик виникнення помилок під час розгортання. Найменшу вагу має ресурсоємність (0.025), проте цей критерій відіграє важливу роль у середовищах із обмеженими обчислювальними можливостями або у сценаріях масштабування.

Однак використати ці ваги напряму неможливо, оскільки самі критерії вимірюються в абсолютно різних одиницях. Час — у секундах. Стабільність — у частці успішних запусків. Кількість ручних дій — у штуках. Ресурси — у відсотках *CPU* та мегабайтах *RAM*. Це означає, що критерії спочатку необхідно перевести в

єдиний формат, який дозволить коректно поєднати їх у загальній формулі. Якщо цього не зробити, великий за масштабом параметр (наприклад, час у секундах) домінуватиме над іншими, навіть якщо його вага менша [38].

З цією метою було виконано лінійне нормування значень усіх критеріїв. На відміну від класичного мінімакс-підходу, який встановлює найгірше значення на рівні нуля, у цьому дослідженні застосовано модифікований варіант нормування з введенням нижньої межі у 0.1. Такий підхід дозволяє уникнути нульових значень і водночас зберігає пропорційність між методами. Значення «1» відповідає найкращому результату серед усіх методів, «0.1» — найгіршому, а проміжні значення відображають їх відносну якість.

Для критеріїв, де більше значення означає кращий результат (стабільність), застосовувалася формула прямого нормування:

$$S_i' = 0.1 + 0.9 \cdot \frac{S_i - S_{min}}{S_{max} - S_{min}} \quad (3.1)$$

де S_i — вихідне значення стабільності, а S_{min} і S_{max} відповідно мінімальне та максимальне значення серед методів.

Для критеріїв, де менше значення є кращим (час, кількість ручних дій, ресурси), використовувалася інверсна формула:

$$X_i' = 0.1 + 0.9 \cdot \frac{X_{max} - X_i}{X_{max} - X_{min}} \quad (3.2)$$

де X_i — вихідне значення відповідного критерію.

Застосування цієї процедури дозволило перетворити початкові вимірювання у порівнювані індекси в межах інтервалу [0.1; 1.0]. Найкращий результат за критерієм наближається до 1, а найгірший — до 0.1. Таким чином, жоден із методів не «обнуляється» повністю, а це особливо важливо при подальшому застосуванні вагових коефіцієнтів, визначених методом Сааті [37].

Після нормування кожен метод отрима в власний набір індексів, які відображають його відносну ефективність за всіма параметрами. Це створило можливість виконати об'єднання критеріїв у межах інтегральної оцінки, побудованої як зважена сума:

$$E = 0.475S' + 0.295T' + 0.146A' + 0.059H' + 0.025R' \quad (3.3)$$

де ваги відповідають відносній важливості критеріїв, визначеній раніше.

Проведення нормування дозволило подати всі критерії в уніфікованому вигляді та сформувати інтегральну оцінку, яка відображає відносну ефективність кожного методу розгортання. Для цього було застосовано ваги, визначені методом аналізу ієрархій, а також нормовані значення стабільності, часу виконання, рівня автоматизації, кількості ручних дій та споживання ресурсів. Отримані інтегральні індекси узагальнюють усі аспекти роботи методів та дозволяють провести повноцінне порівняння їх загальної придатності наведено в таблиці 3.6.

Таблиця 3.6 – Інтегральні оцінки ефективності методів розгортання

Метод	***S'***	***T'***	***A'***	***H'***	***R'***	**Рівень автоматизації**	**Інтегральний показник**
Ручне	0.10	0.10	0.10	0.10	0.929	Відсутня (0%)	0.163
Vagrant	0.55	0.589	0.612	0.600	0.10	Часткова (≈60%)	0.512
Docker	1.00	0.993	0.909	0.900	0.919	Висока (≈90%)	0.958
CI/CD	1.00	1.000	1.000	1.000	1.000	Повна (100%)	1.000

Інтегральна оцінка демонструє, що ефективність методів розгортання визначається не лише окремими технічними характеристиками, а й **ступенем автоматизації**, який істотно впливає як на стабільність роботи, так і на часові та ресурсні витрати.

Метод *CI/CD* займає провідну позицію, що є прямим наслідком його максимальної автоматизації. У цьому підході всі етапи — від отримання змін у

репозиторії до розгортання у робочому середовищі — виконуються автоматично. Саме відсутність ручних операцій забезпечує стабільність, передбачуваність та мінімальний час виконання. Автоматизація повністю усуває людський фактор, який є ключовим джерелом похибок у традиційних підходах, тому *CI*/*CD* отримує найвищі значення за кожним нормованим критерієм. Фактично інтегральний показник *CI*/*CD* відображає взаємопосилюючий вплив автоматизації на всі інші параметри.

Docker демонструє дуже близькі до максимальних значення інтегральної оцінки, що пояснюється високим ступенем автоматизації контейнерного середовища. Контейнери гарантують однаковість конфігурацій, швидкість запуску та відсутність необхідності вручну налаштовувати залежності. Втім, контейнеризація не охоплює автоматичний життєвий цикл доставки оновлень, тому *Docker* поступається *CI*/*CD* саме за рівнем автоматизації, що відображається у трохи нижчому значенні інтегрального показника.

Vagrant забезпечує часткову автоматизацію — насамперед на етапі створення та налаштування віртуального середовища. Однак запуск повноцінної віртуальної машини залишається ресурсоємним та потребує більше часу, ніж запуск контейнера або *CI*/*CD*-процесу. Часткова автоматизація дозволяє зменшити кількість ручних дій та підвищити повторюваність, проте не усуває накладних витрат, характерних для віртуальних машин. У результаті *Vagrant* займає середню позицію, демонструючи баланс між впорядкованістю процесу та високими затратами часу і ресурсів.

Ручне розгортання характеризується повною відсутністю автоматизації, через що вимагає максимальної кількості ручних дій. Це безпосередньо впливає на стабільність, варіативність часу виконання та загальну передбачуваність процесу. Відсутність автоматизованих механізмів робить ручне розгортання найбільш чутливим до помилок оператора та умов середовища, а це природно формує найнижчий інтегральний показник. Усі критерії, окрім споживання ресурсів, демонструють мінімальні нормовані значення, що свідчить про системну обмеженість цього підходу в умовах сучасних вимог до швидкості та

масштабованості. Для отримання узагальненої кількісної оцінки ефективності кожного з досліджуваних методів розгортання на основі нормованих значень критеріїв та їх вагових коефіцієнтів було обчислено інтегральний показник ефективності, результати розрахунку якого наведено на рисунку 3.15.

