МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ПЛАТФОРМИ АВТОМАТИЗОВАНОГО РОЗГОРТАННЯ ІЗОЛЬОВАНИХ PRODUCTION-ПОДІБНИХ СЕРЕДОВИЩ»**

Виконав: студент 2 курсу, групи 8.1224-он
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньо-наукової програми Комп'ютерні науки
(назва освітньої програми)
В.В. Калюжняк
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітньо-наукова програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.М.
(підпис)

" ____ " _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Калюжняку Віталію Вадимовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка платформи автоматизованого розгортання ізольованих production-подібних середовищ

керівник роботи Борю Сергій Юрійович, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 28 » лютого 2025 року № 326-с

2. Строк подання студентом роботи 20.04.2026

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка платформи автоматизованого розгортання ізольованих production-подібних середовищ.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 06.03.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	06.03.2025	
2.	Збір вихідних даних.	29.05.2025	
3.	Обробка методичних та теоретичних	20.09.2025	
	джерел.		
4.	Розробка першого та другого розділу.	15.12.2025	
5.	Розробка третього розділу.	11.03.2026	
6.	Оформлення та нормоконтроль	20.04.2026	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	20.06.2026	

Студент _____ В.В. Калюжняк
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.Ю. Борю
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка платформи автоматизованого розгортання ізольованих production-подібних середовищ»: 56 с., 28 рис., 1 табл., 18 джерел, 1 додаток.

АВТОМАТИЗОВАНЕ РОЗГОРТАННЯ, ІЗОЛЯЦІЯ СЕРЕДОВИЩ, КОНТЕЙНЕРИЗАЦІЯ, ОРКЕСТРАЦІЯ, ХМАРНІ ТЕХНОЛОГІЇ, DEVOPS, CI/CD.

Об’єкт дослідження – процес автоматизованого розгортання та управління ізольованими середовищами в життєвому циклі програмного забезпечення.

Предмет дослідження – методи, засоби та архітектурні рішення задля розробки платформи автоматизованого розгортання production-подібних середовищ із використанням контейнеризації та оркестрації.

Мета роботи: дослідити та розробити платформу автоматизованого розгортання ізольованих production-подібних середовищ, яка забезпечить масштабованість, ізоляцію та наближеність до реального production-середовища в процесі розробки та тестуванні.

Методи дослідження – теоретичний аналіз наукових та технічних джерел, системний аналіз, моделювання архітектури програмних систем, тестування, порівняльний аналіз існуючих платформ і рішень.

У роботі реалізовано прототип платформи та проведено тестування її продуктивності, масштабованості та стабільності. Отримані результати підтвердили ефективність використання контейнерної оркестрації для автоматизації процесів розгортання та підтримки production-подібних середовищ.

Розроблений прототип може бути використаний як основа для створення інфраструктурних рішень.

SUMMARY

Master’s Qualification Theses «Development of a Platform for Automated Deployment of Isolated Environments Similar to Production»: 56 pages, 28 figures, 1 table, 18 references, 1 supplement.

AUTOMATED DEPLOYMENT, ENVIRONMENT ISOLATION, CONTAINERIZATION, ORCHESTRATION, CLOUD TECHNOLOGIES, DEVOPS, CI/CD.

Object of the study – the process of automated deployment and management of isolated environments throughout the software lifecycle.

Subject of the study – methods, tools, and architectural solutions for developing a platform for automated deployment of production-like environments using containerization and orchestration.

Aim of the study: to research and develop a platform for automated deployment of isolated production-like environments that ensures scalability, isolation, and closeness to a real production environment in the development and testing process.

Methods of research – theoretical analysis of scientific and technical sources, system analysis, modeling of software system architecture, testing, and comparative analysis of existing platforms and solutions.

This thesis presents the implementation of a prototype platform and the evaluation of its performance, scalability, and operational stability. The obtained results confirmed the effectiveness of container orchestration technologies for automating deployment processes and supporting production-like environments.

The developed prototype can serve as a foundation for building infrastructure solutions intended for the automated deployment, testing, and management of isolated software environments.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Теоретичні основи автоматизованого розгортання середовищ 10
1.1 Класифікація середовищ розробки, тестування та експлуатації..... 10
1.2 Вимоги, характеристики та роль production-подібних середовищ у життєвому циклі ПЗ..... 12
1.3 Контейнеризація як основа ізоляції середовищ..... 14
1.4 Оркестрація контейнерів та керування інфраструктурою..... 16
2 Проєктування платформи автоматизованого розгортання..... 21
2.1 Аналіз існуючих рішень та платформ..... 21
2.2 Вимоги до платформи автоматизованого розгортання ізольованих середовищ 24
2.2.1 Нефункціональні вимоги..... 25
2.2.2 Функціональні вимоги..... 26
2.3 Архітектура платформи та опис основних компонентів 28
2.4 Модель безпеки та ізоляції production-подібних середовищ..... 31
2.5 Проєктування автоматизованого рішення..... 33
2.5.1 Діаграма використання..... 33
2.5.2 Діаграма прецедентів..... 35
3 Програмна реалізація та тестування платформи 38
3.1 Реалізація механізму автоматизованого розгортання середовищ..... 38
3.2 Інтеграція з системами керування версіями та CI/CD 41
3.3 Тестування платформи ізольованих production-подібних середовищ42
3.4 Аналіз продуктивності та стабільності платформи..... 46

3.5 Практичне застосування платформи у навчальному та науковому процесі ……48
Висновки 51
Перелік посилань..... 533
Додаток А Код Laravel-ендпоінту для webhook..... 555

ВСТУП

Сучасна розробка програмного забезпечення характеризується високою динамікою змін, зростанням складності інформаційних систем та широким використанням хмарних і мікросервісних архітектур. Тому особливої актуальності постає питання автоматизації процесів розгортання, тестування та експлуатації програмних продуктів, забезпечення стабільності та передбачуваності їх роботи.

На сьогодні програмні системи активно застосовуються в різних галузях, а саме:

- інформаційні технології та веб-сервіси – хмарні платформи, SaaS-рішення, розподілені веб-застосунки;
- фінансовий сектор – платіжні системи, онлайн-банкінг, аналітичні сервіси;
- освіта та наукові дослідження – навчальні платформи, віртуальні лабораторії, експериментальні середовища;
- промисловість та бізнес – ERP-системи, системи автоматизації бізнес-процесів;
- державні та соціальні сервіси – портали електронного урядування, інформаційні системи масового користування [1].

Ключові проблеми при розробці та супроводі даних систем є відмінність між середовищами розробки, тестування та реальної експлуатації (production). Невідповідність конфігурацій, версій програмного забезпечення та інфраструктури часто призводить до помилок, які проявляються лише після розгортання у production-середовищі. Тому для мінімізації ризиків у сучасних DevOps-практиках використовуються production-подібні середовища, які дозволять відтворювати умови реальної експлуатації в ізольованому та безпечному вигляді.

Актуальність розробки production-подібних середовищ зумовлена потребою підвищити якість та стабільність програмного забезпечення, скоротити час між розробкою та впровадженням змін, зменшити кількість помилок у production-середовищі та автоматизувати процеси розгортання інфраструктури. Тому, реалізація платформи автоматизованого розгортання ізольованих production-подібних середовищ є актуальною науково-технічною задачею.

Метою цієї кваліфікаційної роботи є розробка та дослідження платформи автоматизованого розгортання ізольованих production-подібних середовищ, що забезпечить відтворюваність, масштабованість до реального production-середовища.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- провести аналіз відомих підходів та інструментів розгортання середовищ у межах DevOps;
- визначити вимоги до production-подібних середовищ та платформи їх автоматизованого розгортання;
- розробити архітектуру платформи з використанням контейнеризації;
- реалізувати прототип платформи автоматизованого розгортання;
- провести тестування працездатності та ефективності запропонованого рішення.

Наукова новизна одержаних результатів полягає у розробці підходу до автоматизованого розгортання ізольованих production-подібних середовищ на основі контейнеризації, що забезпечить масштабованість, продуктивність та стабільність платформи.

Отримані результати можуть бути використані у процесах розробки, тестування, навчання та експериментальних досліджень програмних систем.

Кваліфікаційна робота складається з вступу, трьох розділів, загальних висновків, списку використаних джерел та додатку.

1 ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗОВАНОГО РОЗГОРТАННЯ СЕРЕДОВИЩ

1.1 Класифікація середовищ розробки, тестування та експлуатації

В процесі розробки життєвого циклу (ЖЦ) програмного забезпечення використовується декілька типів середовищ, кожне з яких має власне призначення, вимоги та рівень наближеності до кінцевого production-середовища [2]. Класифікація середовищ дозволить впорядкувати процеси розробки, тестування та експлуатації будь-яких програмних систем, а також підвищити якість та надійність програмного застосунку. Серед них можливо виділити:

- середовище розробки (Development environment), яке призначене для написання, налагодження та первинного тестування програмного коду. Даний рівень характеризується частими змінами конфігурацій, спрощеними налаштуваннями безпеки та можливістю швидкого перезапуску сервісів. Тут, зазвичай, кожен розробник виконує завдання самостійно та може істотно відрізнятися від production-середовища;
- середовище інтеграції (Integration environment), яке використовується для об’єднання окремих компонентів системи та перевірки їхньої сумісності. В цьому середовищі здійснюється інтеграційне тестування, перевіряється коректність взаємодії сервісів, API та зовнішніх залежностей;
- середовище тестування (Testing / QA environment), яке призначене для проведення функціонального, навантажувального та регресійного тестування. Конфігурація даного середовища наближена до production, проте допускається використання тестових даних та

інструментів моніторингу, логування, які орієнтовані на виявлення помилок;

- Staging-середовище (Pre-production, Staging environment), який є максимально наближеним до production та використовується у фінальній перевірці системи перед розгортанням для реальної експлуатації. У staging-середовищі застосовуються ті ж самі конфігурації, версії програмного забезпечення та мережеві політики, що і в production;
- Production-середовище (Production environment), яке призначене для реальної експлуатації програмного продукту кінцевими користувачами. Тут, висуваються найвищі вимоги до стабільності, безпеки, масштабованості та відмовостійкості. Будь-які зміни в середовищі проходять ретельну перевірку на попередніх етапах;
- Production-подібні середовища (Production-like environments). Використовуються для моделювання умов реальної експлуатації з метою тестування, навчання та досліджень. Дані середовища відтворюють архітектуру та конфігурацію production, але є ізольованими та безпечними для експериментів, що робить їх особливо актуальними у сучасних DevOps-практиках (див. табл. 1.1).

Таблиця 1.1 – Класифікація середовищ розробки та експлуатації

Тип середовища	**Призначення**	**Основні характеристики**	**Рівень близькості до production**
Development	Розробка та налагодження коду	Часті зміни, спрощені налаштування, локальне використання	Низький
Integration	Перевірка взаємодії компонентів	Інтеграційне тестування, перевірка API та сервісів	Низький-середній
Testing (QA)	Функціональне та регресійне тестування	Тестові дані, розширене логування, контроль якості	Середній
Staging	Фінальна перевірка перед релізом	Конфігурація, ідентична production, обмежений доступ	Високий

Продовження таблиці 1.1

Production	Реальна експлуатація системи	Високі вимоги до стабільності, безпеки та масштабованості	Максимальний
Production-подібне	Моделювання реальних умов	Ізольованість, автоматизоване розгортання, відтворюваність	Високий

Отже, production-подібні середовища займають проміжне місце між рівнями staging та production. Вони поєднують в собі високий рівень наближеності до реальної експлуатації з безпекою та ізольованістю. Це робить їх ефективним інструментом для тестування, навчання та дослідження, а також обґрунтовує актуальність автоматизованого підходу щодо їх розгортання.

1.2 Вимоги, характеристики та роль production-подібних середовищ у життєвому циклі ПЗ

Production-подібне середовище представляє ізольоване середовище виконання, конфігурація якого відтворює архітектуру, програмні компоненти, мережеві налаштування та параметри безпеки production-системи, але не призначене для обслуговування реальних користувачів. Дані середовища застосовуються на етапах тестування, передрелізної перевірки, навчання та в дослідженнях. Розглянемо основні вимоги до production-подібних середовищ.

Серед них можна виділити:

- максимальну відповідність до production-середовища, що забезпечує коректну перевірку поведінки системи в умовах, наближених до реальної експлуатації;
- ізольованість, яка гарантуватиме безпечне виконання тестів та експериментів без впливу на production-систему;

- відтворювання, яке досягатиметься шляхом автоматизованого розгортання середовищ на основі декларативних описів інфраструктури;
- масштабування, яке необхідна для моделювання різних сценаріїв навантаження;
- керування, яке передбачає можливість автоматизованого створення, оновлення та видалення середовищ [3].

