*БАКАЛАВРСЬКА РОБОТА*

**БР. ІП - 09.00.00.000 ПЗ**

**Група ІП-22-1**

**Кипаца Роман**

**2026**

**Івано-Франківський національний технічний університет нафти і газу**

**Факультет інформаційних технологій**

**Кафедра інженерії програмного забезпечення**

**Кипаца Роман Валерійович**

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

# БАКАЛАВРСЬКА РОБОТА

**Використання DevOps-практик для автоматизації життєвого циклу програмного забезпечення**
(назва роботи)

Інженерія програмного забезпечення

( назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

**Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:**

Здобувач освітнього ступеня _____Р.В. Кипаца
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____Піх Володимир Ярославович, доцент, к.т.н.__
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

**Допущено до захисту**
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

**Івано-Франківськ – 2026**

**Івано-Франківський національний технічний університет нафти і газу**

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

**ЗАТВЕРДЖУЮ:**
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2026 р.

# ЗАВДАННЯ
## НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

**Кипаці Роману Валерійовичу**
(прізвище, ім’я, по-батькові)

**1. Тема проекту (роботи) "Використання DevOps-практик для автоматизації життєвого циклу програмного забезпечення"**
керівник проекту (роботи) Піх Володимир Ярославович, доцент, к.т.н.
затверджені наказом вищого навчального закладу від “ 21 ” квітня 2026 р. № 179/7

**2. Строк подання студентом проекту (роботи)** 10 червня 2026 р.

**3. Вихідні дані до проекту (роботи)** Результати і матеріали отримані під час проходження переддипломної практики

**4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)**
1. Аналіз вимог до програмного забезпечення
2. Аналіз вимог і постановка задачі
3. Проектування системи
4. Реалізація проекту
5. Тестування та впровадження

**5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)**
1. Шлях до того, щоб стати DevOps-інженером AWS (рис.1.3, ст.18)
2. Область застосування DevOps у межах SDLC4(рис. 3.1, ст.40)
3. Піраміда автоматизації тестування (рис. 3.2, ст.42)
4. Cloud DevOps (рис. 3.5, ст.48)
5. Модель iObserve (рис.3.9, ст.53)

**6. Консультанти розділів проекту (роботи)**

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |

**2. Дата видачі завдання** **2026 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломного проекту (роботи) | Строк виконання етапів проекту | Примітка |
|---|---|---|---|
| 1 | Визначення та обгрунтування теми роботи | 17.01.2026 | виконано |
| 2 | Огляд існуючих концепцій, рішень та сервісів в даній області | 21.02.2026 | виконано |
| 3 | Побудова моделі або алгоритму власного рішення | 01.03.2026 | виконано |
| 4 | Документування реалізації власного оригінального рішення вибраними засобами | 21.04.2026 | виконано |
| 5 | Оформлення пояснювальної записки бакалаврської роботи | 02.05.2026 | виконано |

Студент _____Кипаца Р.В.
(підпис)

Керівник роботи_____Піх В.Я
(підпис)

# АНОТАЦІЯ

Бакалаврська робота містить 69 сторінки, 14 рисунків, список використаних джерел із 35 найменувань.

**Метою роботи** є дослідження DevOps-практик та інструментів для автоматизації життєвого циклу програмного забезпечення з метою підвищення ефективності розробки, якості продукту та швидкості його доставки.

**Об'єкт дослідження**: процеси автоматизації життєвого циклу програмного забезпечення.

**Предмет дослідження**: методи, інструменти та підходи DevOps, включаючи CI/CD, інфраструктуру як код (IaC), автоматизацію тестування та розгортання.

**Результати дослідження**: проведено аналіз DevOps-практик, визначено ключові принципи автоматизації процесів розробки, досліджено інструменти та технології DevOps.

**У першому розділі** розглянуто теоретичні основи DevOps, його роль у розробці програмного забезпечення.

**У другому розділі** досліджено методи та інструменти DevOps, зокрема платформи автоматизації, концепції CI/CD та практичні аспекти впровадження DevOps.

**У третьому розділі** розглянуто процес автоматизації життєвого циклу програмного забезпечення, проаналізовано виклики впровадження DevOps та оцінено результати використання відповідних практик.

**Висновок:** використання DevOps-практик дозволяє автоматизувати життєвий цикл програмного забезпечення, підвищити якість продукту, скоротити час розробки та забезпечити безперервну інтеграцію і доставку програмних рішень.

**КЛЮЧОВІ СЛОВА:** DEVOPS, CI/CD, АВТОМАТИЗАЦІЯ, ЖИТТЄВИЙ ЦИКЛ ПЗ, INFRASTRUCTURE AS CODE, DOCKER, KUBERNETES, БЕЗПЕРЕРВНА ІНТЕГРАЦІЯ, БЕЗПЕРЕРВНА ДОСТАВКА.

# ANNOTATION

The bachelor's thesis consists of 69 pages, 14 figures, and a reference list containing 35 sources.

**The aim of the work** is to study DevOps practices and tools for automating the software development lifecycle in order to improve development efficiency, product quality, and delivery speed.

**Research object**: processes of software lifecycle automation.

**Research subject**: DevOps methods, tools, and approaches, including CI/CD, Infrastructure as Code (IaC), test automation, and deployment automation.

**Research results:** an analysis of DevOps practices was conducted, key principles of process automation were identified, DevOps tools and technologies were studied, and the features and effectiveness of their implementation in modern projects were examined.

**The first section** describes the theoretical foundations of DevOps, its role in software development, and the use of modern technologies, including elements of artificial intelligence.

**The second section** analyzes DevOps methods and tools, including automation platforms, CI/CD concepts, and practical aspects of DevOps implementation.

**The third section** covers the automation of the software development lifecycle, analyzes the challenges of DevOps adoption, and evaluates the results of applying these practices.