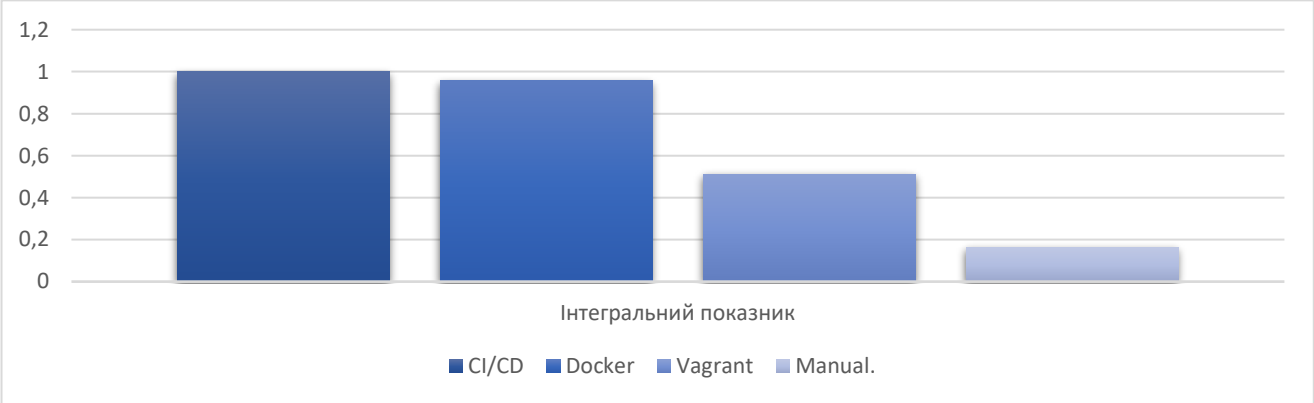

Рисунок 3.15 — Інтегральний показник ефективності методів розгортання

Узагальнюючи результати проведеного нормування та інтегральної оцінки, можна стверджувати, що саме рівень автоматизації визначає загальну ефективність методів розгортання у віртуальних середовищах. Підходи, які мінімізують вплив людського фактора та забезпечують відтворюваність кожного запуску, стабільно демонструють вищі показники за всіма критеріями.

Таким чином, інтегральний аналіз підтверджує перевагу сучасних автоматизованих підходів і дозволяє однозначно визначити напрям подальшої оптимізації процесів розгортання. Методи, що забезпечують автоматизацію, ізоляцію середовища та повторюваність операцій, створюють найкращі умови для масштабування та стабільної експлуатації веб-додатків у віртуальних інфраструктурах.

3.5 Математичне моделювання та статистичний аналіз результатів

Для того щоб кількісно оцінити поведінку кожного методу розгортання, проведено аналіз отриманих експериментальних даних за допомогою інструментів

статистики та регресії. Використання надбудови *Data Analysis* у *Excel* дозволило сформувати набори ключових показників — середні значення, варіативність, регресійні коефіцієнти та кореляції — і представити їх у вигляді таблиць і графіків.

Основою для аналізу стали дані десяти повторних запусків кожного методу. Така кількість вимірювань дає змогу працювати не з поодинокими значеннями, а з реальною вибіркою, за якою можна оцінити стабільність та характер розподілу часу. Параметри на зразок стабільності, ручних дій чи рівня автоматизації не змінюються при кожному запуску, тому розглядаються окремо — на рівні загального порівняння між методами, а не як статистичні ряди.

За допомогою процедури описової статистики були отримані ключові характеристики експериментальних вибірок, зокрема середні значення часу розгортання, стандартні відхилення, дисперсії, а також мінімальні та максимальні зафіксовані величини (таблиця 3.7). Сукупність цих показників дозволяє оцінити не лише типову тривалість виконання процесу для кожного методу, а й ступінь його стабільності при багаторазовому повторенні в однакових умовах.

Середні значення відображають загальний рівень часових витрат, тоді як медіана дає змогу оцінити репрезентативність цього значення та виявити можливий вплив поодиноких відхилень. Близькість середнього та медіанного значень у більшості методів свідчить про відсутність суттєвих перекосів у розподілі часу, що підтверджується також малими значеннями асиметрії.

Стандартне відхилення та дисперсія характеризують ступінь розсіювання результатів навколо середнього значення. Чим більшими є ці показники, тим менш передбачуваним є процес розгортання. Високі значення дисперсії для ручного методу вказують на значну залежність результатів від зовнішніх факторів і людського втручання, тоді як для автоматизованих підходів, зокрема Docker та CI/CD, спостерігається суттєво менший розкид значень. Це свідчить про стабільність і детермінований характер автоматизованих процесів.

Додаткову інформацію про форму розподілу надають коефіцієнти ексцесу та асиметрії. Їхні від’ємні значення вказують на відсутність різко виражених піків і свідчать про відносно рівномірний характер розподілу часу виконання. Розмах

значень, а також різниця між мінімальними та максимальними спостереженнями, дозволяють наочно порівняти чутливість кожного методу до варіацій умов виконання та підтверджують, що зі зростанням рівня автоматизації ця чутливість суттєво зменшується.

Таблиця 3.7 – Дисперсійні характеристики експериментальних даних для різних методів розгортання

Показник	**Ручне**	***Vagrant***	***Docker***	***CI/CD***
Середнє	1354	629,5	30,5	20,1
Стандартна похибка	36,0617	17,3917	0,9574	0,6403
Медіана	1362,5	622,5	30,5	20,5
Стандартне відхилення	114,037	54,9975	3,0277	2,0248
Дисперсія вибірки	13004,44	3024,72	9,17	4,1
Ексцес	–1,3868	–0,9726	–1,2	–1,3425
Асиметрія	–0,2558	–0,0230	0	–0,1686
Розмах	320	170	9	6
Мінімум	1180	540	26	17
Максимум	1500	710	35	23
Сума	13540	6295	305	201
Кількість спостережень	10	10	10	10

Отримані значення демонструють суттєві відмінності між методами. Ручний спосіб має найбільше стандартне відхилення (понад 114 секунд), що означає високий рівень стохастичності та значну залежність від людського фактора: навіть незначні затримки у виконанні окремих дій призводять до суттєвих коливань загального часу. Метод *Vagrant* демонструє середню варіативність (близько 55 секунд), що пов’язано з тим, що частина процесу делегована інструменту автоматизації, але сам запуск усе ще проходить у віртуальній машині, чутливій до

навантаження системи. Найкращі результати спостерігаються у *Docker* та *CI/CD*: їхні стандартні відхилення становлять відповідно приблизно 3 та 2 секунди, а розмах значень є мінімальним. Це означає, що обидва методи виконуються майже однаково при кожному повторенні, а їх робота характеризується практичною відсутністю випадкових відхилень.