Production-подібні середовища можна охарактеризувати використанням сучасних підходів DevOps, контейнеризації та Infrastructure as Code. Вони дозволяють швидко створити ізольовані копії системи з контрольованими параметрами доступу та безпеки. Також важливою характеристикою середовищ є прозорість конфігурацій, яка забезпечує можливість до аналізу, аудиту та повторного використання середовищ.

Крім того, production-подібні середовища можуть підтримувати інтеграцію з CI/CD-конвеєрами, системами моніторингу та логування, що дозволить оцінювати стабільність і продуктивність програмного забезпечення ще до його розгортання у production.

У життєвому циклі програмного забезпечення production-подібні середовища виконують важливу роль проміжної ланки між етапами розробки, тестування та експлуатації. Так як вони дозволяють виявляти помилки, проблеми продуктивності та вразливості безпеки на ранніх етапах – це зменшує витрати на їх усунення.

Застосування production-подібних середовищ сприяє підвищенню якості програмного забезпечення, скороченню часу виходу продукту на ринок та зменшенню кількості збоїв у production. Такі середовища є ефективним інструментом для навчання персоналу, відпрацювання аварійних сценаріїв та проведення наукових і експериментальних досліджень [4].

1.3 Контейнеризація як основа ізоляції середовищ

Велика кількість програмних систем характеризуються високою складністю, модульною архітектурою та залежністю від значної кількості бібліотек і сервісів. За таких умов особливої актуальності набуває проблема ізоляції середовищ виконання, яка безпосередньо впливає на стабільність, безпеку та відтворюваність програмного забезпечення. Одним із ключових технологічних підходів до вирішення даної проблеми є контейнеризація.

Контейнеризація – це технологія віртуалізації на рівні операційної системи, яка дозволяє запустити програмні застосунки в ізольованих середовищах (контейнерах), які спільно використовують ядро операційної системи, проте мають власний набір бібліотек, залежностей і конфігурацій [5]. На відміну від повноцінної віртуалізації, контейнеризація забезпечує менші накладні витрати та швидке розгортання середовищ (див. рис.1.1).



Рисунок 1.1 – Принцип контейнеризації

Основними механізмами ізоляції контейнерів є простори імен (namespaces) та контрольні групи (cgroups), які забезпечують розмежування процесів, файлових систем, мережевих інтерфейсів і ресурсів між контейнерами. Таким чином, можна запустити декілька незалежних середовищ на одному фізичному або віртуальному сервері без взаємного впливу.

Контейнеризація забезпечує високий рівень відтворюваності середовищ, так як конфігурація контейнера описується у вигляді декларативного файлу, який містить повний перелік необхідних компонентів для запуску застосунку. Такий підхід мінімізує ризики, пов’язані з відмінностями між середовищами розробки, тестування та експлуатації.

Перевагою контейнеризації є можливість швидкого створення та видалення ізольованих середовищ, що особливо актуально для production-подібних середовищ. Вона дозволяє автоматизувати процеси тестування, перевірки безпеки, навчання та експериментальних досліджень без впливу на реальне production-середовище.

В DevOps-практиці контейнеризація є базовою технологією, яка забезпечує принцип «один раз зібрано – всюди запущено» (build once, run anywhere)[6]. Контейнери легко інтегруються з CI/CD-конвеєрами, системами оркестрації та інструментами Infrastructure as Code, що створює основу для автоматизованого керування життєвим циклом середовищ.

Розглянемо порівняльну схему архітектури контейнерів та віртуальних машин з принципами ізоляції середовищ ПЗ (див. рис 1.2):

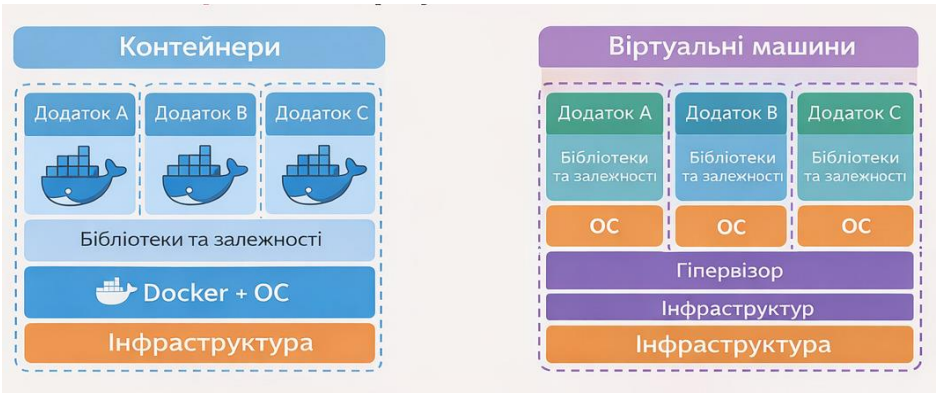


Рисунок 1.2 – Порівняння роботи контейнерів та віртуальних машин

У віртуальних машинах кожен застосунок розгортається у власному ізольованому середовищі, який включає повноцінну гостьову операційну систему, бібліотеки та залежності. Управління такими середовищами здійснюється за допомогою гіпервізора, який працює поверх апаратної

інфраструктури. Даний підхід забезпечує високий рівень ізоляції, проте потребує значних обчислювальних ресурсів і характеризується тривалим часом запуску [7].

Але контейнерна архітектура, на відміну від віртуалізації, базується на віртуалізації на рівні операційної системи. Контейнери спільно використовують ядро операційної системи хосту, але мають ізольовані файлові системи, процеси та мережеві налаштування. Кожен контейнер містить лише необхідні бібліотеки та залежності для роботи конкретного застосунку, це значно зменшує накладні витрати та прискорює розгортання середовищ.

Контейнеризація дозволяє розміщувати більшу кількість ізольованих середовищ на одній інфраструктурі за рахунок ефективнішого використання ресурсів. Водночас віртуальні машини забезпечують більш жорстку ізоляцію завдяки використанню окремої операційної системи для кожного середовища.

Отже, зі схеми видно, що доцільно використовувати контейнеризацію як основу для побудови ізольованих production-подібних середовищ, так як вона поєднує достатній рівень ізоляції з високою ефективністю використання ресурсів та швидкістю розгортання.

Підсумовуючи, можна сказати, що контейнеризація виступає фундаментальною технологією для побудови ізольованих production-подібних середовищ, забезпечуючи ефективну ізоляцію, відтворюваність та масштабованість, вона є необхідною умовою для розробки платформи автоматизованого розгортання середовищ у межах даної кваліфікаційної роботи.

1.4 Оркестрація контейнерів та керування інфраструктурою

Зі зростанням контейнеризованих застосунків та їх складності архітектури, виникає потреба у централізованому керуванні контейнерами, їх життєвим циклом та інфраструктурними ресурсами. Для вирішення задачі

використовуються системи оркестрації контейнерів, які забезпечують автоматизоване розгортання, масштабування та підтримку працездатності програмних систем.

Оркестрація контейнерів – це процес автоматизації об'єднання контейнерів у мережу та управління ними, що дозволяє розгортати додатки в потрібному масштабі. Контейнеризація пов'язує код програми з усіма файлами та бібліотеками, необхідними для роботи у будь-якій інфраструктурі. Архітектура мікросервісів може містити сотні або навіть тисячі контейнерів у міру зростання та ускладнення програм [8]. Інструменти оркестрації контейнерів покликані спростити управління інфраструктурою контейнерів за рахунок автоматизації всього їх життєвого циклу – від виділення ресурсів та планування до розгортання та видалення. Організації можуть отримати вигоду з масштабної контейнеризації без додаткових витрат на обслуговування. Контейнери стали стандартною обчислювальною одиницею для програм, оптимізованих під хмару. Постачальники хмарних послуг пропонують інстанси віртуальних серверів для виконання всіх видів обчислювальних робочих навантажень, які ідеально підходять для робочих навантажень на основі контейнерів. Єдина вимога для запуску контейнерів полягає в тому, щоб на сервері була запущена служба контейнеризації, така як Docker.

Docker – це інструмент із відкритим вихідним кодом для пакування програмного забезпечення та пов'язаних з ним бібліотек, системних інструментів, коду та середовища виконання у контейнер [9]. Це легке рішення для запуску кількох контейнерів та управління ними на одному інстансі сервера, але масштабування ускладнюється (див. рис.1.3).

До появи платформ оркестрації контейнерів організації використовували складні сценарії, щоб керувати розгортанням, плануванням та видаленням контейнерів на кількох пристроях. Збереження цих сценаріїв створювало складнощі, такі як контроль версій та масштабування налаштувань. Оркестрація контейнерів автоматизує та усуває їх, вирішуючи проблеми, пов'язані з ручним керуванням.

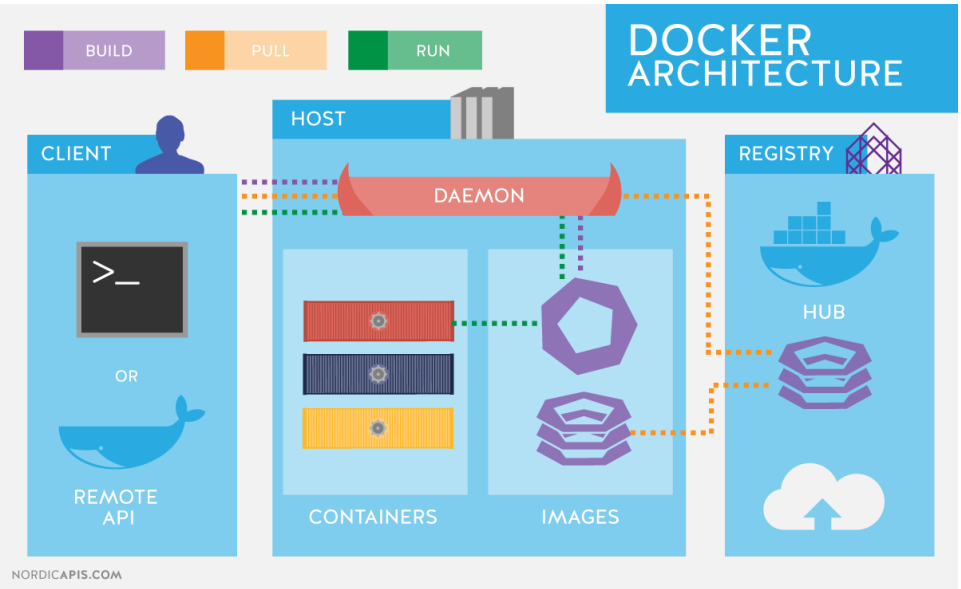


Рисунок 1.3 – Архітектура Docker

Розглянемо приклади використання оркестрації контейнерів. Вони необхідні, коли:

– керуємо контейнерами та масштабуємо їх у кількох інстансах;
– запускаємо безліч різних контейнерних програм;
– запускаємо різні версії додатків (наприклад, тестування та виробництво безперервної інтеграції та доставки (CI/CD)) одночасно;
– забезпечуємо безперервність служби програм у разі збою сервера, запустивши кілька інстансів (реплік) контейнера;
– запускаємо кілька інстансів програми у різних географічних регіонах;
– максимально використовуємо декілька інстансів серверів для складання бюджету;
– запускаємо великі контейнерні програми, що складаються з тисяч різних мікросервісів.

Одна з найбільших переваг оркестрації контейнерів полягає в тому, що вона автоматизує масштабованість, доступність та продуктивність контейнерних додатків. Можна налаштувати інструменти оркестрації контейнерів для масштабування залежно від попиту, доступності мережі та

інфраструктурних обмежень. Рішення для оркестрації контейнерів дозволяє відстежувати продуктивність у контейнерній мережі та автоматично переналаштовувати контейнери, щоб досягти оптимальної продуктивності.

Контейнери – це автономні програми чи мікросервіси на базі Linux з усіма бібліотеками та функціями, необхідними для роботи практично на будь-якому типі машин. Оркестрація контейнерів здійснюється шляхом управління контейнерами групи екземплярів сервера (також називають вузлами). Група вузлів, у яких працюють взаємозалежні контейнери, називається кластером.

Для оркестрації контейнерів потрібне, по-перше, базове рішення щодо контейнеризації, що працює на кожному вузлі кластера – це Docker. На вузлах також має бути запущений інструмент оркестрації. Призначений головний вузол з площиною управління є контролером рішення по оркестрації. Адміністратор рішення використовує графічний інтерфейс або контролер командного рядку на головному вузлі управління та моніторингу інструменту оркестрації контейнерів [10].