**Conclusion:** the use of DevOps practices enables automation of the software lifecycle, improves product quality, reduces development time, and ensures continuous integration and delivery of software solutions.

**KEY WORDS:** DEVOPS, CI/CD, AUTOMATION, SOFTWARE LIFECYCLE, INFRASTRUCTURE AS CODE, DOCKER, KUBERNETES, CONTINUOUS INTEGRATION, CONTINUOUS DELIVERY

**ЗМІСТ**

# ВСТУП

В даний час в світі програмне забезпечення є ключовим елементом функціонування цифрової економіки, бізнес-систем та інформаційних сервісів. Зростання складності програмних продуктів, швидкі темпи розвитку технологій та необхідність безперервного оновлення систем підкреслюють важливість впровадження ефективних підходів до автоматизації процесів розробки, тестування та розгортання програмного забезпечення. Сучасні виклики, пов’язані зі скороченням часу доставки програмних продуктів, підвищенням їх якості та забезпеченням стабільності роботи систем, вказують на необхідність використання DevOps-практик, які дозволяють інтегрувати процеси розробки та супроводу в єдиний автоматизований цикл.

**Актуальність теми** зумовлена потребою у створенні ефективних підходів до автоматизації життєвого циклу програмного забезпечення, що забезпечують безперервну інтеграцію, доставку та розгортання програмних продуктів. Традиційні підходи до розробки часто не відповідають сучасним вимогам щодо швидкості релізів, масштабованості систем та стабільності їх роботи. У зв’язку з цим особливого значення набувають DevOps-практики, які базуються на автоматизації процесів, використанні інфраструктури як коду та впровадженні безперервних процесів інтеграції та доставки.

**Метою роботи** є дослідження DevOps-практик та їх застосування для автоматизації життєвого циклу програмного забезпечення з метою підвищення ефективності розробки, якості програмних продуктів та швидкості їх доставки.

**Завданнями дослідження** є аналіз теоретичних основ DevOps, дослідження інструментів та технологій автоматизації, вивчення концепцій безперервної інтеграції та доставки (CI/CD), аналіз підходів до використання інфраструктури як коду, дослідження ролі штучного інтелекту в DevOps-процесах, визначення життєвого циклу DevOps, аналіз викликів впровадження DevOps та оцінка ефективності його використання.

**Об’єктом дослідження** є процеси життєвого циклу програмного забезпечення, що включають розробку, тестування, розгортання та супровід програмних систем.

**Предметом дослідження** є методи, практики та інструменти DevOps, що застосовуються для автоматизації життєвого циклу програмного забезпечення, зокрема CI/CD-пайплайни, контейнеризація, інфраструктура як код та засоби моніторингу.

**Методи дослідження** включають аналіз наукових джерел для вивчення теоретичних основ DevOps, порівняльний аналіз інструментів автоматизації, моделювання процесів життєвого циклу програмного забезпечення, а також узагальнення практичного досвіду впровадження DevOps-підходів у сучасних проєктах.

Розроблений підхід передбачає автоматизацію ключових етапів життєвого циклу програмного забезпечення, включаючи інтеграцію коду, автоматичне тестування, розгортання та моніторинг системи. Особлива увага приділятиметься підвищенню швидкості доставки програмних продуктів, забезпеченню їх стабільності, масштабованості та надійності. Очікується, що впровадження DevOps-практик дозволить значно скоротити час розробки, підвищити якість програмного забезпечення та забезпечити безперервність процесів його оновлення.

Бакалаврська робота містить 69 сторінки, 14 рисунків, 3 розділи, список використаних джерел із 35 найменувань.

# РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ DEVOPS ТА АВТОМАТИЗАЦІЇ ЖИТТЄВОГО ЦИКЛУ ПЗ

## 1.1 Основи DevOps та його роль у розробці програмного забезпечення

Автоматизація є одним із ключових компонентів архітектури DevOps, що сприяє ефективності, надійності та адаптивності в процесах розробки та розгортання програмного забезпечення. У цьому розділі детально розглядаються дослідження переваг автоматизації в контексті DevOps. Цей розділ має на меті надати читачам всебічне уявлення про суттєвий вплив автоматизації на сучасні процеси розробки програмного забезпечення шляхом розгляду теоретичних ідей та емпіричних досліджень.

Культурний та професійний рух, спрямований на усунення розриву між розробкою програмного забезпечення (Dev) та IT-операціями (Ops), в останні роки був визначений як DevOps. Скорочення життєвого циклу розробки систем при одночасному збереженні частих випусків оновлень, виправлень та нових функцій, що відповідають бізнес-цілям, є фундаментальним компонентом мислення DevOps. Досліджували ці фундаментальні ідеї, зосереджуючись на тому, як розробка та операції можуть бути інтегровані за допомогою стандартних інструментів та процедур. Ці роботи підкреслюють незамінну роль автоматизації в досягненні безперебійної співпраці та ефективності робочого процесу між командами розробників та операцій.

Крім того, автоматизація в розробці програмного забезпечення виходить за рамки використання основних інструментів; вона являє собою зміну парадигми до створення цілісного середовища, де збірка програмного забезпечення, тестування, розгортання та управління інфраструктурою автоматизовані для підвищення швидкості та надійності. Емпіричні дані, наведені, підкреслюють критичний зв'язок між автоматизацією в DevOps та високою продуктивністю організації. Автоматизація забезпечує швидкий, надійний та частий потік програмного забезпечення від розробки до експлуатації, що є основоположним

принципом успіху моделі DevOps.

Крім того, вкрай важливо застосовувати системний підхід, щоб забезпечити сумісність із загальними цілями та зрозуміти взаємозв'язок операцій DevOps. Організації можуть краще зрозуміти, як зміни в одному аспекті життєвого циклу розробки програмного забезпечення можуть вплинути на весь процес, розглядаючи його як інтегровану систему. Ця перспектива особливо актуальна при розгляді автоматизації, оскільки вона гарантує, що автоматизовані процеси розробляються відповідно до загальних цілей розробки та експлуатації програмного забезпечення.