Подальший аналіз був присвячений дослідженню залежності часу виконання від порядкового номера запуску. Кореляційний аналіз показав, що для ручного методу зв'язок між номером запуску та часом є незначним ($r \approx -0.10$), що свідчить про хаотичний характер зміни часу. Для *Vagrant* кореляція слабко позитивна ($r \approx 0.22$), але також статистично незначуща. У випадку *Docker* та *CI/CD* коефіцієнти кореляції практично дорівнюють нулю (–0.02…0.01), що підтверджує повну стабільність часу та відсутність накопичувальних ефектів.

Для формального підтвердження цих спостережень було побудовано лінійні регресійні моделі виду:

$$t(n) = a \cdot n + b \tag{3.4}$$

Параметри лінійних регресій, наведені в таблиці 3.8, дозволяють оцінити не лише загальний характер зміни часу розгортання при послідовних запусках, а й ступінь передбачуваності роботи кожного з розглянутих методів. Побудова регресійних моделей дає змогу виявити наявність систематичних тенденцій у динаміці часу виконання та відокремити їх від випадкових коливань, зумовлених зовнішніми факторами або нестабільністю процесу.

Коефіцієнт нахилу a відображає напрямок і швидкість зміни часу розгортання зі зростанням номера запуску. Додатні або від'ємні значення цього коефіцієнта можуть свідчити про наявність трендів, пов'язаних із накопиченням ефектів конфігурації, кешуванням або адаптацією середовища. Параметр b, що є вільним членом рівняння регресії, визначає усереднений базовий рівень часу, навколо якого відбуваються коливання результатів.

Коефіцієнт детермінації R^2 характеризує якість апроксимації експериментальних даних лінійною моделлю та показує, яку частку загальної варіації часу розгортання можна пояснити систематичною залежністю від номера запуску. Низькі значення R^2 свідчать про переважання випадкових коливань і, відповідно, про стохастичний та менш передбачуваний характер поведінки методу. Натомість вищі значення цього показника вказують на стабільність процесу та більшу керованість результатів.

Таблиця 3.8 – Параметри регресійних моделей

Метод	**Нахил *a***	**Вільний член *b***	**R^2**	**Інтерпретація**
Ручне	–3.818	1375.00	0.010	Повна відсутність тенденції, значний шум
Vagrant	+4.152	606.67	0.052	Слабка тенденція до збільшення часу
Docker	+0.0667	30.13	0.004	Практично горизонтальна лінія
CI/CD	–0.0061	20.13	0.00008	Ідеальна стабільність та передбачуваність

Для ручного способу нахил має значення –3.818, що формально вказує на незначне зменшення часу при переході до наступних запусків. Однак значення $R^2 = 0.010$ свідчить, що ця залежність є випадковою і не має практичного сенсу: модель пояснює лише 1% коливань часу. Це означає, що ручний метод максимально чутливий до зовнішніх факторів і дій користувача, а зміни в часі є хаотичними. Значний вільний член (1375 с) відображає високий середній рівень часу розгортання.

Метод *Vagrant* має позитивний нахил $a = 4.152$, тобто час запуску має слабку тенденцію до збільшення при повторенні. Це може бути наслідком зміни стану віртуальної машини або накопичення внутрішніх затримок, характерних для гостьового середовища. Проте значення $R^2 = 0.052$ залишається низьким, що вказує

на випадковий характер коливань і лише орієнтовну закономірність. Вільний член $b = 606.67$ узгоджується із середнім значенням часу розгортання, а характер графіка демонструє помірний розкид.

У *Docker* нахил лінії тренду становить лише $a = 0.0667$, що практично дорівнює нулю. Це означає, що час розгортання не залежить від номера запуску, а система працює однаково при будь-яких повторних виконаннях. Значення $R^2 = 0.004$ підтверджує відсутність тренду: лише 0,4% варіації часу можуть бути пояснені лінійною моделлю. Таким чином, *Docker* демонструє одну з найкращих стабільностей і найвищу передбачуваність серед досліджених методів.

Найбільш показовими є результати *CI*/*CD*. Нахил $a = -0.0061$ є настільки малим, що його можна вважати нульовим, а коефіцієнт $R^2 = 0.00008$ фактично дорівнює нулю. Це означає, що час виконання зовсім не залежить від номера запуску, а коливання є мінімальними й стохастично непомітними. Цей метод демонструє майже ідеальну повторюваність та максимальну автоматизацію, що повністю відповідає природі *CI*/*CD*-процесів — стандартизованих та відтворюваних сценаріїв.

Таким чином, порівняння параметрів регресійних моделей дозволяє чітко розмежувати категорії методів: ручне та частково автоматизоване розгортання (*Vagrant*) характеризуються помітними випадковими коливаннями, тоді як *Docker* і *CI*/*CD* демонструють майже незмінний час виконання, що робить їх оптимальними в сценаріях, де критично важливі передбачуваність і стабільність.

Побудова графіків лінійної регресії часу розгортання дозволяє наочно оцінити стабільність результатів та характер зміни показників у серії експериментів для кожного з досліджуваних методів.

Для ручного методу розгортання характерним є значний розкид експериментальних значень часу виконання відносно лінії тренду, що свідчить про істотну залежність результатів від зовнішніх факторів і людського впливу. Відхилення окремих вимірювань від усередненої тенденції вказують на відсутність стабільного та повторюваного сценарію виконання, у межах якого кожен запуск проходив би за однакових умов. Така поведінка є наслідком ручної конфігурації

середовища, варіативності послідовності дій та можливих помилок оператора, що наочно проілюстровано на рисунку 3.16.