Як тільки контейнери запущені у кластері, інструмент оркестрації керує загальним станом системи, забезпечуючи її збереження у стані продуктивності. (див. рис.1.4)

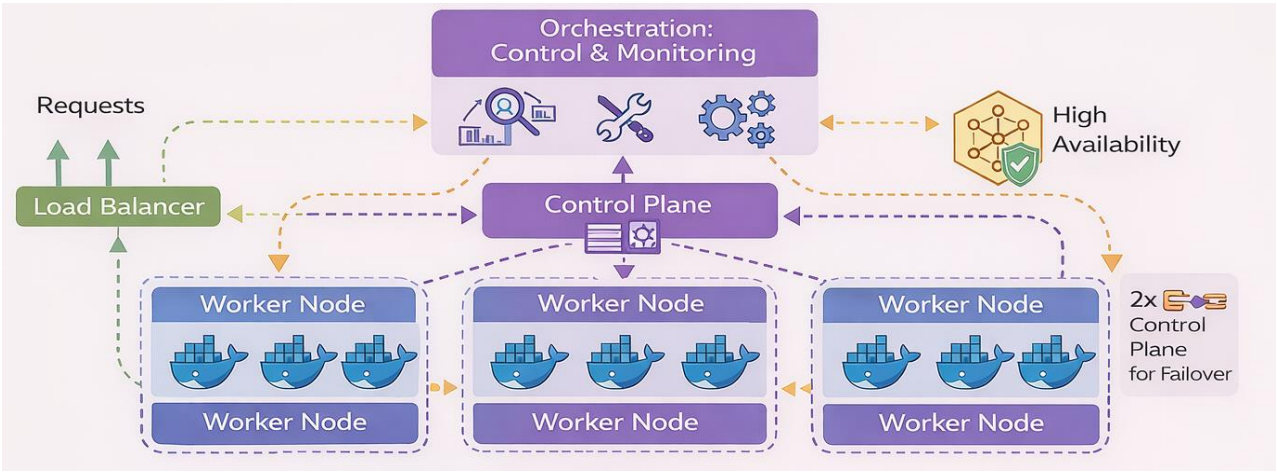


Рисунок 1.4 – Оркестрація контейнерів у кластері

Сюди можемо віднести:

- розподіл ресурсів між контейнерами;
- розгортання контейнерів на нових вузлах або видалення контейнерів;
- балансування навантаження трафіку до програми.

Оркестрація контейнерів керує життєвим циклом контейнерів для оптимізації та захисту великих, складних багатоконтейнерних робочих навантажень та середовищ. Вона може керувати такою кількістю контейнерних програм, які потрібні організації. Використання кількох головних вузлів для забезпечення високої доступності та стійкості до відмови типово для більш високих вимог організації [11].

У даному розділі розглянуто теоретичні основи автоматизованого розгортання середовищ програмного забезпечення. Проаналізовано класифікацію середовищ розробки, тестування та експлуатації, а також визначено місце production-подібних середовищ у життєвому циклі програмного забезпечення.

Використання production-подібних середовищ дозволяє суттєво зменшити ризики, пов’язані з розбіжностями між середовищами розробки та реальної експлуатації, і підвищити якість програмного забезпечення. Основними вимогами до таких середовищ є ізольованість, відтворюваність, масштабованість та максимальна наближеність до production-середовища.

Отже, можна сказати, що контейнеризація є ефективним засобом ізоляції середовищ виконання, забезпечуючи швидке розгортання, ефективне використання ресурсів і уніфікацію конфігурацій. Застосування оркестрації контейнерів та підходів керування інфраструктурою дозволяє автоматизувати життєвий цикл середовищ, підвищити їх надійність та керованість.

Теоретичний аналіз показав актуальність розробки платформи автоматизованого розгортання ізольованих production-подібних середовищ та слугує основою для подальшого проєктування архітектури й реалізації програмного рішення.

2 ПРОЄКТУВАННЯ ПЛАТФОРМИ АВТОМАТИЗОВАНОГО РОЗГОРТАННЯ

2.1 Аналіз існуючих рішень та платформ

Стрімкий розвиток сучасних програмних систем, мікросервісних архітектур та DevOps-практик зумовило потребу у швидкому, відтворюваному та керованому розгортанні середовищ виконання. Особливої актуальності набуло створення ізольованих production-подібних середовищ, які дозволяють проводити тестування, експерименти та навчальні дослідження без ризику для реальної експлуатації програмного забезпечення.

На даний час існує невелика кількість платформ та інструментів, що частково вирішують задачу автоматизованого розгортання середовищ. Проте, більшість з них орієнтовані на окремі аспекти життєвого циклу ПЗ – розробку, хмарне розгортання або керування інфраструктурою.

Найбільш відомі з них: Coder, Heroku та Terraform. Розглянемо їх можливості, переваги і обмеження при створенні ізольованих production-подібних середовищ.

Coder являється платформою для створення та управління ізольованими середовищами розробки, які запускаються у вигляді контейнерів або віртуальних машин у хмарній чи локальній інфраструктурі. Дана платформа централізує робочі середовища розробників та забезпечує їх повне відтворення.

Середовища у Coder описуються декларативно та можуть бути інтегровані з системами керування версіями, такими як Git. Вона дозволяє забезпечити єдині стандарти конфігурації, що мінімізує проблему «працює на моєму комп’ютері». Платформа також підтримує механізми контролю доступу та ізоляції між користувачами [12] (див. рис.2.1).

Разом з тим, Coder орієнтований переважно на development-середовища та не розрахований на повноцінне відтворення production-навантажень,

складних мережевих конфігурацій або сценаріїв експлуатації. Це обмежує його застосування для задач, пов’язаних із тестуванням або моделюванням реального production-середовища. Серед переваг можна виділити: централізоване управління середовищами розробки, відтворення конфігурацій та зручний веб-інтерфейс. Проте наявні і недоліки. Такі як обмежена підтримка production-подібних сценаріїв та фокус на середовищах розробки, а не експлуатації.

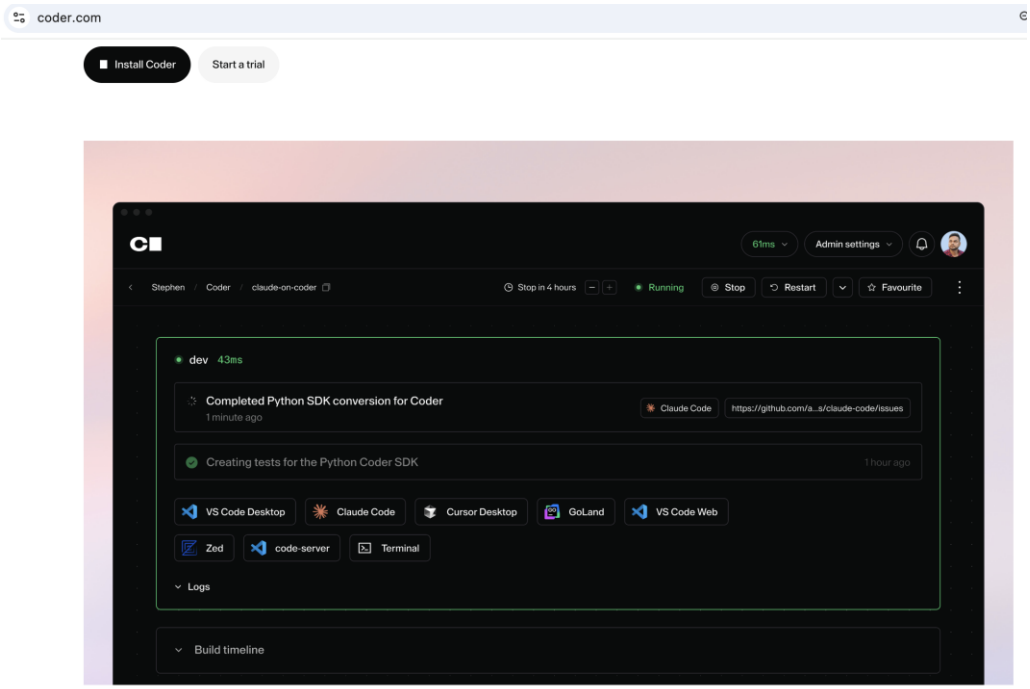


Рисунок 2.1 – Візуальний вигляд платформи Coder

Heroku – хмарна платформа класу PaaS, яка забезпечує повністю кероване середовище для розгортання та масштабування веб-застосунків. Платформа приховує інфраструктурні деталі, надає розробникам простий інтерфейс для деплойменту додатків (див. рис.2.2).

Heroku автоматизує процеси масштабування, балансування навантаження та оновлення застосунків, цим вона приваблива для швидкого запуску проєктів. Наявна інтеграція з CI/CD дозволяє реалізувати безперервну доставку програмного забезпечення [13].

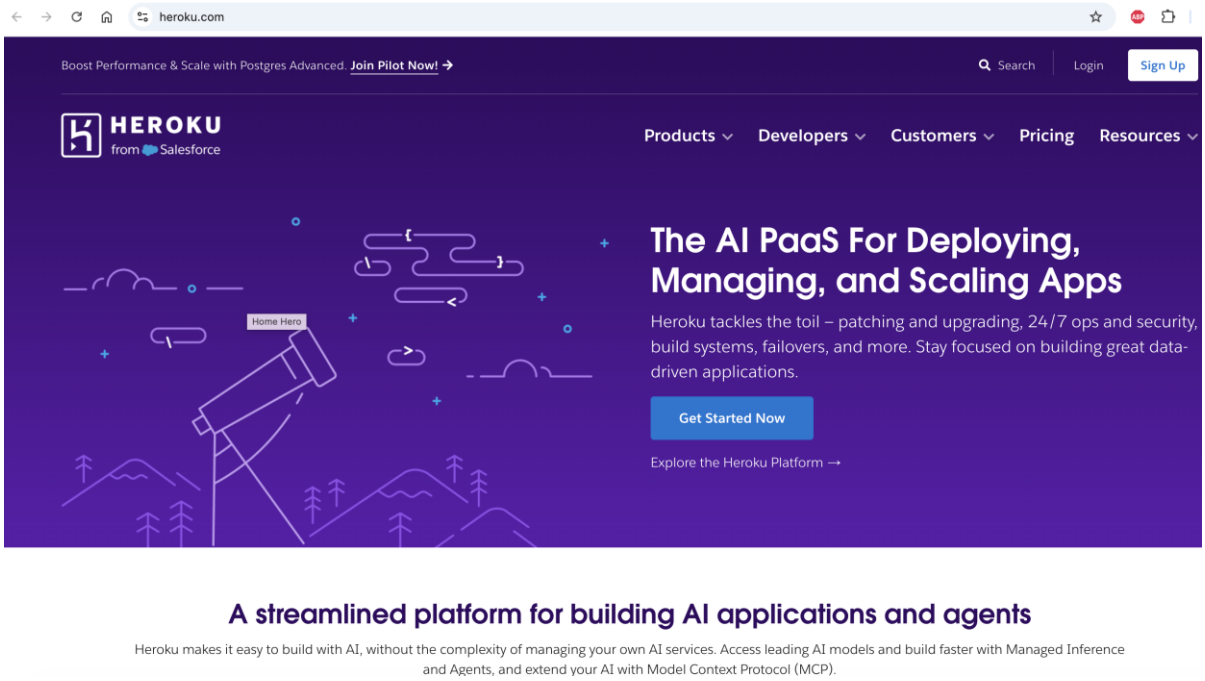


Рисунок 2.2 – Візуальний вигляд платформи Heroku

Проте, високий рівень абстракції обмежує можливість повного контролю над середовищем. Ускладнює створення середовищ, ідентичних production з точки зору мережевих налаштувань, політик безпеки та ресурсних обмежень. Таким чином, Heroku підходить для швидкого деплойменту, але не для точного моделювання production-подібних середовищ. Недоліками є:

– обмежений контроль над інфраструктурою;
– залежність від постачальника платформи;
– складність відтворення production-середовища поза платформою.

Terraform являється інструментом Infrastructure as Code, призначеним для опису, створення та керування інфраструктурою у вигляді конфігураційних файлів. Дана платформа дозволяє автоматизувати розгортання серверів, мереж, сховищ даних та інших інфраструктурних компонентів.

Основною перевагою Terraform є висока відтворюваність середовищ та можливість використання єдиного підходу для різних хмарних і локальних платформ. Інструмент широко застосовується у поєднанні з контейнеризацією та системами оркестрації (див. рис. 2.3).