Більше того, ландшафт автоматизації DevOps характеризується різноманітними інструментами та технологіями, які оптимізують різні етапи розробки та життєвого циклу програмного забезпечення зображено на рисунку 1.1. Від Jenkins для безперервної інтеграції та доставки до Docker та Kubernetes для контейнеризації та оркестрації, ці технології представляють перехід від ручних до автоматизованих робочих процесів, необхідних для досягнення швидкої та надійної доставки програмного забезпечення.


Рисунок 1.1 - Зображення циклів безперервної інтеграції, доставки та зворотного зв'язку, що стимулюють швидку розробку, тестування та розгортання в сучасних середовищах розробки програмного забезпечення

Підсумовуючи, у цьому розділі розглядається функція автоматизації в архітектурі DevOps. У ньому аналізуються теоретичні ідеї, фактичні дані та

галузеві практики, а також встановлюється основа для подальшого обговорення окремих аспектів автоматизації DevOps, таких як технології, найкращі практики та перешкоди.

DevOps виник як рішення недоліків існуючої моделі розробки програмного забезпечення та розриву зв'язків між командами розробки та експлуатації. Модель Waterfall створювала ізольованість між командами, що затримувало зворотний зв'язок, мінімізувало співпрацю та призводило до неузгоджених цілей. Патрік Дебуа винайшов термін «DevOps» ще у 2009 році, щоб зрозуміти об'єднання процесів розробки та експлуатації в надії покращити співпрацю між ними, скоротити час, необхідний для виведення програмного забезпечення на ринок, та покращити якість можливого програмного забезпечення з часом [5]. DevOps розвивається в багатьох сферах з моменту свого заснування. Через деякий час DevOps почав використовувати такі практики, як безперервна інтеграція та безперервна доставка (CI/CD), інфраструктура як код (IaC) та автоматизоване тестування, кожна з яких вирішує конкретні проблеми SDLC [6]. Ґрунтуючись на цих практиках, організації почали розробляти інструменти для швидшого розгортання програмного забезпечення, швидкого реагування на помилки та випуску високоякісного програмного забезпечення з меншими людськими зусиллями.

DevOps – це більше, ніж просто набір практик; це культурний та професійний рух, спрямований на об'єднання розробки програмного забезпечення (Dev) та IT-операцій (Ops). Основна мета полягає у скороченні життєвого циклу розробки систем, одночасно часто надаючи функції, виправлення та оновлення у тісній відповідності до бізнес-цілей. У роботі використовується наративний стиль для дослідження цих концепцій, підкреслюючи інтеграцію розробки та операцій за допомогою спільних процесів та інструментів. Важливість інтеграції розробки та операцій за допомогою спільних процедур та інструментів також відзначають, які вказують на автоматизацію як ключ до досягнення безперебійної ефективності робочого процесу.

Дослідження ідеї DevOps та стверджують, що організаційні перешкоди слід усунути для сприяння швидшій та надійнішій розробці програмного

забезпечення. Аргумент стверджує, що традиційне розділення розробки та операцій призводить до неефективності та вузьких місць. У своїх практичних ідеях щодо впровадження DevOps, наголошуючи на автоматизації як ключовому компоненті.

Автоматизація значною мірою допомагає оптимізувати життєвий цикл розробки програмного забезпечення. Команди DevOps можуть зменшити кількість помилок, що виникають вручну, та пришвидшити терміни реалізації, автоматизуючи такі процеси, як компіляція коду, тестування та розгортання. Підкреслюють важливість автоматизації, яка дозволяє командам швидко та послідовно випускати зміни завдяки безперервній інтеграції та реалізації (CI/CD).

Як зазначали автоматизація сприяє співпраці та комунікації між командами розробки та експлуатації. За даними, шляхи постійної доставки та інтеграції пропонують швидкий зворотний зв'язок щодо змін у коді, заохочуючи відповідальність та прозорість.

Автоматизація є основою DevOps, що дозволяє компаніям надавати послуги клієнтам швидше та ефективніше. Організації можуть стати більш інноваційними та адаптивними на конкурентному ринку, впроваджуючи автоматизацію та усуваючи внутрішні бар'єри.

Автоматизація в DevOps виходить за рамки простого використання інструментів; вона передбачає створення єдиного середовища, де автоматизована збірка програмного забезпечення, тестування, розгортання та управління інфраструктурою підвищують швидкість та надійність. Пропонують емпіричні дані, які встановлюють зв'язок між відмінною продуктивністю організації та автоматизацією. Автори підкреслюють внесок автоматизації в основний принцип моделі DevOps – швидку, надійну та часту поставку програмного забезпечення.

Автоматизація відіграє важливу роль у сприянні здатності фірм оперативно реагувати на потреби ринку, пришвидшуючи розгортання та тестування змін у програмному забезпеченні. Команди DevOps можуть автоматизувати процеси інтеграції коду, тестування та розгортання, щоб зменшити затримки та підвищити надійність випуску програмного забезпечення.

В опитуванні також підкреслюється важливість автоматизації для розвитку культури постійного вдосконалення в DevOps-компаніях. Команди можуть виділяти більше часу та ресурсів на інновації та діяльність, що додає цінність, автоматизуючи повторювані операції та уникаючи ручного втручання.

Емпіричні результати підкреслюють, як автоматизація має революційний ефект у системах DevOps. Організації можуть досягти підвищеної гнучкості, надійності та ефективності у своїх процесах розробки програмного забезпечення, впроваджуючи автоматизацію як фундаментальний принцип, що зрештою призводить до конкурентної переваги та успіху бізнесу.