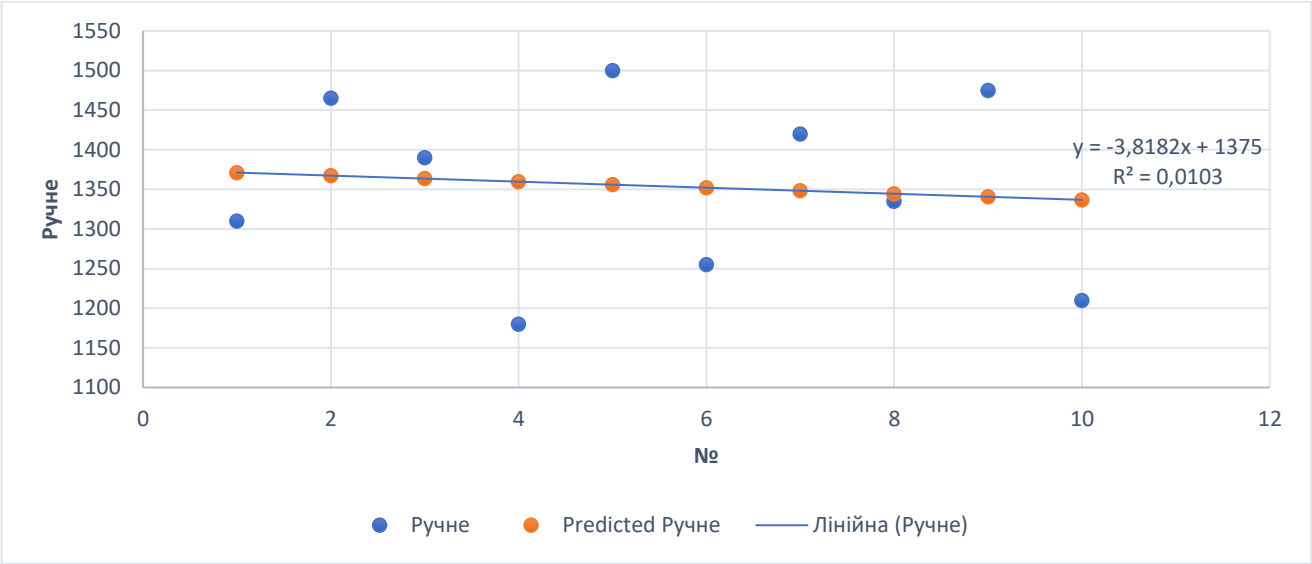

Рисунок 3.16 – Лінійна регресія часу розгортання для ручного методу

У випадку використання Vagrant спостерігається зменшення варіативності часу розгортання порівняно з ручним методом, однак збереження помітного розкиду точок відносно лінії регресії вказує пояснює недостатню стабільність цього підходу, що показано на рисунку 3.17.

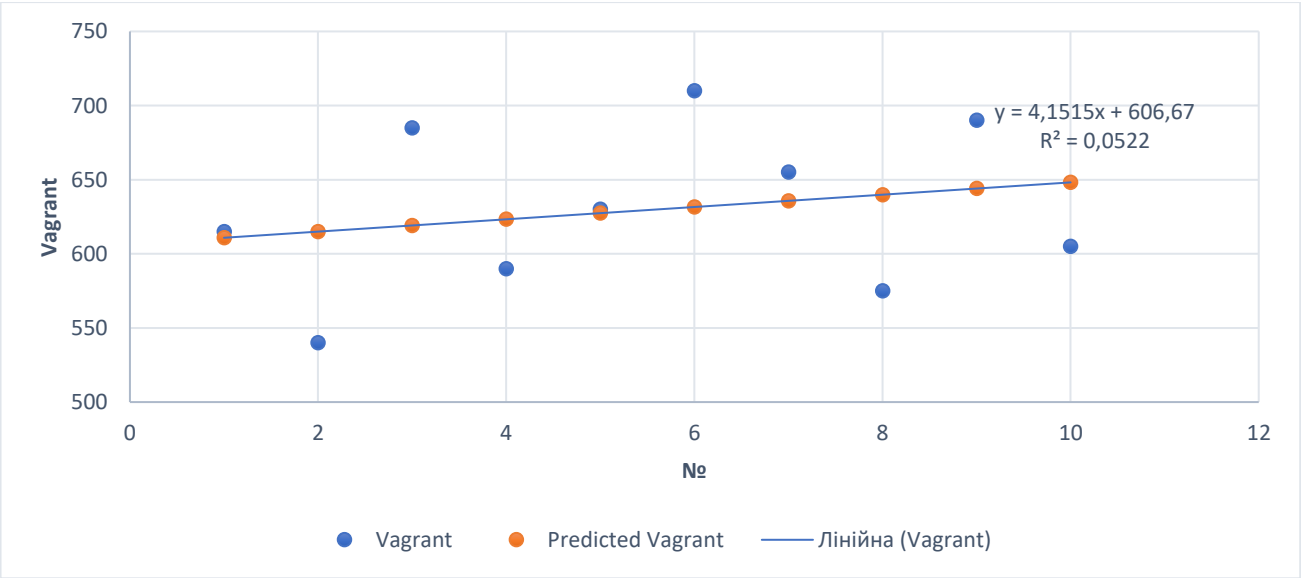

Рисунок 3.17 – Лінійна регресія часу розгортання для методу *Vagrant*

Для методу розгортання на основі Docker характерною є висока повторюваність результатів, що проявляється у щільному розміщенні точок поблизу горизонтальної лінії регресії, як показано на рисунку 3.18.