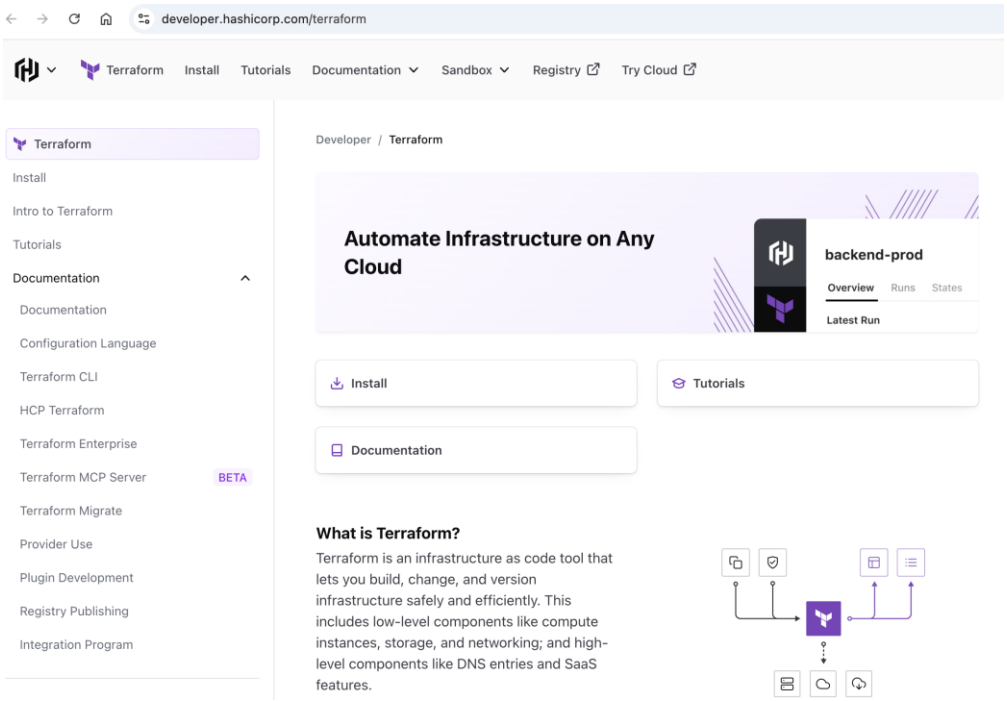


Рисунок 2.3 – Візуальний вигляд платформи Terraform

Проте, Terraform не надає власного механізму виконання застосунків або керування контейнерами, що потребує інтеграції з додатковими інструментами. Тому його доцільно розглядати як фундамент для побудови платформи, а не як завершене рішення. Недоліками виступають: відсутність вбудованих механізмів оркестрації, потреба у додаткових компонентах та підвищена складність для початкового налаштування [14].

Але основний недолік проаналізованих платформ є їх платна основа.

2.2 Вимоги до платформи автоматизованого розгортання ізольованих середовищ

Вимоги до платформи автоматизованого розгортання ізольованих production подібних середовищ сформувалися на основі аналізу проблем сучасних підходів до організації середовищ розробки та необхідності забезпечення ефективної, безпечної та масштабованої системи для команд розробників.

Тому вимоги до даної розробки можна поділити на: функціональні та нефункціональні, кожна з яких має критичне значення для успішного функціонування платформи та можна виділити у підвимоги. Розглянемо основні з них.

2.2.1 Нефункціональні вимоги

Нефункціональні вимоги визначають, як платформа повинна працювати, які характеристики продуктивності, надійності, безпеки та масштабованості вона повинна забезпечувати. Дані вимоги дуже важливі для забезпечення стабільної роботи платформи в корпоративному середовищі. Розглянемо їх:

- вимоги до продуктивності – платформа повинна забезпечити швидке виконання операцій без блокування користувацького інтерфейсу. Запуск задач у черзі повинний відбуватися з мінімальною затримкою (менше 1 секунди від моменту ініціації до початку виконання виконувачем). Доставка broadcast-подій до користувацького інтерфейсу повинна відбуватися з затримкою менше 2 секунд від моменту відправлення події виконувачем;
- вимоги до масштабованості – розробка повинна забезпечувати горизонтальне масштабування для обробки зростаючого навантаження. Система черг підтримувати додавання нових виконувачів для обробки більшої кількості одночасних операцій без зміни конфігурації або зупинки системи;
- вимоги до безпеки – безпека платформи забезпечується через кілька рівнів захисту. Авторизація користувачів здійснюється через Laravel Auth middleware з підтримкою сесій та токенів. Приватні broadcast-канали повинні перевіряти права доступу користувача перед підпискою на канал. Ізоляція середовищ у Kubernetes забезпечується через окремі namespace для кожного середовища, що запобігає конфліктам ресурсів та забезпечує мережеву сегментацію. RBAC

(Role-Based Access Control) у Kubernetes налаштовується для контролю доступу до подів, логів та інших ресурсів;

- вимоги до інфраструктури та залежностей – платформа залежить від наявності певної інфраструктури та сервісів, які налаштовані та доступні для її функціонування;
- вимоги до кластеру Kubernetes – кластер Kubernetes налаштований та доступний для розгортання середовищ. Кластер має достатні ресурси (CPU, пам’ять, диск) для підтримки необхідної кількості середовищ. Ingress налаштований для забезпечення доступу до середовищ через HTTP/HTTPS;
- вимоги до Redis –він має бути доступний та налаштований для використання як брокер черг та каналів broadcast. Redis забезпечує достатню продуктивність для обробки навантаження та має налаштування персистентності для забезпечення збереження даних при перезапуску.

2.2.2 Функціональні вимоги

Функціональні вимоги визначають, що саме повинна виконувати платформа та які операції вона підтримує, можливості надає користувачам. Дані вимоги базуються на аналізі потреб команд розробників та вимогах до автоматизації процесів створення та управління середовищами. Розглянемо їх детально:

- вимоги до створення та видалення середовищ – платформа реалізує автоматизоване створення ізольованих середовищ розробки на основі готових шаблонів (template). Кожне середовище повністю ізольоване від інших середовищ, що забезпечуються через використання окремих namespace у Kubernetes та мережевої сегментації. Вона підтримує множину шаблонів для різних типів проєктів: веб-додатки на PHP (LAMP/LEMP стеки), Node.js, Python, Java та інші технології. Процес

створення середовища максимально простий для користувача, достатньо вибрати шаблон, вказати назву середовища та натиснути кнопку створення. Розробка автоматично створює необхідні ресурси в Kubernetes, де Pod з контейнерами для веб-сервера, бази даних, кешу та допоміжних сервісів, Ingress для доступу через HTTP/HTTPS, ConfigMap та Secrets для зберігання конфігурацій та чутливих даних;

- вимоги до керування сервісами середовища – розробка надає можливість керування окремими сервісами всередині середовища: запуск, зупинка, перезапуск сервісів. Наприклад, для середовищ на базі PHP платформа повинна підтримувати керування веб-сервером (Apache/Nginx), базою даних (MySQL/MariaDB), кешем (Redis), а також можливість перемикання версій PHP;
- вимоги до розгортання проєктів – забезпечує автоматизоване розгортання проєктів у середовищах розробки. Процес розгортання включає копіювання коду з Git-репозиторію, встановлення залежностей, виконання міграцій бази даних, налаштування конфігурацій та перезапуск сервісів. Розгортання виконується асинхронно через систему черг, щоб не блокувати веб-інтерфейс та дозволити користувачу продовжувати роботу з платформою;
- вимоги до перегляду логів – забезпечує можливість перегляду логів виконання операцій для діагностики проблем та відстеження виконання команд. Логи зберігаються у файлах та доступні через веб-інтерфейс з можливістю фільтрації та пошуку. Вони містять інформацію про час виконання операцій, виведення команд, помилки та попередження;
- вимоги до інтеграції з інструментами розробки – забезпечує on-demand доступ до інструментів розробки, таких як IDE (наприклад, PhpStorm через code-server або інші веб-IDE), phpMyAdmin для управління базами даних, та інші допоміжні інструменти. Доступ до

цих інструментів надається через веб-інтерфейс з можливістю вмикання/вимикання за потреби;

– вимоги до авторизації та розмежування доступів – забезпечує авторизацію користувачів та розмежування доступів до середовищ та операцій. Користувачі мають можливість працювати тільки з середовищами, до яких вони мають права доступу. Платформа підтримує ролі користувачів (наприклад, розробник, адміністратор) з різними рівнями доступу.

2.3 Архітектура платформи та опис основних компонентів

Хмарна платформа для автоматизації розгортання ізольованих середовищ розробки побудована на принципах мікросервісної архітектури та поділу виконання роботи між різними компонентами. Система складається з трьох основних рівнів: презентаційного (frontend), прикладного (backend) та інфраструктурного (Kubernetes, Redis, Echo Server) (див. рис.2.4).

Презентаційний рівень відповідає за взаємодію з користувачем. Він включає веб-інтерфейс, побудований на базі Laravel Blade та JavaScript, який забезпечує відображення інформації про середовища, керування сервісами, запуск операцій та відстеження прогресу в реальному часі.

Прикладний рівень реалізує бізнес-логіку системи. Він включає контролери Laravel для обробки HTTP-запитів, систему черг для виконання довготривалих операцій, механізм broadcast-подій для сповіщення користувачів та сервіси для взаємодії з Kubernetes API.

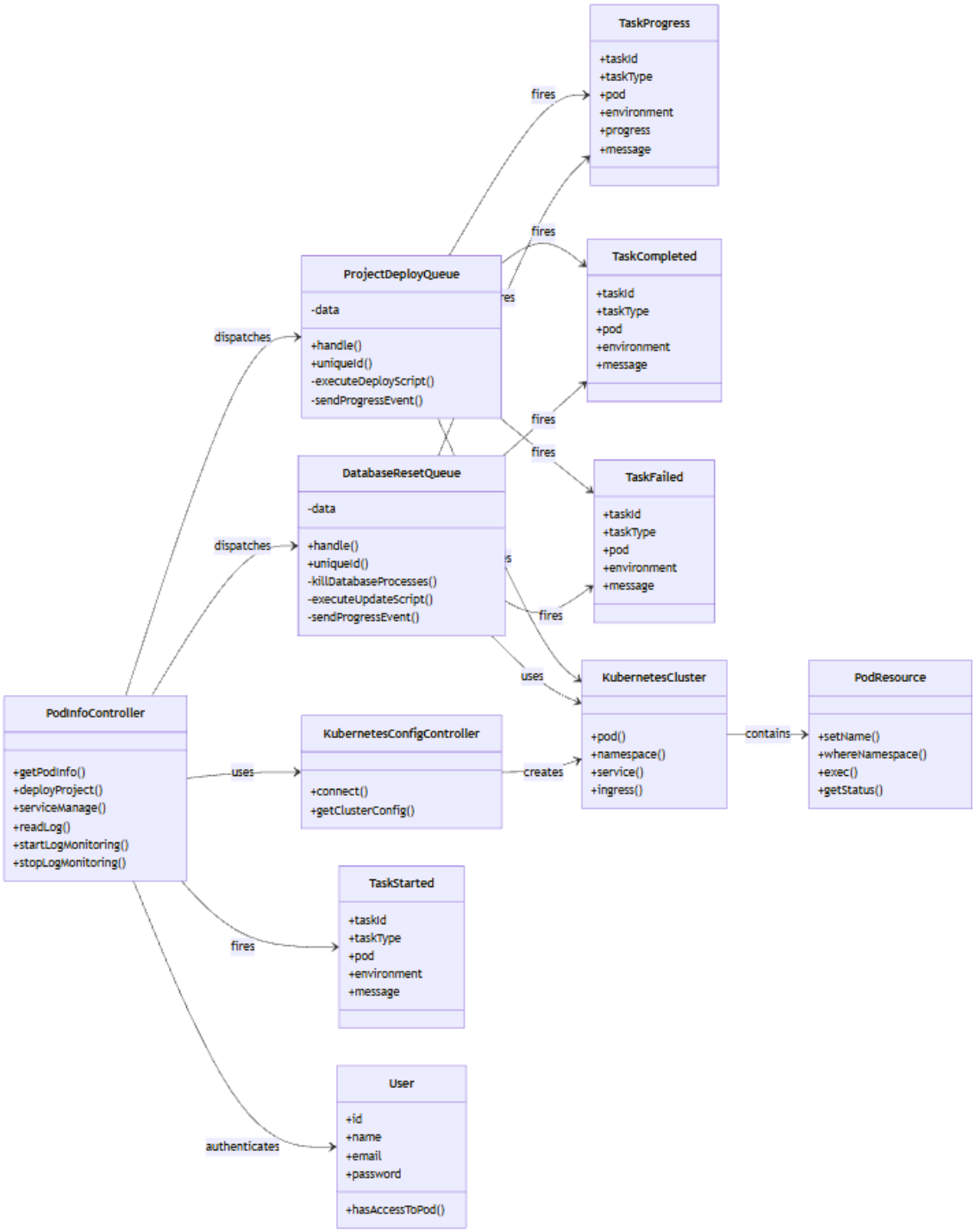


Рисунок 2.4 – UML діаграма роботи системи

Інфраструктурний рівень забезпечує базові сервіси для роботи системи: Kubernetes для оркестрації контейнерів, Redis для черг та broadcast-каналів, Laravel Echo Server для WebSocket-з’єднань. Розглянемо основні компоненти:

– PodInfoController є основним контролером системи, який обробляє HTTP-запити від користувачів та координує виконання операцій над середовищами розробки;

- KubernetesConfigController – це сервісний клас, який забезпечує підключення до кластеру Kubernetes та налаштування конфігурації;
- ProjectDeployQueue – це клас Laravel Job, який інкапсулює логіку розгортання проєктів у середовищах розробки;
- DatabaseResetQueue – це клас Laravel Job, який інкапсулює логіку скидання баз даних у середовищах розробки;
- TaskStarted, TaskProgress, TaskCompleted, TaskFailed. Дані класи реалізують Laravel Events, які транслюються через broadcast-механізм для сповіщення користувачів про стан виконання задач;
- User – є моделлю Laravel, яка представляє користувача системи та забезпечує авторизацію та розмежування доступів;
- KubernetesCluster та PodResource. Дані класи надаються бібліотекою renoki-co/php-k8s для роботи з Kubernetes API;
- PodResource забезпечує операції над конкретним pod.