Неперервна інтеграція (CI) – це практика автоматичного об'єднання змін коду в централізований репозиторій та їх автоматичного тестування. Безперервна інтеграція (CI Continuous Integration) – це практика, яка дозволяє командам розробників часто інтегрувати новий код у спільний репозиторій, тоді як інші гілки можуть автоматично створювати та запускати тести щоразу, коли надходить новий код, а використання методів CI/CD значно зменшує кількість помилок інтеграції та покращує швидкість доставки програмного забезпечення [7]. Методи CI/CD автоматизують ручні процеси, дозволяючи організаціям випускати нові релізи з мінімальними дефектами та набагато швидшими термінами виконання.

Автоматизоване тестування є надзвичайно важливим для успішного DevOps. Це означає, що код ретельно тестується на кожному етапі життєвого циклу розробки, щоб запобігти потраплянню помилок у продакшн. Це економить час, який не потрібно витрачати на ручне тестування, а також
підтримує якість продукту протягом усього процесу SDLC. Крім того, як зазначається в [10], інтеграція автоматичного тестування в DevOps зменшує ймовірність дефектів і забезпечує швидший зворотний зв'язок щодо змін у коді.

Розуміння того, як різні частини процесу DevOps взаємопов'язані та як зміни в одному компоненті можуть вплинути на систему, вимагає системного підходу. Ця точка зору гарантує, що автоматизовані процедури відповідають загальним цілям, що підтверджується теоретичними моделями.

Розуміння складних взаємозв'язків між багатьма частинами процесу DevOps та того, як зміни в одній частині можуть вплинути на всю систему, вимагає системного мислення. Ця методологія гарантує, що автоматизовані процедури відповідають загальним цілям, як це рекомендовано теоретичними моделями.

Системне мислення підкреслює всебічне розуміння життєвого циклу розробки програмного забезпечення, в якому кожен компонент впливає та взаємодіє з усіма іншими компонентами. Ця точка зору є критично важливою в DevOps, де автоматизація присутня всюди. Вона гарантує, що автоматизовані

Процеси розроблені таким чином, щоб вони відповідали загальним цілям команд, відповідальних за розробку та експлуатацію програмного забезпечення, а не діяли окремо.

Організації можуть виявляти залежності, проблеми та цикли зворотного зв'язку в рамках DevOps, застосовуючи системне мислення. Таким чином, вони можуть розробляти та впроваджувати стратегії автоматизації, які максимізують систему, що призводить до підвищення ефективності та результативності. Крім того, сприяючи співпраці та комунікації між командами DevOps, системне мислення сприяє поширенню стандартних знань про систему та її цілі.

Підсумовуючи, системне мислення є основоположним у DevOps, забезпечуючи проектування та виконання автоматизованих процесів. Застосовуючи цей цілісний підхід, організації можуть узгодити зусилля з автоматизації з цілями, тим самим сприяючи постійному вдосконаленню своїх практик DevOps.

### 1.2 Інструменти та технології автоматизації в DevOps

Практика DevOps вимагає підтримки відповідних технологій. Існує кілька інструментів DevOps, таких як Jenkins та Codeship (безперервна інтеграція, безперервне тестування), Puppet та Ansible (управління хмарою), New Relic та AWS CloudWatch (моніторинг) , Bitbucket та Github (репозиторій), MongoDB (управління базами даних NoSQL) та HipChat (комунікація команди DevOps).

Каталогізація таких інструментів та їх корисності дозволить організаціям приймати обґрунтовані рішення щодо різних типів технологій DevOps та їхніх локальних потреб можна побачити на рис 1.2.

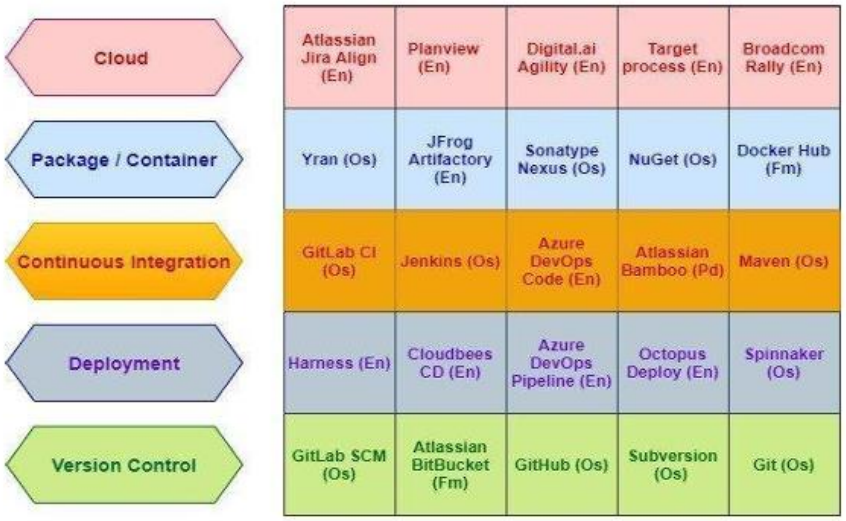


Рисунок 1.2 - «Срібна куля» для індустрії програмного забезпечення

Широкий спектр інструментів і технологій широко використовується в автоматизації DevOps для пришвидшення та покращення етапів розробки та експлуатації програмного забезпечення. Деякі важливі технології, що трансформують автоматизацію в безперервній інтеграції, доставці, контейнеризації та оркестрації.

Автоматизація DevOps головним чином залежить від різних інструментів і технологій, призначених для пришвидшення та покращення етапів розробки та експлуатації програмного забезпечення. Деякі важливі технології, що трансформують автоматизацію в безперервній інтеграції, доставці, контейнеризації та оркестрації, - це Jenkins, Docker та Kubernetes.