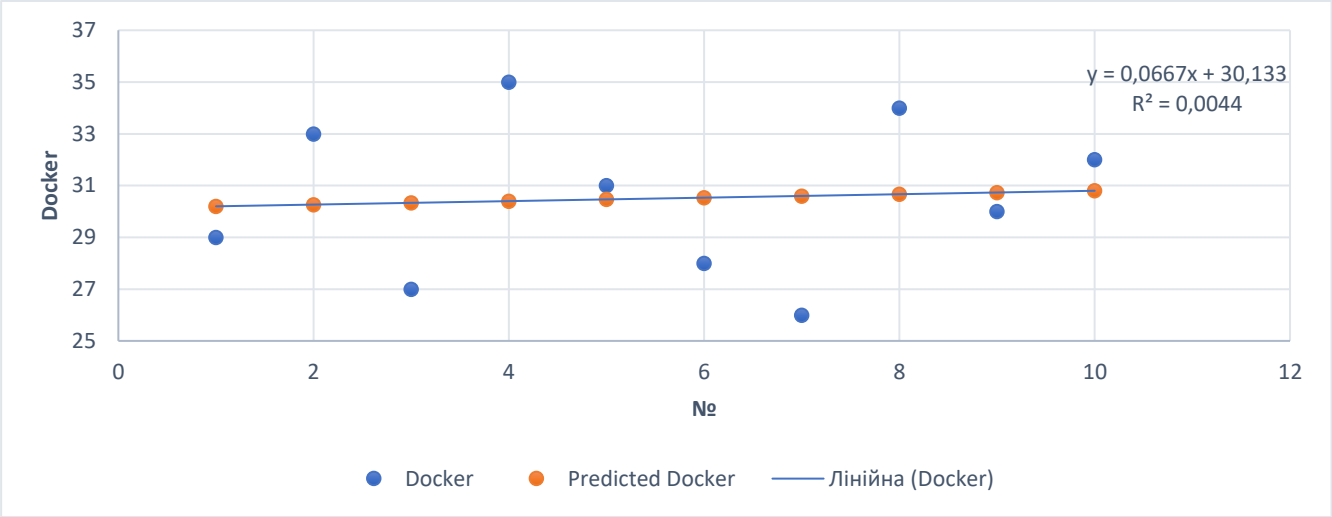

Рисунок 3.18 – Лінійна регресія часу розгортання для *Docker*

Аналогічна тенденція спостерігається і для CI/CD-підходу, де точки практично повністю прилягають до горизонтальної лінії регресії, що свідчить про максимальну стабільність та передбачуваність часу розгортання, що проілюстровано на рисунку 3.19.

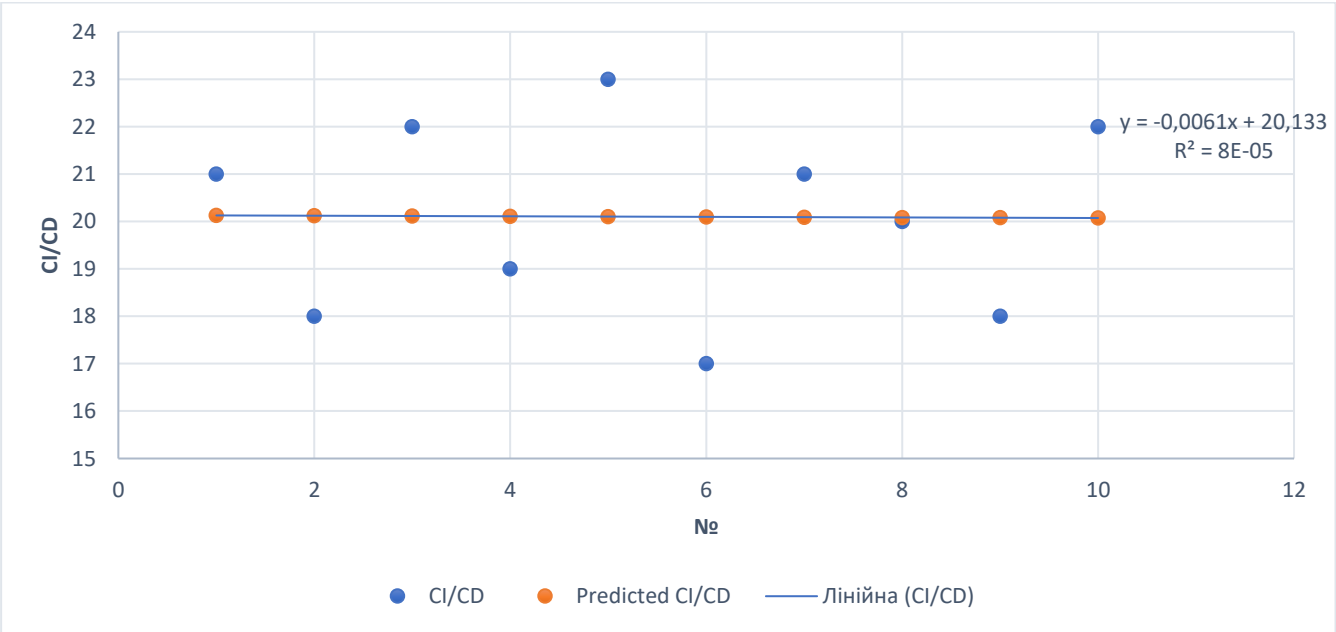

Рисунок 3.19 – Лінійна регресія часу розгортання для *CI/CD*

Окрім регресійного аналізу було розглянуто зв’язки між середнім часом розгортання та іншими характеристиками методів. Найсильніший зв’язок спостерігається між часом і стабільністю: коефіцієнт кореляції становить –0.996, що свідчить про практично лінійну залежність — чим вища стабільність, тим меншу тривалість має розгортання. Аналогічним чином, час сильно корелює з кількістю ручних дій ($r \approx +0.99$), що підтверджує негативний вплив людського фактора на швидкість і передбачуваність процесу.

Кореляційні залежності, наведені в таблиці 3.9, дозволяють оцінити, наскільки стабільним є кожен метод під час повторних запусків та як окремі характеристики впливають на середній час розгортання. Кореляція між номером запуску та часом у ручному методі становить –0.10, що свідчить про відсутність закономірності та випадковий характер змін часу. У *Vagrant* зв’язок слабко позитивний (0.22), однак також не є статистично стійким. Майже нульові значення *Docker* та *CI/CD* підтверджують, що повторні виконання цих методів дають практично однаковий результат незалежно від порядку запуску.

Таблиця 3.9 – Кореляційні залежності між часом та характеристиками методів

Параметри	**Кореляція *r***
Час ↔ № запуску (*Manual*)	–0.10
Час ↔ № запуску (*Vagrant*)	+0.22
Час ↔ № запуску (*Docker*)	≈ 0
Час ↔ № запуску (*CI/CD*)	≈ 0
Час ↔ Стабільність (між методами)	–0.996
Час ↔ Ручні дії (між методами)	+0.99

Найвиразніша залежність простежується між середнім часом та характеристиками методів. Дуже сильна негативна кореляція між часом і стабільністю (–0.996) означає, що стабільніші методи виконуються помітно швидше. Водночас сильна позитивна кореляція між часом і кількістю ручних дій

(≈+0.99) показує, що саме участь користувача є основним фактором уповільнення та варіативності.

Отримані статистичні характеристики та кореляційні залежності дозволяють чітко зрозуміти природу роботи кожного з досліджених методів. Ручний спосіб зберігає найбільшу варіативність і нерівномірність, тоді як *Docker* і *CI/CD* демонструють майже незмінний час виконання та мінімальну чутливість до зовнішніх факторів. Узагальнена картина підтверджує ключову роль автоматизації у підвищенні стабільності та передбачуваності процесу, а отримані статистичні результати природно продовжують загальний аналіз ефективності методів, дозволяючи надалі розглядати їх у ширшому порівняльному контексті.