Взаємодія компонентів системи відбувається за наступною схемою:

- користувач ініціює операцію через веб-інтерфейс, що надсилає HTTP-запит до PodInfoController;
- PodInfoController обробляє запит: перевіряє авторизацію користувача, формує дані для Job, відправляє подію TaskStarted для сповіщення користувача та ставить Job у чергу Redis;
- Worker отримує Job з черги та викликає метод handle() відповідного Job (ProjectDeployQueue або DatabaseResetQueue);
- Job виконує операцію: підключається до pod через KubernetesCluster, виконує необхідні скрипти та відправляє події прогресу через broadcast-механізм;
- події транслюються через Redis у broadcast-канал pod.{pod}, де {pod} – ідентифікатор середовища;
- Laravel Echo Server отримує події з Redis та доставляє їх клієнтам через WebSocket-з’єднання;

– клієнтський JavaScript отримує події та оновлює інтерфейс в реальному часі, відображаючи прогрес виконання операції.

Така архітектура забезпечує неблокуючу обробку довготривалих операцій, прозорість процесу виконання для користувача та можливість масштабування системи, обробки великої кількості одночасних операцій.

2.4 Модель безпеки та ізоляції production-подібних середовищ

В проєктуванні платформи автоматизованого розгортання важливим аспектом являється забезпечення належного рівня безпеки та ізоляції середовищ. Production-подібні середовища мають максимально відтворювати умови реального production-оточення, гарантуючи захист даних, стабільність системи та незалежність роботи окремих середовищ.

Модель безпеки платформи базується на принципах контейнеризації, ізоляції ресурсів, контрольованого доступу. Контейнери забезпечують логічне відокремлення додатків та їх залежностей, це дозволяє запускати декілька незалежних середовищ на одній фізичній або віртуальній інфраструктурі. Кожне середовище містить власний набір контейнерів, мережевих налаштувань та конфігурацій, що запобігає взаємному впливу між різними середовищами.

Ізоляція мережевих взаємодій є ще одним важливим компонентом моделі безпеки. Для кожного середовища створюється окрема віртуальна мережа, у межах якої взаємодіють контейнерні сервіси.

Модель безпеки забезпечує:

– ізоляцію середовищ — кожне середовище розгортається в окремому Kubernetes namespace;
– контроль доступу — користувачі бачать і керують лише своїми середовищами;
– захист секретів — SSH-ключі, паролі та токени зберігаються безпечно;

– обмеження реєстрації — доступ лише для дозволених доменів email.

Кожне середовище створюється в окремому namespace з унікальною назвою: `<project-name>-<username>` (див.рис.2.5).

```
public function submit(Request $request) {
    $payload = $request->all();
    $projectName = $payload['project-name'];
    // Назва namespace включає username для ізоляції
    $namespaceName = $projectName . "-" . Auth::user()->username;
    $namespace = $cluster->namespace()->setName($namespaceName);
    $namespace->create();
    // Всі ресурси створюються в даному namespace
    $resources = $cluster->fromTemplatedYamlFile(
        storage_path('k8s_templates/' . $projectTemplate . '.yml'),
        $vars
    );
```

Рисунок 2.5 – Створення середовища

При цьому системні namespace виключені з відображення та модифікації (див.рис.2.6).

```
public function getEnvironments() {
 $restricted_services = array(
     'calico-apiserver',
     'calico-system',
     'cert-manager',
     'default',
     'ingress-nginx',
     'kube-node-lease',
     'kube-public',
     'kube-system',
     'lens-metrics',
     'tigera-operator'
 );
 // Фільтрація: виключаємо системні namespace
 foreach ($data as $key => $ns) {
     if (!in_array($ns['metadata']['name'], $restricted_services)) {
         // Показуємо тільки користувацькі середовища
```

Рисунок 2.6 – Захист системних namespace

Реєстрація дозволена лише для e-mail з певних доменів (наприклад, @absoluteweb.com). Користувач бачить лише свої середовища (namespace закінчується на -username), адміністратор – усі. Розроблено генерацію

унікальних паролів для середовищ, тобто кожне середовище отримує унікальний пароль, який не перетинається з іншими. Виконується перевірка referer для захисту від прямого доступу (див.рис.2.7).

```
public function environmentInfo($environment) {
    // Захист: доступ тільки зі списку середовищ
    if (request()->headers->get('referer') != route('environments')) {
        return redirect()->route('environments');
    }
    // Отримання інформації про pod-и в namespace
    $data = json_decode(($cluster->pod()->whereNamespace($environment)->get()), TRUE);
    return view('EnvironmentInfo')->with('data', $data);
}
```

Рисунок 2.7 – Перевірка referer

Дана модель безпеки забезпечує повну ізоляцію production-подібних середовищ та запобігає несанкціонованому доступу до ресурсів інших користувачів.

2.5 Проєктування автоматизованого рішення

Проєктування автоматизованого рішення на основі діаграм прецедентів та використання дозволяє чітко визначити функціональність системи, взаємодію користувачів з системою та детальну послідовність виконання операцій. Вони забезпечують правильне розуміння вимог, ефективну реалізацію та можливість тестування системи на відповідність вимогам. Діаграми служать основою для розробки детальних специфікацій, допомагають виявити потенційні проблеми на ранніх етапах проєктування.

2.5.1 Діаграма використання

Діаграма варіантів використання – це вихідне концептуальне подання або концептуальна модель системи в процесі її проектування і розробки.

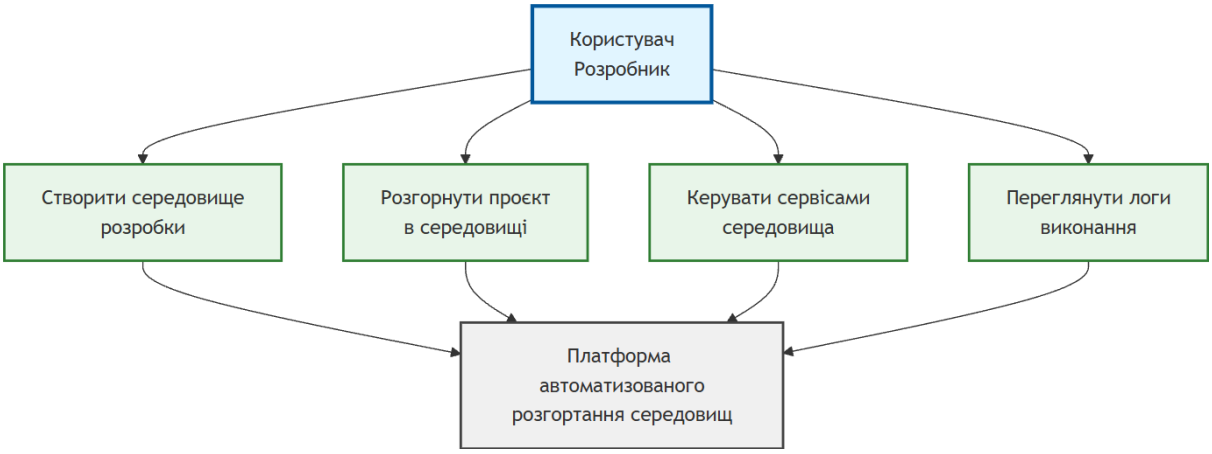


Рисунок 2.8 – Діаграма використання користувача

Дана платформа включає в себе модель користувача та адміністратора. Розглянемо модель користувача (див. рис.2.8):

– створити середовище розробки– користувач вибирає шаблон середовища та створює нове ізольоване середовище розробки з необхідними сервісами (веб-сервер, база даних, кеш);
– розгорнути проєкт в середовищі – користувач ініціює процес розгортання проєкту, який включає копіювання коду, встановлення залежностей, виконання міграцій бази даних;
– керувати сервісами середовища– користувач може запускати, зупиняти, перезапускати сервіси середовища, а також перемикати версії PHP;
– переглянути логи виконання– користувач отримує доступ до логів виконання операцій для діагностики проблем та відстеження виконання команд.

Робота моделі адміністратора (див. рис.2.9):

– керувати користувачами – адміністратор може створювати, редагувати та видаляти користувачів, налаштовувати права доступу до середовищ;

– моніторинг стану системи – адміністратор отримує інформацію про стан системи: навантаження на черги, стан воркерів, використання ресурсів кластера;
– видалити середовище– адміністратор може видаляти середовища з автоматичним очищенням всіх пов’язаних ресурсів.

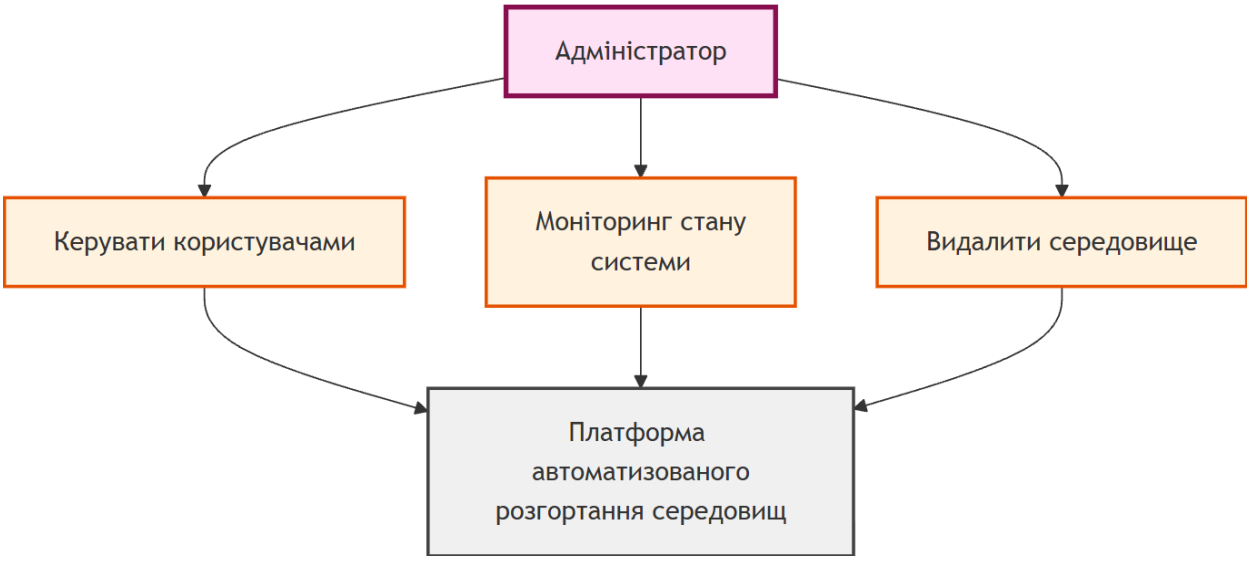


Рисунок 2.9 – Діаграма використання адміністратора

2.5.2 Діаграма прецедентів

Діаграма прецедентів відображає основні функції системи та взаємодію акторів (користувачів) з системою. Вона показує, що саме можна робити з платформою автоматизованого розгортання ізольованих середовищ. Розглянемо діаграму прецедентів платформи [15] (див. рис.2.10):

– авторизація користувача – система перевіряє, чи авторизований користувач та чи має він права на створення середовищ;
– вибір шаблону – користувач обирає шаблон середовища з доступних (LAMP, LEMP, Node.js, Python тощо);
– валідація шаблону – система перевіряє, чи існує обраний шаблон та чи він доступний для використання;

– створення namespace – система створює окремий namespace в Kubernetes для ізоляції ресурсів нового середовища;