Ці рішення дозволяють командам DevOps автоматизувати важливі процедури, підвищуючи продуктивність, надійність та масштабованість. Наприклад, Jenkins автоматизує розробку, тестування та розгортання програмних додатків, сприяючи безперервній інтеграції та доставці. Docker трансформує контейнеризацію, пропонуючи портативний, легкий метод упаковки та розподілу

програмного забезпечення в різних середовищах. Крім того, спрощуючи оркестрацію контейнерів, команди DevOps можуть автоматизувати розгортання, масштабування та адміністрування контейнерних додатків у розподіленому середовищі за допомогою Kubernetes.

Використовуючи ці інструменти, організації можуть зменшити кількість помилок, що виникають вручну, пришвидшити процес доставки програмного забезпечення та підвищити загальну операційну ефективність. Команди DevOps можуть більше зосередитися на інноваціях та наданні цінності споживачам, впроваджуючи ці технології автоматизації, замість трудомісткої ручної роботи, яка займає багато часу.

Автоматизація процедур збірки та розгортання, збереження можливості розгортання коду та гарантування безперервного тестування є прикладами найкращих практик, яких необхідно дотримуватися для ефективного впровадження автоматизації в DevOps. Підкреслюють, що для оптимізації автоматизованих процесів за допомогою механізмів зворотного зв'язку на основі метрик вкрай важливо впроваджувати культуру безперервного розвитку та навчання.

Найкращі практики DevOps, включаючи автоматизацію, контейнеризацію та CI/CD, стали незамінними для IT-компаній, таких як A4 Technology Solutions. Ці практики не лише забезпечують швидшу та надійнішу розповсюдження програмного забезпечення, але й мають значний вплив на IT-Індустрію див. рис. 1.3.


Рисунок 1.3 - Шлях до того, щоб стати DevOps-інженером AWS

Як показує ринкова аналітика, DevOps знаходиться на траєкторії зростання, і його впровадження продовжує революціонізувати підхід організацій до розробки та експлуатації програмного забезпечення. Впровадження DevOps – це більше, ніж просто тренд; залишатися конкурентоспроможним у сучасному швидкоплинному цифровому світі є необхідним.

Ці найкращі практики гарантують ефективне та результативне впровадження автоматизації в DevOps. Організації можуть забезпечити швидке та надійне розгортання змін коду у виробничих середовищах, підтримуючи можливості розгортання коду. Автоматизація процесів збірки та розгортання оптимізує конвеєр доставки програмного забезпечення, зменшуючи кількість помилок, що виникають вручну, та прискорюючи час виходу на ринок.

Безперервне тестування гарантує ретельне тестування змін коду протягом усієї розробки, підвищуючи якість та надійність програмного забезпечення.

Крім того, впровадження культури постійного вдосконалення та навчання дозволяє організаціям оптимізувати свої процеси автоматизації з часом. Збираючи та аналізуючи показники, пов'язані з ефективністю автоматизації, команди DevOps можуть визначати області для покращення та приймати рішення на основі даних для підвищення ефективності та результативності. Такий ітеративний підхід до автоматизації сприяє інноваціям та постійному вдосконаленню практики розробки програмного забезпечення.

Підсумовуючи, дотримання найкращих практик та культура постійного вдосконалення є важливими для ефективного впровадження автоматизації в DevOps. Дотримуючись цих принципів та використовуючи механізми зворотного зв'язку на основі метрик, організації можуть максимізувати переваги автоматизації та досягти високого рівня гнучкості, надійності та ефективності у своїх процесах розробки програмного забезпечення.

Автоматизація в DevOps має багато переваг, але також має проблеми. До них належать проблеми безпеки, зміни в організаційній культурі та складність інтеграції інструментів. Досягнення балансу між швидкістю та стабільністю має вирішальне значення для запобігання зниженню якості чи надійності

програмного забезпечення під час швидкого розгортання, що стало можливим завдяки автоматизації.

Автоматизація в DevOps має багато переваг, але також має недоліки. До них належать проблеми безпеки, зміни в організаційній культурі та складність інтеграції інструментів. Як зазначено, досягнення балансу між швидкістю та стабільністю є важливим для гарантування того, що швидке розгортання за допомогою автоматизації не поставить під загрозу якість чи надійність програмного забезпечення.

Дослідження показує, що командам DevOps може знадобитися допомога через складність інтеграції різних інструментів та технологій автоматизації.

Додаткові витрати та складність процесу автоматизації можуть бути пов'язані з тим, що кілька технологій мають несумісні інтерфейси або потребують унікальної інтеграції. Крім того, для повного впровадження автоматизації в DevOps також необхідні зміни в організаційній культурі. Впровадження та ефективність автоматизації можуть бути ускладнені відсутністю командної роботи, опором змінам та акцентом на старих методах.

Крім того, автоматизовані процеси створюють проблеми безпеки, які можуть наражати підприємства на небезпеку. Прогалини безпеки та витоки даних можуть бути результатом вразливостей скриптів автоматизації або неправильних конфігурацій конвеєра автоматизованого розгортання. Команди DevOps повинні надавати безпеці головний пріоритет протягом усього життєвого циклу автоматизації. Щоб зменшити ці ризики, вони повинні вживати надійних заходів безпеки, включаючи перевірки коду, автоматизоване тестування безпеки та методи контролю доступу.

Крім того, автоматизація DevOps вимагає балансу між стабільністю та швидкістю. Хоча швидке надання функцій та оновлень залежить від швидкого розгортання, якість та надійність програмного забезпечення повинні бути достатніми. Наголошено на важливості впровадження таких процедур, як автоматизоване тестування, плани відкату та розгортання за принципом «канарі», щоб гарантувати, що зміни, внесені за допомогою автоматизації, не матимуть негативного

впливу на виробництво.