Висновки до розділу

Експериментальні дослідження, проведені в рамках цього розділу, дозволили всебічно оцінити поведінку різних методів розгортання вебдодатків у контрольованих умовах. Ідентичність програмного середовища, використання єдиного вихідного коду та стандартизована методика вимірювань забезпечили коректність порівняння та відтворюваність результатів. Розгортання проводилося на єдиному тестовому стенді, а час, навантаження на ресурси, кількість ручних дій та стабільність фіксувалися за уніфікованою процедурою, що гарантувало точність подальшого статистичного аналізу.

Отримані дані продемонстрували суттєві відмінності між ручними й автоматизованими підходами. Ручний спосіб показав найбільшу варіабельність часу, підвищену чутливість до людського фактора та високі значення дисперсії. Метод на основі *Vagrant* частково зменшив кількість ручних операцій, проте зберіг значну інерційність старту та нерівномірність результатів між запуском і запуском. Статистичний аналіз підтвердив ці висновки: дисперсійні характеристики ручного й *Vagrant*-підходів істотно перевищують відповідні показники *Docker* і *CI/CD*, що свідчить про нижчу передбачуваність та стабільність традиційних методів.

Побудовані регресійні моделі дозволили оцінити тенденції зміни часу виконання при збільшенні кількості запусків. Автоматизовані підходи демонструють практично лінійну, стійку поведінку без значних відхилень, тоді як ручний спосіб характеризується нерівномірним розподілом значень та тенденцією до зростання часу в окремих спробах. Кореляційний аналіз підтвердив: між кількістю запусків і часом виконання для *Docker* та *CI*/*CD* не спостерігається істотних залежностей, що вказує на їхню високу відтворюваність, тоді як ручні методи демонструють помітну кореляцію з нерівномірністю навантаження середовища.

Застосований інтегральний показник ефективності, який об’єднує стабільність, час виконання, рівень автоматизації, кількість ручних дій та ресурсоспоживання, дав змогу сформувати узагальнену оцінку кожного методу. Отримані значення чітко розмежували методи за рівнем ефективності: *CI*/*CD* отримав найвищий результат завдяки повній автоматизації та мінімальній варіабельності параметрів; *Docker* посів друге місце, поєднуючи стабільність із швидким повторним розгортанням; *Vagrant* опинився посередині через часткову автоматизацію та тривалий час ініціалізації; ручний спосіб закономірно отримав найнижчий інтегральний бал через нестабільність і значну кількість ручних дій.

Узагальнюючи результати, можна стверджувати, що рівень автоматизації є ключовим чинником підвищення продуктивності та передбачуваності процесів розгортання. Автоматизовані методи не лише зменшують час виконання та кількість помилок, а й забезпечують стабільнішу поведінку системи у довгих серіях запусків, що підтверджено як статистичними показниками, так і результатами математичного моделювання. Це дозволяє розглядати *CI*/*CD* та контейнеризацію як найбільш ефективні та перспективні інструменти для сучасних систем розгортання в умовах інтенсивної розробки та потреби в безперервності сервісів.

ВИСНОВКИ

У кваліфікаційній роботі проведено комплексне дослідження процесів і методів розгортання додатків, які в сукупності формують систему розгортання у віртуальному середовищі. Розглянуто як ручні методи, так і автоматизовані підходи, що забезпечують керованість і відтворюваність результатів. У роботі процес розгортання інтерпретується як цілісний інженерний механізм, що об’єднує інфраструктурні компоненти, середовище виконання та засоби автоматизації.

Проаналізовано віртуалізацію як базову технологію сучасних середовищ розгортання та узагальнено еволюцію підходів до розгортання програмних систем. Встановлено, що традиційні ручні методи характеризуються низькою передбачуваністю результатів, підвищеною трудомісткістю та значною залежністю від людського фактора, що обмежує можливості їх використання в умовах зростання складності інформаційних систем. Обґрунтовано доцільність переходу до автоматизованих підходів як засобу підвищення стабільності, відтворюваності та керованості процесів розгортання.

Сформовано узагальнену модель системи розгортання додатків у віртуальному середовищі, у межах якої процес розгортання подано як послідовність формалізованих етапів із визначеними станами та контрольними точками. Запропонований підхід створює основу для системного аналізу різних реалізацій систем розгортання та їх порівняння за єдиною методологією.

Обґрунтовано систему критеріїв оцінювання ефективності систем розгортання, що охоплює технічні та експлуатаційні характеристики процесу, зокрема часові показники, стабільність виконання, рівень автоматизації та залежність від ручних операцій. Для визначення відносної значущості критеріїв застосовано метод аналізу ієрархій, що дозволило формалізувати багатокритеріальний підхід до оцінювання та зменшити суб’єктивність прийняття рішень.

Розроблено методику комплексного оцінювання ефективності систем розгортання на основі інтегрального показника, який забезпечує узагальнення різних характеристик процесу в єдину кількісну оцінку. Проведене експериментальне дослідження підтвердило, що зі зростанням рівня автоматизації зменшуються варіації часу розгортання, підвищується стабільність процесу та знижується залежність від людського фактора. Автоматизовані системи розгортання продемонстрували вищу передбачуваність результатів порівняно з ручними підходами.

Отримані результати свідчать, що рівень автоматизації є визначальним чинником ефективності систем розгортання додатків у віртуальному середовищі, оскільки безпосередньо впливає на стабільність процесів, передбачуваність часових характеристик та надійність функціонування системи в цілому. Застосування інтегрального показника дозволило здійснити комплексне порівняння систем розгортання за сукупністю критеріїв ефективності та виявити загальні тенденції зміни результативності залежно від рівня автоматизації. Подальший статистичний аналіз результатів багаторазових запусків підтвердив отримані узагальнені висновки, показавши, що зі зростанням рівня автоматизації зменшується розкид значень часу розгортання та підвищується повторюваність результатів, що свідчить про зниження впливу неконтрольованих факторів і людського втручання. Запропонований підхід може бути використаний як методологічна основа для аналізу та вибору систем розгортання в умовах обмежених ресурсів і нестабільної інфраструктури.