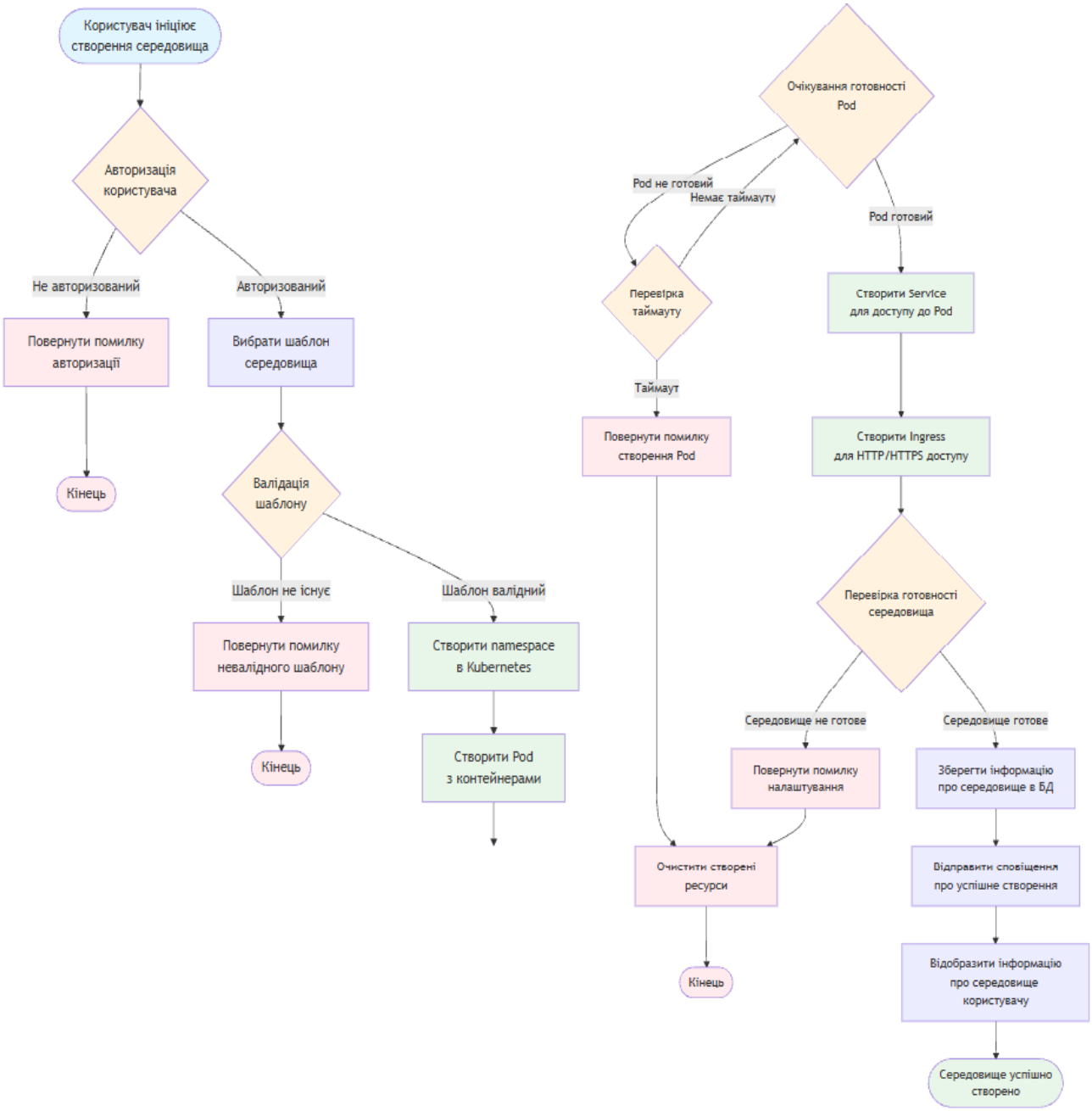


Рисунок 2.10 – Діаграма прецедентів

– створення Pod – система створює Pod з контейнерами згідно з обраним шаблоном;

- очікування готовності Pod – система очікує, поки Pod стане готовим до роботи. Якщо Pod не стає готовим протягом встановленого часу, система повертає помилку та очищає створені ресурси;
- створення Service– система створює Service для забезпечення стабільного мережевого доступу до Pod;
- створення Ingress – система створює Ingress для забезпечення HTTP/HTTPS доступу до середовища з можливістю налаштування TLS;
- перевірка готовності середовища– система виконує фінальну перевірку готовності середовища;
- збереження інформації про середовище – система зберігає інформацію про створене середовище в базі даних платформи: назва, namespace, URL, статус;
- відправлення сповіщення – система надсилає сповіщення користувачу про успішне створення середовища через веб-інтерфейс;
- відображення інформації користувачу – система відображає користувачу детальну інформацію про створене середовище: URL для доступу, статус сервісів.

На кожному етапі створення середовища система перевіряє успішність виконання операцій. У разі помилки система виконує очищення створених ресурсів (CleanupResources) для запобігання «залишкових» ресурсів в кластері та повертає користувачу детальну інформацію про помилку.

Отже, проаналізувавши існуючі рішення та провівши проєктування платформи, спираючись на алгоритм перейдемо до програмної реалізації системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПЛАТФОРМИ

3.1 Реалізація механізму автоматизованого розгортання середовищ

Реалізований програмний продукт має наступні вимоги та налаштування кластеру Kubernetes, зокрема апаратні вимоги:

- Master-вузол CPU: 8 ядер, Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz, RAM: 64 GB DDR4, диск SSD NVMe 500 GB, мережа: 1 Gbit/s (мінімум), статична IP-адреса, ОС Ubuntu Server LTS 24.04;
- Worker-вузли (для кожного з двох worker-вузлів): CPU: 64 ядра, AMD EPYC 7502P 32-Core Processor, RAM: 256 GB DDR4, диск: 8 TB NVMe SSD, мережа 1 Gbit/s (мінімум), статична IP-адреса, ОС (рекомендовано): Ubuntu Server LTS 22.04 / 20.04 (x86_64).

Програмні вимоги до кластеру наступні: kubeadm v1.24.3, kubelet v1.24.3, kubectl v1.24.3 та NTP-синхронізація часу на всіх вузлах.

Налаштування master-вузла за допомогою kubeadm:

а) встановити Docker/Containerd, kubeadm, kubelet, kubectl відповідної версії:

sudo apt install -y apt-transport-https ca-certificates curl docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt update

sudo apt install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl

б) ініціалізувати кластер:

*kubeadm init *

*--kubernetes-version=v1.24.3 *

*--pod-network-cidr=10.244.0.0/16 *

--apiserver-advertise-address=<IP master-вузла>

в) зберегти і налаштувати kubeconfig для адміністратора (копіювання /etc/kubernetes/admin.conf в $HOME/.kube/config);

г) розгорнути мережевий CNI-плагін:

kubectl create -f

https://raw.githubusercontent.com/projectcalico/calico/v3.31.3/manifests/tigera-operator.yaml

д) встановити Nginx Ingress для роутингу HTTP / HTTPS

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm repo update

helm install ingress-nginx ingress-nginx/ingress-nginx --create-namespace --namespace ingress-nginx

е) виконати команду, яка створить токен для підключення worker-вузлів: *kubeadm token create --print-join-command.*

Для підключення worker-вузлів необхідно:

- на кожному worker-вузлі встановити container runtime, kubeadm, kubelet, kubectl;
- виконати команду приєднання, згенеровану kubeadm init;
- перевірити статус вузлів через kubectl get nodes.

Додаткові налаштування кластеру для автоматизованого розгортання:

- налаштувати Ingress-контролер Nginx Ingress для доступу до застосунків ззовні;
- встановити cert-manager для автоматичного отримання/оновлення TLS-сертифікатів;
- kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.19.2/cert-manager.yaml;

– інтегрувати кластер з CI/CD (GitLab CI, GitHub Actions, Jenkins тощо) для автоматизованого деплою.

Вимоги та налаштування серверу під застосунок:

– апаратні характеристики серверу під Laravel: CPU: 16 ядер, AMD Ryzen 7 7700 8-Core Processor, RAM: 64 GB DDR4, диск: NVMe 500 GB, мережа: 1 Gbit/s (мінімум), статична IP-адреса, ОС (рекомендовано): Ubuntu Server LTS 22.04 / 20.04 (x86_64);

– програмні вимоги: PHP 8.3.6 (з розширеннями: mbstring, intl, pdo_mysql, bcmath, json, curl, openssl, tokenizer, xml, gd, zip та ін.), Web-сервер: Nginx, PHP-FPM, база даних MariaDB 10.11, Composer (для управління залежностями Laravel), Git (для деплою з репозиторію), Redis, Supervisor / systemd unit для воркерів черг Laravel (php artisan queue:work);

– вимоги до Laravel-застосунку: версія Laravel: 9.52.20, файл .env для кожного середовища (dev / stage / prod) із окремими параметрами DB_HOST, DB_DATABASE, DB_USERNAME, DB_PASSWORD, APP_ENV, APP_DEBUG, APP_URL, CACHE_DRIVER, QUEUE_CONNECTION, SESSION_DRIVER;

– налаштування черг і планувальника: Supervisor / systemd сервіс для queue:work.

Загальна структура налаштування веб-сервера має наступний вигляд (див.рис.3.1):

```
server {
    server_name your_domain;
    listen 80;
    root /path/to/your /laravel /public;
    index index.php index.htm index.html;

    location / {
        try_files $uri $uri/ /index.php?$args;
    }
    location ~ \.php(/|$) {
    try_files $uri $fastcgi_script_name =404;
    fastcgi_pass unix:/run/php/php8.3-fpm.sock;
    }
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
}
```

Рисунок 3.1 – Конфігураційні налаштування серверу

Для PHP-FPM (див.рис.3.2):

```
listen = /run/php/php8.3-fpm-cloud.sock
listen.owner = your_user
listen.group = your_user
pm = dynamic
pm.max_children = 20
pm.start_servers = 10
pm.min_spare_servers = 10
pm.max_spare_servers = 20
```

Рисунок 3.2 – PHP-FPM

Зв'язок між Laravel-продуктом і Kubernetes-класстером відбувається через kubernetes config file (який було отримано при ініціалізації кластеру), він має бути розташований у директорії storage, та вказаний у .env файлі. Доступ до файлу read only та обмежений у використанні тільки користувача Laravel.

3.2 Інтеграція з системами керування версіями та CI/CD

Якщо у вихідній системі немає готового функціоналу CI/CD, його можна додати за рахунок:

- власного webhook-ендпоінту платформи;
- CI/CD-пайплайна в GitLab / GitHub, який при пуші викликає цей webhook;
- Jobs-скриптів, які всередині платформи створюють ізольоване середовище, використовуючи назву проєкту, вибрану гілку (branch), шаблон середовища ('project-template'), версію PHP тощо.

Маршрут для webhook створення середовища (файл `routes/api.php`) має наступний програмний код (див.рис.3.3):

```php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\EnvironmentWebhookContro

Route::post('/ci/create-environment', [EnvironmentWeb
    ->name('ci-create-environment');
```

Рисунок 3.3 – Маршрут для webhook

Для налаштування виклику webhook із CI/CD потрібно виконати (див. додаток А):

– webhook Git-сервера, який запускає pipeline у GitLab / GitHub Actions;
– job у цьому pipeline звертається до API ‘/api/ci/create-environment’ і передає назву проєкту та гілку.

У результаті кожен пуш у гілку може автоматично створювати або оновлювати ізольоване production-подібне середовище саме для цієї гілки, без дій вручну.

3.3 Тестування платформи ізольованих production-подібних середовищ

Тестування розробленої платформи автоматизованого розгортання ізольованих production-подібних середовищ проводилось з метою перевірки її функціональної коректності, стабільності роботи, рівня ізоляції та відповідності визначеним вимогам. Основним завданням тестування було підтвердити можливість створення відтворюваних, масштабованих і безпечних середовищ у межах кластерної інфраструктури.

Форма реєстрації користувача, де e-mail тільки з дозволених доменів із налаштувань системи (див.рис.3.4).