На завершення, автоматизація в DevOps має багато чого запропонувати, але для повного розкриття її потенціалу необхідно подолати перешкоди. Команди DevOps можуть ефективно використовувати автоматизацію для пришвидшення розробки програмного забезпечення, зберігаючи при цьому високі стандарти якості та надійності, вирішуючи проблеми, пов'язані з корпоративною культурою та безпекою, а також дотримуючись балансу між швидкістю та стабільністю. Ці проблеми включають труднощі з інтеграцією інструментів та труднощі з балансуванням швидкості та стабільності [1].

### 1.3 Використання штучного інтелекту та машинного навчання в DevOps

Оскільки складність та масштаб програмних систем продовжують зростати, організації все частіше звертаються до штучного інтелекту (ШІ) та машинного навчання (МН) для покращення своїх практик DevOps. Технології ШІ та МН пропонують нові можливості для автоматизації та оптимізації різних аспектів життєвого циклу розробки програмного забезпечення, від інтеграції та тестування коду до розгортання та обслуговування. Штучний інтелект (ШІ) стосується розробки комп'ютерних систем, які можуть виконувати завдання, що зазвичай потребують людського інтелекту, такі як міркування, навчання та прийняття рішень. Машинне навчання (МН), підмножина ШІ, зосереджується на створенні алгоритмів та моделей, які дозволяють комп'ютерам навчатися на даних та робити прогнози або рішення без явного програмування. Моделі ШІ/МН можуть аналізувати історичні дані з конвеєрів CI/CD, інфраструктури та додатків, щоб прогнозувати потенційні проблеми, такі як збої розгортання, обмеження ресурсів або зниження продуктивності. Моделі ШІ/МН можуть постійно контролювати продуктивність системи та виявляти аномалії в режимі реального часу, дозволяючи командам реагувати на проблеми до їх загострення. Штучний інтелект/машинне навчання (ШІ/МН) може покращити автоматизоване тестування,

визначаючи найважливіші тестові випадки, оптимізуючи покриття тестами та навіть генеруючи нові тестові випадки на основі змін коду. ШІ/МН дозволяє розробляти самовідновлювальні системи, які можуть автоматично виявляти та вирішувати проблеми без втручання людини, такі як масштабування ресурсів, відкат розгортань або застосування виправлень.

Технології штучного інтелекту/машинного навчання (AI/ML) можуть значно підвищити ефективність та точність процесів DevOps, автоматизуючи завдання, які традиційно виконувалися вручну, потребували багато часу та були схильними до помилок. Наприклад, інструменти аналізу коду на базі штучного інтелекту можуть автоматично виявляти та виправляти проблеми з кодом, зменшуючи потребу в ручному перегляді коду. Аналогічно, інструменти розгортання на базі штучного інтелекту можуть оптимізувати терміни та стратегію розгортання, мінімізуючи ризик збоїв та забезпечуючи плавне розгортання. Однією з найважливіших переваг AI/ML в DevOps є здатність проактивно виявляти та вирішувати проблеми, перш ніж вони вплинуть на виробництво. Моделі прогнозної аналітики можуть виявляти потенційні проблеми на основі історичних даних та рекомендувати превентивні дії, такі як масштабування ресурсів або затримка розгортання. Моделі виявлення аномалій можуть контролювати продуктивність системи в режимі реального часу, попереджаючи команди про незвичайні закономірності або поведінку, які можуть свідчити про проблему. Забезпечуючи проактивне виявлення та вирішення проблем, AI/ML зменшує час простою, підвищує надійність системи та покращує загальний користувацький досвід. Технології AI/ML також можуть оптимізувати використання ресурсів у середовищах DevOps, зменшуючи витрати та підвищуючи ефективність. Наприклад, інструменти управління ресурсами на основі штучного інтелекту можуть аналізувати моделі використання та прогнозувати попит, що дозволяє організаціям ефективніше розподіляти ресурси. Це може призвести до значної економії коштів, особливо в хмарних середовищах, де використання ресурсів оплачується на основі оплати за використання.

Окрім оптимізації розподілу ресурсів, штучний інтелект/машинне навчання (ШІ/МО) також може допомогти організаціям ефективніше керувати своєю інфраструктурою [3]. Наприклад, інструменти на базі ШІ можуть автоматично масштабувати інфраструктуру залежно від потреб у режимі реального часу, забезпечуючи ефективне використання ресурсів та збереження продуктивності програм. Інтегруючи ШІ та МО у свої практики DevOps, організації можуть вийти на нові рівні ефективності, точності та надійності, що дозволить їм швидше та впевненіше розповсюджувати програмне забезпечення. З розвитком технологій ШІ/МО очікується, що їхня роль у DevOps зростатиме, пропонуючи ще більше можливостей для покращення процесу розробки та розгортання програмного забезпечення.

Випадки використання штучного інтелекту/машинного навчання в DevOps Автоматизований аналіз якості коду за допомогою штучного інтелекту/модельного навчання (AI/ML) передбачає використання інтелектуальних алгоритмів для оцінки та покращення якості коду. Моделі штучного інтелекту/модельного навчання (AI/ML) можуть аналізувати кодові бази, щоб виявити потенційні проблеми, забезпечити дотримання стандартів кодування та пропонувати покращення. Моделі штучного інтелекту/модельного навчання можуть прогнозувати та вирішувати проблеми з кодом, аналізуючи історичні зміни коду, виявляючи закономірності, пов'язані з помилками, та навчаючись на попередніх оглядах коду. Ці моделі навчаються на великих наборах даних коду, що дозволяє їм розпізнавати поширені помилки коду та потенційні вразливості. Наприклад, моделі штучного інтелекту можуть прогнозувати області коду, які з більшою ймовірністю містять помилки, на основі закономірностей, що спостерігаються в подібних кодових базах. Вони також можуть пропонувати виправлення або покращення, порівнюючи код з найкращими практиками та відомими рішеннями. Такий проактивний підхід допомагає виявляти проблеми на ранніх етапах розробки, зменшуючи ймовірність потрапляння помилок у продакшн.