Наукова новизна роботи полягає у поєднанні системного моделювання процесу розгортання, багатокритеріального аналізу та експериментальної перевірки ефективності різних підходів на основі єдиного програмного об’єкта. Практичне значення отриманих результатів полягає у можливості застосування розробленої методики для обґрунтованого вибору способів розгортання додатків під час проєктування та модернізації інформаційних систем у різних сферах застосування

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Silberschatz A., Galvin P. B., Gagne G. Operating System Concepts. – 11th ed. – Hoboken : Wiley, 2022. – 976 с.
2. Hwang K., Fox G. C., Dongarra J. J. Distributed and Cloud Computing: From Parallel Processing to the Internet of Things. – 2nd ed. – Morgan Kaufmann, 2022. – 560 с.
3. Pahl C., Jamshidi P. Cloud Container Technologies: A State-of-the-Art Review // IEEE Transactions on Cloud Computing. – 2021. – Т. 9, № 3. – С. 1097–1112.
4. Zhang Q., Chen M., Li L. Virtualization and Containerization Technologies in Cloud Computing // Journal of Cloud Computing. – 2021. – Т. 10, № 1.
5. Kim G., Humble J., Debois P., Willis J. The DevOps Handbook. – 2nd ed. – Portland : IT Revolution Press, 2021. – 480 с.
6. Bass L., Weber I., Zhu L. DevOps: A Software Architect’s Perspective. – Updated ed. – Boston : Addison-Wesley, 2022. – 360 с.
7. Shahin M., Ali Babar M., Zhu L. Continuous Integration, Delivery and Deployment: A Systematic Review // IEEE Software. – 2021. – Т. 38, № 2. – С. 27–34.
8. Rahman A. A., Williams L. Software Deployment Pipelines: A Research Review // ACM Computing Surveys. – 2022. – Т. 55, № 4.
9. Forsgren N., Humble J., Kim G. Accelerate: The Science of Lean Software and DevOps. – Updated ed. – Portland : IT Revolution Press, 2023. – 288 с.
10. Burns B., Beda J., Hightower K. Kubernetes Patterns. – 2nd ed. – Sebastopol : O’Reilly Media, 2022. – 320 с.
11. VMware. Introduction to Virtualization [Електронний ресурс]. – URL: https://www.vmware.com/topics/virtualization (дата звернення: 30.09.2025).
12. IBM. Hybrid Cloud and Automation [Електронний ресурс]. – URL: https://www.ibm.com/cloud (дата звернення: 03.10.2025).
13. Linux Foundation. Container Fundamentals [Електронний ресурс]. – URL: https://www.linuxfoundation.org (дата звернення: 08.10.2025).

14. Red Hat. What is Infrastructure as Code [Електронний ресурс]. – URL: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac (дата звернення: 02.11.2025).

15. HashiCorp. Vagrant Documentation [Електронний ресурс]. – URL: https://developer.hashicorp.com/vagrant (дата звернення: 06.11.2025).

16. Docker Inc. Docker Documentation [Електронний ресурс]. – URL: https://docs.docker.com (дата звернення: 11.11.2025).

17. Cloud Native Computing Foundation. Cloud Native Landscape [Електронний ресурс]. – URL: https://landscape.cncf.io (дата звернення: 15.11.2025).

18. Kubernetes. Official Documentation [Електронний ресурс]. – URL: https://kubernetes.io/docs (дата звернення: 20.11.2025).

19. GitHub. GitHub Actions Documentation [Електронний ресурс]. – URL: https://docs.github.com/actions (дата звернення: 02.12.2025).

20. Google Cloud. CI/CD Best Practices [Електронний ресурс]. – URL: https://cloud.google.com/docs/ci-cd (дата звернення: 05.12.2025).

21. Amazon Web Services. DevOps on AWS [Електронний ресурс]. – URL: https://aws.amazon.com/devops
(дата звернення: 09.12.2025).

22. Microsoft. Azure DevOps Documentation [Електронний ресурс]. – URL: https://learn.microsoft.com/devops (дата звернення: 13.12.2025).

23. Flask. Official Documentation [Електронний ресурс]. – URL: https://flask.palletsprojects.com (дата звернення: 17.12.2025).

24. SQLite. Official Documentation [Електронний ресурс]. – URL: https://www.sqlite.org/docs.html (дата звернення: 19.12.2025).

25. Oracle. VirtualBox User Manual [Електронний ресурс]. – URL: https://www.virtualbox.org/manual (дата звернення: 22.12.2025).

26. Git SCM. Git Documentation [Електронний ресурс]. – URL: https://git-scm.com/doc (дата звернення: 24.12.2025).

27. Google Cloud. State of DevOps Report 2023 [Електронний ресурс] – URL: https://services.google.com/fh/files/misc/2023_final_report_sodr.pdf (дата звернення: 26.12.2025).

28. Google Cloud. State of DevOps Report 2024 [Електронний ресурс] – URL: https://services.google.com/fh/files/misc/2024_final_dora_report.pdf (дата звернення: 27.12.2025).

29. Stack Overflow. Developer Survey 2024 [Електронний ресурс]. – URL: https://survey.stackoverflow.co/2024 (дата звернення: 28.12.2025).

30. McKinsey & Company. The Value of DevOps [Електронний ресурс]. – URL:https://www.mckinsey.com/capabilities/people-and-organizational-performance/our-insights/the-value-of-devops (дата звернення: 29.12.2025).

31. Gartner. DevOps Market Trends [Електронний ресурс]. – URL: https://www.gartner.com/en/information-technology/insights/devops (дата звернення: 30.12.2025).

32. IBM. Cloud Automation and DevOps [Електронний ресурс]. – URL: https://www.ibm.com/automation/devops (дата звернення: 31.12.2025).

33. NIST SP 800-53 Rev. 5. Security and Privacy Controls for Information Systems. – 2023. – URL: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final (дата звернення: 31.12.2025).

34. McKinsey & Company. The Value of DevOps [Електронний ресурс]. - URL: https://www.mckinsey.com/capabilities/people-and-organizationalperformance/our-insights/the-value-of-devops (дата звернення: 15.01.2025).

35. Gartner. DevOps Market Trends [Електронний ресурс]. – 2024. – URL: https://www.gartner.com/en/information-technology/insights/devops (дата звернення: 18.01.2025).

36. IBM. Cloud Automation and DevOps [Електронний ресурс]. – 2025. – URL: https://www.ibm.com/automation/devops (дата звернення: 22.01.2025).

37. Саати Т. Л. Прийняття рішень. Метод аналізу ієрархій / пер. з англ. – Київ : Основи, 2021. – 280 с.

38. Семенов В. В., Петренко І. С. Застосування методу аналізу ієрархій для оцінювання ефективності інформаційних систем // Наукові праці Національного авіаційного університету. – 2023. – № 1. – С. 45–52.

39. КПІ ім. Ігоря Сікорського. Метод аналізу ієрархій (AHP) [Електронний ресурс]. – URL: https://ela.kpi.ua/handle/123456789/ahp (дата звернення: 20.12.2025).