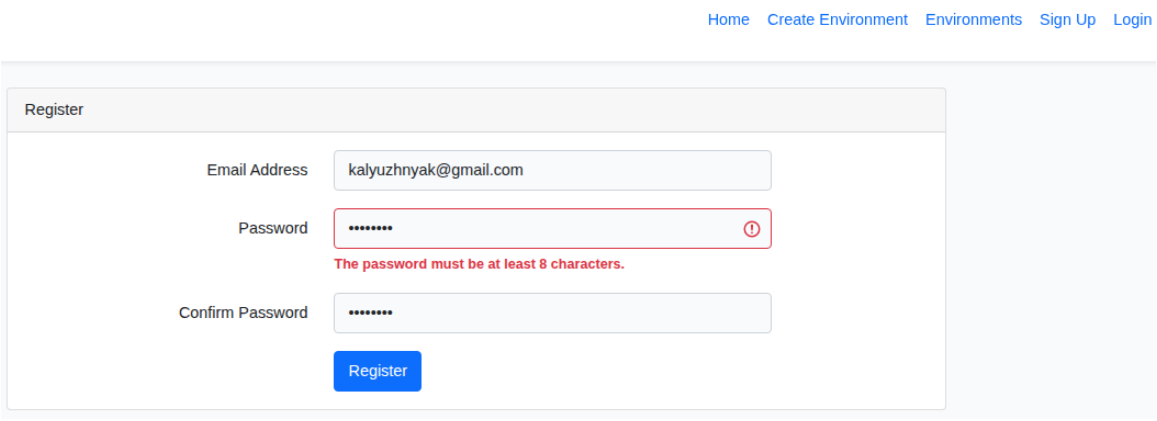


Рисунок 3.4 – Форма авторизації

Успішна реєстрація з дозволеним доменом (див.рис.3.5).

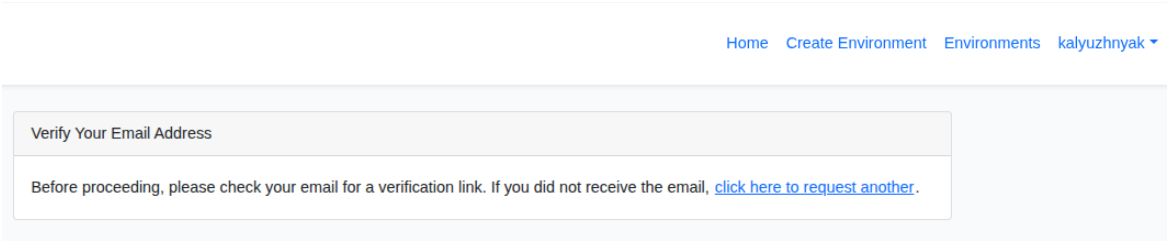


Рисунок 3.5 – Успішна авторизація

Екран профілю користувача з відображенням SSH-ключів (public/private key) (див. рис. 3.6).

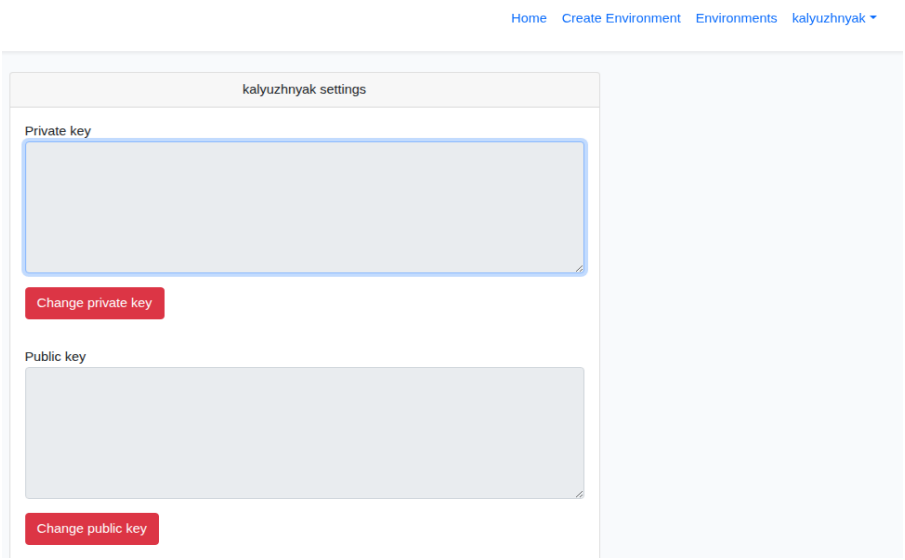


Рисунок 3.6 – Профіль користувача

Створення нового середовища (RequestPage → Create environment). Для цього необхідно обрати «Environment template» зі списку шаблонів та ввести поле «Project branch» (для шаблонів, де потрібна гілка) (див.рис.3.7).

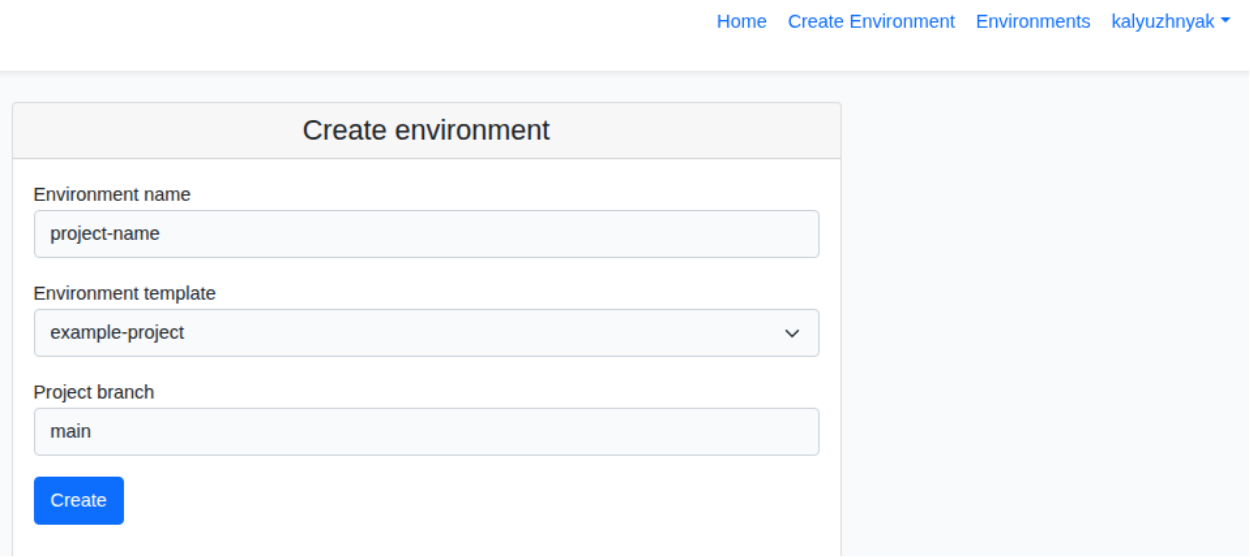


Рисунок 3.7 – Створення середовища

Загальний список середовищ (namespaces) для користувача реалізовано наступним чином (див.рис.3.8).

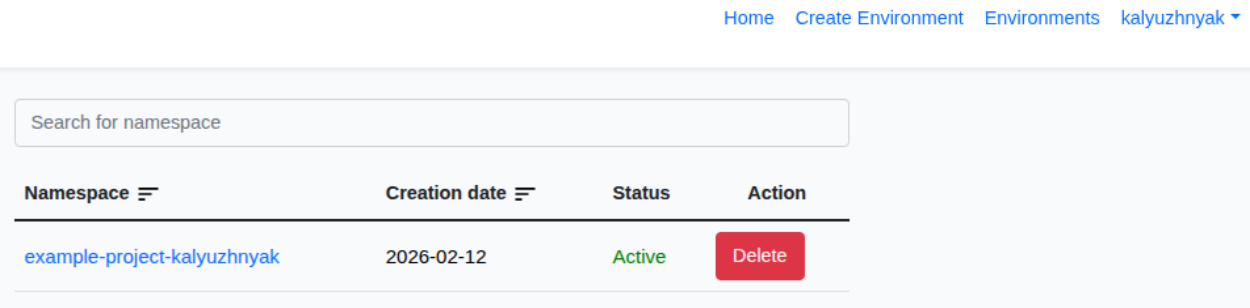


Рисунок 3.8 – Перегляд середовищ

Перехід зі списку середовищ до сторінки «Environment info» для конкретного namespace (див.рис.3.9).

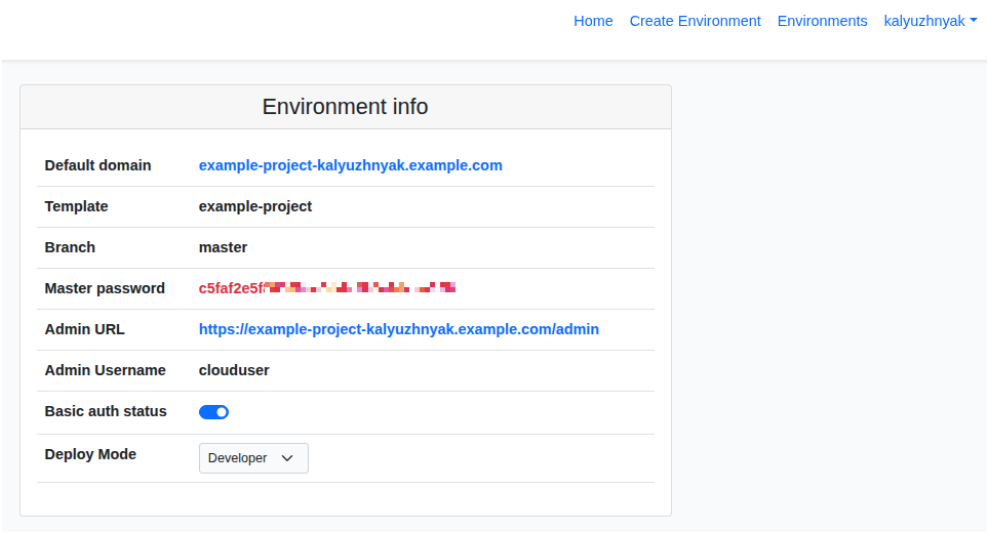


Рисунок 3.9 – Інформація про середовище

Реалізовано доступ до IDE та phpMyAdmin для середовища. Блок IDE, який містить URL таких IDE як VSCode, PHPStorm, та систему управління базами Phpmyadmin (див.рис.3.10).

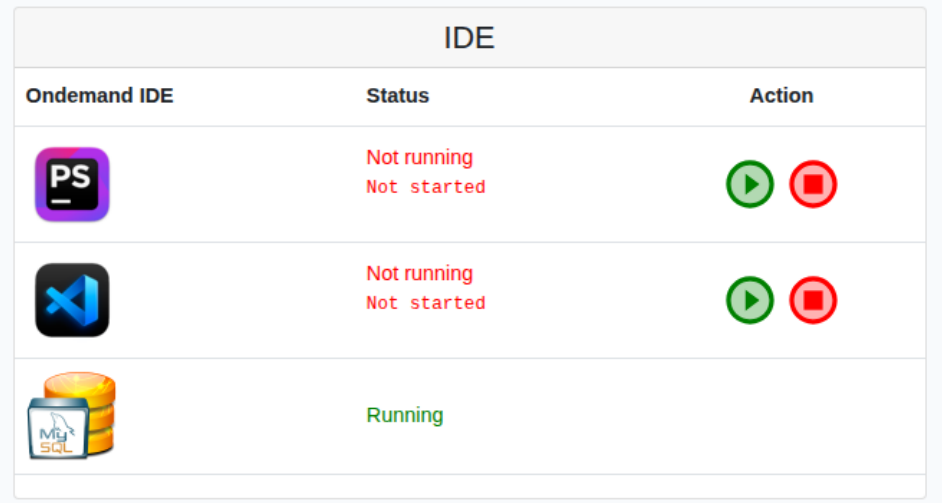


Рисунок 3.10 – доступ до IDE та phpMyAdmin

Блок Services, який включає основні сервіси (nginx, mysql, exim, php та його модуль xdebug) (див.рис.3.11).

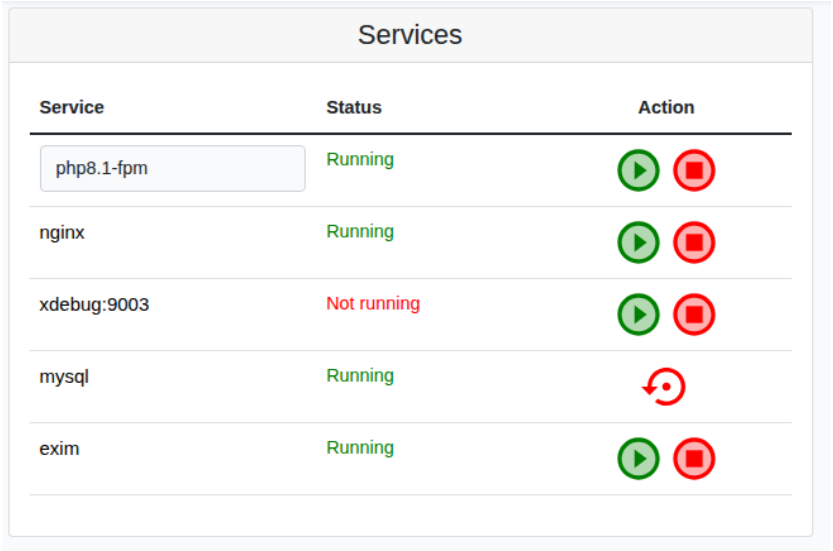


Рисунок 3.11 – Основні сервіси

Проведене тестування підтвердило працездатність розробленої платформи автоматизованого розгортання ізольованих production-подібних середовищ. Система забезпечує належний рівень ізоляції, підтримує механізми масштабування та відновлення після відмов, що дозволяє використовувати її як у науковому процесі, так і для моделювання реальних production-сценаріїв.