DeepCode використовує алгоритми машинного навчання для аналізу коду та надання інформації про потенційні проблеми. Він використовує великий набір

даних проектів з відкритим кодом для вивчення шаблонів кодування та виявлення таких проблем, як вразливості безпеки, проблеми з продуктивністю та «запахи» коду. Codacy інтегрується з конвеєрами CI/CD для забезпечення автоматизованих перевірок коду. Він використовує штучний інтелект для оцінки якості коду, виявлення проблем та забезпечення дотримання стандартів кодування. Codacy підтримує кілька мов програмування та пропонує налаштовувані правила та інтеграції з популярними системами контролю версій. Ці інструменти бездоганно інтегруються в робочий процес

розробки, надаючи зворотний зв'язок у режимі реального часу та рекомендації для покращення якості коду. Прогнозна аналітика для розгортання передбачає використання штучного інтелекту для прогнозування оптимального часу та стратегій розгортання змін коду. Аналізуючи історичні дані розгортання, моделі штучного інтелекту можуть рекомендувати найкращий час для розгортання, оцінювати потенційний вплив та пропонувати стратегії для мінімізації ризику. Моделі штучного інтелекту можуть аналізувати такі фактори, як історичні показники успішності розгортання, завантаження системи, моделі активності користувачів та умови навколишнього середовища, щоб передбачити оптимальні вікна розгортання. Це допомагає організаціям уникати годин пікового навантаження, зменшувати ймовірність збоїв та забезпечувати безперебійний процес розгортання. Прогнозна аналітика також може рекомендувати стратегії розгортання, такі як «синьо-зелені» розгортання або «канареєчні» релізи, на основі історичних даних та поточних системних умов. Ці стратегії дозволяють поступово розгортати та мінімізувати вплив потенційних проблем.

Тематичне дослідження: Facebook використовує прогнозну аналітику для оптимізації процесу розгортання. Аналізуючи історичні дані розгортання, компанія може передбачити ймовірність збоїв розгортання та відповідно коригувати свої стратегії розгортання [2]. Такий підхід допоміг Facebook підтримувати високу доступність та продуктивність під час оновлень. Netflix використовує прогнозну аналітику на основі штучного інтелекту для управління своїм складним середовищем розгортання. Компанія використовує моделі машинного навчання

для прогнозування рівня успішності розгортання та рекомендацій стратегій розгортання. Це дозволило Netflix розгортати зміни коду з мінімальними перебоями та підтримувати високий рівень доступності послуг.

Виявлення аномалій у конвеєрах CI/CD передбачає використання моделей AI/ML для моніторингу та виявлення незвичайних закономірностей або поведінки в конвеєрі. Це дозволяє раннє виявляти проблеми та автоматизовано реагувати для запобігання збоям. Моделі AI/ML постійно моніторять конвеєри CI/CD на наявність аномалій, таких як неочікувані зміни в часі збірки, збої тестування або помилки розгортання. Ці моделі аналізують історичні дані та вивчають нормальні робочі моделі, що дозволяє їм виявляти відхилення в режимі реального часу. Наприклад, модель AI може виявити аномалію, якщо процес збірки раптово займає значно більше часу, ніж зазвичай. Виявляючи ці відхилення, модель може попередити команду DevOps та надати інформацію про потенційні причини. Після виявлення аномалії моделі AI/ML можуть автоматично запускати попередньо визначені відповіді або коригувальні дії. Наприклад, якщо аномалія виявлена під час розгортання, система може автоматично повернутися до попередньої стабільної версії або застосувати виправлення для вирішення проблеми.

Тематичне дослідження: GitHub Actions інтегрує виявлення аномалій на основі штучного інтелекту (ШІ) у свої конвеєри CI/CD [4]. Платформа використовує моделі машинного навчання для моніторингу робочих процесів на предмет незвичайної поведінки та автоматично вживає коригувальних заходів, таких як зупинка проблемного розгортання або повідомлення команди про потенційні проблеми. Самовідновлювальна інфраструктура стосується систем, які можуть автоматично виявляти збої та відновлюватися після них без втручання людини. Системи самовідновлення на основі ШІ використовують алгоритми машинного навчання для виявлення та вирішення проблем, забезпечуючи безперервну роботу та мінімальний час простою. Системи самовідновлення використовують моделі ШІ/МО для моніторингу стану та продуктивності інфраструктури. Коли виявляється збій, ці моделі можуть автоматично запускати дії відновлення, такі

як перезапуск служб, масштабування ресурсів або переналаштування середовищ. Наприклад, якщо сервер перестає реагувати, система на основі ШІ може автоматично запустити новий екземпляр і перенаправити трафік, щоб забезпечити безперервну доступність служб. Аналогічно, якщо критичний компонент програми виходить з ладу, система може автоматично розгорнути резервну копію або відкат до попереднього стабільного стану. Приклади систем самовідновлення у виробничих середовищах. Google Cloud Platform використовує механізми самовідновлення на основі штучного інтелекту для підтримки справності своїх хмарних сервісів. Платформа використовує моделі машинного навчання для виявлення та вирішення проблем у режимі реального часу, таких як автоматичне масштабування ресурсів у відповідь на підвищений попит або відновлення після збоїв сервісів. Amazon Web Services (AWS) пропонує можливості самовідновлення через свої сервіси, такі як AWS Auto Scaling та AWS Elastic Load Balancing. Ці сервіси використовують штучний інтелект/моніторинг навчання (ШІ/МО) для моніторингу інфраструктури та автоматичного налаштування ресурсів або перенаправлення трафіку у відповідь на збої або зміни попиту.