Основні файли проєкту

```
#__init__.py
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager
from config import Config
import os
db = SQLAlchemy()
login_manager = LoginManager()
login_manager.login_view = 'main.login'

def create_app():
    app = Flask(__name__)
    app.config.from_object(Config)

    db.init_app(app)
    login_manager.init_app(app)

    with app.app_context():
        # імпортуємо моделі для створення таблиць
        from . import models
        db.create_all()

        # blueprints / маршрути
        from .routes import bp as main_bp
        app.register_blueprint(main_bp)

        if models.User.query.count() == 0:
```

```
        admin = models.User(username='admin', email='admin@example.com',
is_admin=True)
        admin.set_password('admin123')
        db.session.add(admin)
        db.session.commit()

    return app

#routes.py

from flask import Blueprint, render_template, redirect, url_for, flash, request,
current_app, jsonify, abort
from .models import User, Profile, ActivityLog, News
from . import db
from .forms import RegisterForm, LoginForm, ProfileForm, NewsForm
from flask_login import login_user, logout_user, login_required, current_user
from .utils import save_avatar
import psutil
from functools import wraps

def admin_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if not current_user.is_admin:
            abort(403)
        return f(*args, **kwargs)
    return decorated_function

bp = Blueprint('main', __name__)
```

```
@bp.route('/')
def index():
    news_list = News.query.order_by(News.created_at.desc()).all()
    return render_template('index.html', news_list=news_list)


@bp.route('/register', methods=['GET', 'POST'])
def register():
    form = RegisterForm()
    if form.validate_on_submit():
        if User.query.filter((User.username == form.username.data) | (User.email ==
form.email.data)).first():
            flash('Користувач із таким іменем або email вже існує', 'warning')
            return redirect(url_for('main.register'))
        user = User(username=form.username.data, email=form.email.data)
        user.set_password(form.password.data)
        db.session.add(user)
        db.session.commit()
        profile = Profile(user_id=user.id)
        db.session.add(profile)
        db.session.commit()
        # лог активності
        log = ActivityLog(user_id=user.id, action='register')
        db.session.add(log)
        db.session.commit()
        flash('Реєстрація успішна. Увійдіть у систему.', 'success')
        return redirect(url_for('main.login'))
    return render_template('register.html', form=form)


@bp.route('/login', methods=['GET', 'POST'])
```

```
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user and user.check_password(form.password.data):
            login_user(user, remember=form.remember.data)
            log = ActivityLog(user_id=user.id, action='login')
            db.session.add(log)
            db.session.commit()
            flash('Успішний вхід', 'success')
            return redirect(url_for('main.dashboard'))
        flash('Невірні дані', 'danger')
    return render_template('login.html', form=form)

@bp.route('/logout')
@login_required
def logout():
    log = ActivityLog(user_id=current_user.id, action='logout')
    db.session.add(log)
    db.session.commit()
    logout_user()
    return redirect(url_for('main.index'))

@bp.route('/profile', methods=['GET', 'POST'])
@login_required
def profile():
    form = ProfileForm()
    profile = current_user.profile
    if form.validate_on_submit():
```

```
        if form.full_name.data:
            profile.full_name = form.full_name.data
        profile.bio = form.bio.data
        if form.avatar.data:
            avatar_path = save_avatar(form.avatar.data, f'user{current_user.id}')
            if avatar_path:
                profile.avatar = avatar_path
        db.session.commit()
        log = ActivityLog(user_id=current_user.id, action='update_profile')
        db.session.add(log)
        db.session.commit()
        flash('Профіль оновлено', 'success')
        return redirect(url_for('main.profile'))
    return render_template('profile.html', form=form, profile=profile)
@bp.route('/dashboard')
@login_required
def dashboard():
    if not current_user.is_admin:
        flash('Недостатньо прав', 'danger')
        return redirect(url_for('main.index'))
    users = User.query.order_by(User.created_at.desc()).all()
    return render_template('dashboard.html', users=users)

@bp.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

@bp.errorhandler(500)
def server_error(e):
```

```
    return render_template('500.html'), 500


@bp.route('/metrics')
def metrics():
    p = psutil.Process()
    mem = p.memory_info().rss / 1024 / 1024  # MB
    cpu = p.cpu_percent(interval=0.1)
    return jsonify({
        'cpu_percent': cpu,
        'memory_mb': round(mem, 2)
    })

@bp.route('/admin/add-news', methods=['GET', 'POST'])
@login_required
@admin_required
def add_news():
    form = NewsForm()
    if form.validate_on_submit():
        new_article = News(
            title=form.title.data,
            summary=form.summary.data,
            content=form.content.data,
            image_url=form.image_url.data,
            user_id=current_user.id
        )
        db.session.add(new_article)
        db.session.commit()
```

```
        log = ActivityLog(user_id=current_user.id, action='add_news')
        db.session.add(log)
    db.session.commit()
        flash('Новину успішно додано!', 'success')
        return redirect(url_for('main.index'))
    return render_template('add_news.html', form=form)

@bp.route('/news/<int:news_id>')
def news_detail(news_id):
    news = News.query.get_or_404(news_id)
    return render_template('news_detail.html', news=news)
```

Додаток Б

Конфігураційний файл контейнеризації *Dockerfile*

```
FROM python:3.13-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
ENV PYTHONUNBUFFERED=1
EXPOSE 5000
CMD ["python", "app.py"]
```

Файл сценарію автоматизованого розгортання *CI/CD*

```
name: CI Pipeline

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: "3.13"
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
    - name: Run test script
      run: |
        python -c "print('CI/CD pipeline executed successfully')"
```

Конфігураційний файл середовища віртуалізації *Vagrantfile*

```
Vagrant.configure("2") do |config|

  config.vm.box = "ubuntu/focal64"
  config.vm.hostname = "flask-deploy-vm"
  config.vm.network "forwarded_port", guest: 5000, host: 5000
  config.vm.provider "virtualbox" do |vb|
    vb.name = "FlaskDeploymentVM"
    vb.memory = "2048"
    vb.cpus = 2
  end

  config.vm.synced_folder ".", "/vagrant", type: "virtualbox"

  config.vm.provision "shell", inline: <<-SHELL
    sudo apt-get update -y
    sudo apt-get install -y python3 python3-pip python3-venv

    python3 -m venv /home/vagrant/venv
    source /home/vagrant/venv/bin/activate

    pip install --upgrade pip
    pip install flask

    echo 'export FLASK_APP=app.py' >> /home/vagrant/.bashrc
    echo 'export FLASK_RUN_HOST=0.0.0.0' >> /home/vagrant/.bashrc
  SHELL
end
```