3.4 Аналіз продуктивності та стабільності платформи

Реалізація платформи автоматизованого розгортання production-подібних середовищ передбачає проведення аналізу її продуктивності, масштабованості та стабільності. Дослідження виконувалося з метою оцінки ефективності розробленого рішення, визначення його можливостей у роботі з декількома середовищами та перевірки стабільності функціонування системи. Порівняння виконувалося з трьома основними аналогами: Heroku, Coder, Terraform. Переваги розробленої платформи по відношенню до інших це фіксована вартість, повний контроль (див.рис.3.12).

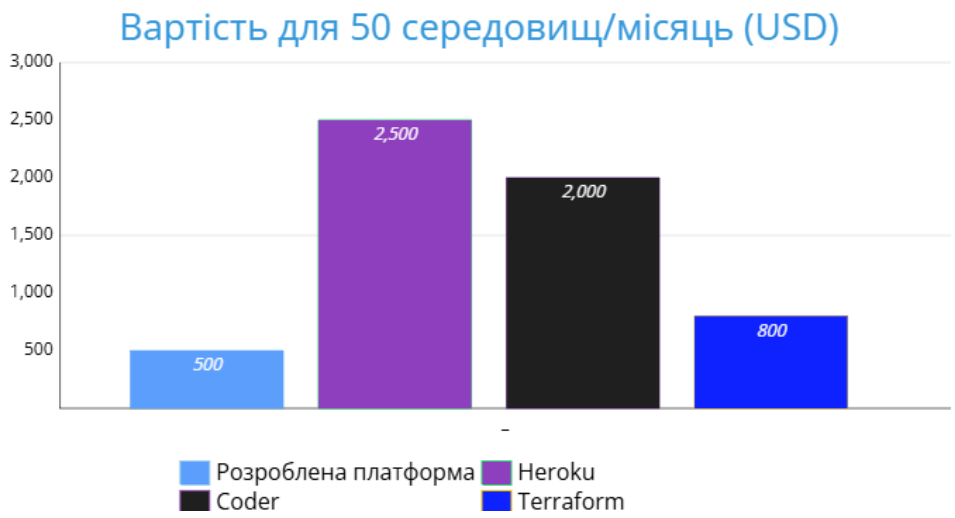


Рисунок 3.12 – Аналіз вартості

Розроблена платформа має найкращий показник при аналізі налаштування нового користувача – 2 хвилини. Даний аналіз проводився на повністю ідентичних проєктах та параметрах (див.рис.3.13).



Рисунок 3.13 – Часовий аналіз

Найменший час розгортання демонструє розроблена платформа, для якої середній час створення та запуску середовища становить приблизно 3 хвилини. Це пояснюється використанням попередньо налаштованих контейнерних конфігурацій, автоматизованих сценаріїв розгортання та мінімальною кількістю ручних налаштувань (див.рис.3.14).



Рисунок 3.14 – Аналіз розгортання проєкту

Реалізована платформа забезпечує абсолютний контроль над усіма аспектами інфраструктури. Натомість наявні аналоги мають обмежений доступ до управління, і передбачають наявність закритих налаштувань.

Результати демонструють, що запропонована платформа забезпечує більш швидке розгортання production-подібних середовищ, що є важливою перевагою для процесів розробки, тестування та безперервної інтеграції. Скорочення часу створення середовища дозволяє підвищити ефективність роботи розробників, пришвидшити цикл розробки програмного забезпечення, покращити загальну продуктивність DevOps-процесів.

3.5 Практичне застосування платформи у навчальному та науковому процесі

Розроблена платформа автоматизованого розгортання ізольованих production-подібних середовищ може бути ефективно використана як у навчальному процесі підготовки фахівців IT-спеціальностей, так і у науково-дослідній діяльності. Її впровадження дозволить створити сучасну інфраструктурну основу для проведення лабораторних робіт, практичних

занять, курсових і дипломних проєктів, а також експериментальних досліджень. В межах освітніх програм з комп’ютерних наук, програмної інженерії та кібербезпеки дана платформа може використовуватися для:

- створення індивідуальних ізольованих середовищ для кожного студента;
- організації лабораторних робіт з контейнеризації та оркестрації;
- моделювання production-сценаріїв розгортання ПЗ;
- навчання принципам DevOps та Infrastructure as Code;
- проведення практичних занять із розподілених систем.

Використання автоматизованого розгортання дозволить створювати середовища заздалегідь за визначеними шаблонами, що забезпечить однакові умови виконання завдань для всіх студентів. Таким чином, усувається проблема різних локальних конфігурацій та зменшується час на налаштування програмного забезпечення.

У науковому процесі платформа може використовуватися для:

- відтворення експериментальних середовищ;
- моделювання різних сценаріїв навантаження;
- дослідження ефективності алгоритмів масштабування;
- аналізу відмовостійкості розподілених систем;
- тестування прототипів мікросервісних архітектур.

Production-подібні середовища дозволяють проводити експерименти в умовах, максимально наближених до реальної експлуатації. Отже, підвищується достовірність отриманих результатів та забезпечується повторюваність експериментів.

Автоматизація розгортання дає змогу швидко створювати нові тестові сценарії без ручного налаштування інфраструктури, що суттєво скорочує час підготовки дослідження.

Впровадження платформи забезпечить централізоване керування навчальною інфраструктурою, зменшить витрати на адміністрування, підвищить рівень практичної підготовки студентів.

Розроблена платформа має значний потенціал практичного застосування у навчальному та науковому процесі. Так як вона дозволяє створювати ізольовані, масштабовані та відтворювані production-подібні середовища, сприяє підвищенню якості освітньої підготовки та ефективності наукових досліджень. Впровадження відповідає сучасним тенденціям цифрової трансформації освітнього середовища та розвитку DevOps-практик.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було розглянуто питання автоматизованого розгортання ізольованих production-подібних середовищ у межах сучасного DevOps-підходу. У ході роботи виконано проєктування, розробку та реалізацію прототипу платформи, що забезпечує створення, масштабування та керування ізольованими середовищами на основі контейнеризації та оркестрації.

У теоретичній частині досліджено класифікацію середовищ розробки, тестування та експлуатації, визначено їх роль у життєвому циклі ПЗ. Розглянуто сучасні підходи до контейнеризації, принципи оркестрації контейнерів та їх значення для забезпечення масштабованості, відмовостійкості й ефективного використання ресурсів. Проаналізовано існуючі інструменти автоматизації інфраструктури та визначено їх можливості й обмеження.

У практичній частині описано архітектуру розробленої платформи, визначено технічні вимоги до її функціонування, реалізовано механізм автоматизованого розгортання production-подібних середовищ та проведено тестування працездатності й ефективності запропонованого рішення. Результати тестування підтвердили стабільність роботи системи, належний рівень ізоляції середовищ та можливість їх масштабування при змінному навантаженні.

Дослідження, проведені під час виконання дипломної роботи, дозволили зробити узагальнення теоретичного та практичного характеру щодо доцільності використання контейнерної оркестрації як основи для створення ізольованих production-подібних середовищ.

Відповідно до поставленої мети під час виконання роботи були реалізовані такі завдання:

– проведено аналіз відомих підходів та інструментів розгортання середовищ у межах DevOps;

- визначено вимоги до production-подібних середовищ та платформи їх автоматизованого розгортання;
- розроблено архітектуру платформи з використанням контейнеризації;
- реалізовано прототип платформи автоматизованого розгортання;
- проведено тестування працездатності та ефективності запропонованого рішення.

Підсумовуючи результати виконаної роботи, можна стверджувати, що всі поставлені завдання реалізовано, а мету дослідження досягнуто. Розроблена платформа може бути використана для створення ізольованих production-подібних середовищ у навчальному та науковому процесі, а також як основа для подальшого вдосконалення інфраструктурних рішень у сфері автоматизації розгортання програмних систем.

ПЕРЕЛІК ПОСИЛАНЬ

1. Newman S. Building Microservices. 2nd ed. Sebastopol : O'Reilly Media, 2021. 617 p.

2. Lukša M. Kubernetes in Action. 2nd ed. Shelter Island : Manning Publications, 2023. 768 p.

3. Domingus J., Arundel J. Cloud Native DevOps with Kubernetes. 2nd ed. Sebastopol : O'Reilly Media, 2022. 356 p.

4. Burns B., Beda J., Hightower K. Kubernetes: Up and Running. 3rd ed. Sebastopol : O'Reilly Media, 2022. 368 p.

5. Red Hat. State of Kubernetes Security Report 2023. URL : https://www.redhat.com (дата звернення : 21.02.2026).

6. Kubernetes Documentation. URL : https://kubernetes.io/docs (дата звернення : 21.02.2026).

7. Morris K. Infrastructure as Code. 2nd ed. Sebastopol : O'Reilly Media, 2022. 240 p.

8. HashiCorp. Terraform Documentation. URL : https://developer.hashicorp.com/terraform/docs (дата звернення : 21.02.2026).

9. Coder Technologies. Coder Documentation. URL : https://coder.com/docs (дата звернення : 21.02.2026).

10. Heroku Dev Center. Heroku Platform Documentation. URL : https://devcenter.heroku.com (дата звернення : 21.02.2026).

11. Kim G., Humble J., Debois P., Willis J. The DevOps Handbook. 2nd ed. Portland : IT Revolution Press, 2021. 480 p.

12. Bass L., Weber I., Zhu L. DevOps: A Software Architect's Perspective. Updated ed. Boston : Addison-Wesley, 2022. 240 p.

13. Turnbull J. The Docker Book. 2022 ed. 2022. 350 p.

14. Hightower K. et al. Kubernetes Patterns. Sebastopol : O'Reilly Media, 2021. 250 p.

15. Brikman Y. Terraform: Up and Running. 3rd ed. Sebastopol : O’Reilly Media, 2022. 398 p.

16. Humble J., Farley D. Continuous Delivery. Updated ed. Boston : Addison-Wesley, 2021. 512 p.

17. Ahmad H., Treude C., Wagner M., Szabo C. Smart HPA: A resource-efficient horizontal pod auto-scaler for microservice architectures // arXiv. 2024. URL : https://arxiv.org/abs/2403.07909 (дата звернення : 21.02.2026).

18. CNCF. Cloud Native Survey 2023. URL : https://www.cncf.io/reports (дата звернення : 21.02.2026).

ДОДАТОК А

Код Laravel-ендпоінту для webhook

```
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Log;
use App\Jobs\ProjectDeployQueue;

class EnvironmentWebhookController extends Controller
{
   public function handle(Request $request)
   {
      // Перевірка (щоб викликати міг тільки CI/CD)
      $secret = config('services.deploy_webhook.secret');
      $token  = $request->header('X-CI-Token');

      if (! $token || $token !== $secret) {
         abort(403, 'Invalid webhook token');
      }
      //  Отримати дані про проєкт і гілку від CI/CD
      //    CI-job надсилає щось на кшталт:
      //    {
      //      "project_name": "example",
      //      "project_template": "example-master",
      //      "branch": "feature/new-checkout",
      //      "php_version": "8.3"
      //    }
```

```
            $projectName     = $request->input('project_name');
            $projectTemplate = $request->input('project_template');
            $branch          = $request->input('branch');
            $phpVersion      = $request->input('php_version', '8.3');

            if (! $projectName || ! $projectTemplate || ! $branch) {
                return response()->json([
                    'status'  => 'error',
                    'message' => 'Missing required fields: project_name, project_template or branch',
                ], 422);
            }
            Log::info('Deploy webhook received', [
                'project' => $projectName,
                'template' => $projectTemplate,
                'branch' => $branch,
                'php_version' => $phpVersion,
            ]);
            //  Створення нового середовища на основі цих даних
            ProjectDeployQueue::dispatch(
                $projectName,
                $projectTemplate,
                $branch,
                $phpVersion
            );

            return response()->json([
                'status'  => 'ok',
                'message' => 'Environment creation queued',
            ]);
        }
    }
```