### 1.4 Висновки по розділу

У першому розділі було розглянуто теоретичні основи використання DevOps-практик для автоматизації життєвого циклу програмного забезпечення. Проаналізовано сутність підходу DevOps, його основні принципи та роль у забезпеченні ефективної взаємодії між командами розробки й експлуатації. Досліджено сучасні інструменти та технології автоматизації, що використовуються для безперервної інтеграції, тестування, розгортання та моніторингу програмних систем. Також розглянуто можливості застосування штучного інтелекту та машинного навчання у DevOps-процесах для прогнозування помилок, оптимізації ресурсів і підвищення надійності програмного забезпечення. Отримані результати дозволяють сформувати теоретичну базу для подальшого дослідження методів та практичної реалізації DevOps-рішень.

# РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ АВТОМАТИЗАЦІЇ DEVOPS

## 2.1 Інструменти та платформи DevOps

Інструменти штучного інтелекту та машинного навчання все частіше інтегруються в робочі процеси DevOps для підвищення автоматизації, ефективності та надійності. Ці інструменти варіюються від фреймворків машинного навчання до спеціалізованих платформ, розроблених для середовищ DevOps.

TensorFlow, PyTorch та інші фреймворки машинного навчання. TensorFlow — це фреймворк машинного навчання з відкритим кодом, розроблений Google, який широко використовується для створення та розгортання моделей штучного інтелекту. Він забезпечує комплексну екосистему для навчання та обслуговування моделей машинного навчання, що робить його придатним для різних DevOps-застосунків, включаючи прогнозну аналітику та виявлення аномалій. PyTorch, розроблений Facebook, — це ще один популярний фреймворк машинного навчання, відомий своїм динамічним обчислювальним графом та простотою використання. PyTorch використовується для розробки та розгортання моделей машинного навчання в DevOps, зокрема для таких завдань, як аналіз якості коду та моніторинг у режимі реального часу. Інші фреймворки машинного навчання, такі як Scikit-Learn, Keras та XGBoost, також використовуються в DevOps для виконання конкретних завдань. Ці фреймворки пропонують різні інструменти та алгоритми для створення та оцінки моделей машинного навчання.

Інтеграція з існуючими інструментами DevOps. Фреймворки AI/ML можна інтегрувати з існуючими інструментами DevOps для розширення їхніх можливостей. Наприклад: Jenkins, популярний інструмент CI/CD, можна розширити за допомогою плагінів AI/ML, щоб забезпечити інтелектуальний аналіз коду, прогнозну аналітику та виявлення аномалій у конвеєрах CI/CD.

Docker, платформа контейнеризації, може використовувати моделі AI/ML для оптимізації розподілу ресурсів, управління станом контейнерів та автоматизації рішень щодо масштабування. Terraform, інструмент IaC, може інтегруватися з

платформами управління ресурсами на основі штучного інтелекту для оптимізації забезпечення та конфігурації інфраструктури.

Платформи для DevOps на основі штучного інтелекту. Кілька платформ пропонують можливості на основі штучного інтелекту, спеціально розроблені для середовищ DevOps. Ці платформи надають інтегровані рішення для моніторингу, автоматизації та оптимізації. Kubernetes, платформа для оркестрації контейнерів з відкритим кодом, може інтегруватися з інструментами штучного інтелекту/машинного навчання для розширення своєї функціональності. Наприклад, алгоритми на основі штучного інтелекту можуть оптимізувати розподіл ресурсів, керувати справністю контейнерів та автоматизувати рішення щодо масштабування на основі даних у режимі реального часу. Datadog надає комплексну платформу спостереження з функціями на основі штучного інтелекту для моніторингу, ведення журналу та аналітики. Її можливості машинного навчання дозволяють виявляти аномалії, прогнозувати аналітику та автоматизовано реагувати на інциденти. New Relic пропонує рішення для спостереження на основі штучного інтелекту для моніторингу та управління продуктивністю програм. Її моделі машинного навчання можуть виявляти аномалії, прогнозувати проблеми з продуктивністю та надавати практичну інформацію для оптимізації [26].

Тематичні дослідження. LinkedIn використовує штучний інтелект/машинне навчання (ШІ/МН) для різних DevOps-застосунків, включаючи автоматизований аналіз якості коду та прогнозну аналітику для розгортання. Компанія використовує моделі машинного навчання для покращення перевірки коду, оптимізації стратегій розгортання та забезпечення надійності системи. IBM інтегрує ШІ/МН у свої інструменти DevOps для покращення автоматизації та моніторингу. Наприклад, платформа Watson AIOps від IBM використовує машинне навчання для аналізу системних журналів, виявлення аномалій та автоматизації управління інцидентами, підвищуючи операційну ефективність та зменшуючи час простою. Spotify використовує інструменти на основі ШІ для моніторингу та управління своєю складною інфраструктурою. Компанія використовує моделі машинного навчання для аналізу показників продуктивності, виявлення аномалій та оптимізації використання ресурсів, забезпечуючи безперебійний

користувацький досвід. Ці тематичні дослідження демонструють трансформаційний вплив ШІ та МН на практики DevOps, підкреслюючи потенціал цих технологій для підвищення автоматизації, ефективності та надійності розробки та розгортання програмного забезпечення. Проблеми та обмеження. Ефективність моделей штучного інтелекту та машинного навчання в DevOps значною мірою залежить від якості та доступності даних, на яких вони навчаються. Високоякісні дані мають вирішальне значення для розробки точних та надійних моделей, але в цьому відношенні може виникнути кілька проблем: Дані, що використовуються для навчання моделей штучного інтелекту/машинного навчання, можуть надходити з різних джерел і не завжди бути узгодженими. Неузгоджені дані можуть призвести до моделей, які працюють погано або надають ненадійні результати. Неповні дані можуть обмежувати здатність моделей штучного інтелекту/машинного навчання до ефективного навчання. Відсутність точок даних або прогалини в історичній інформації можуть вплинути на точність моделі та призвести до неоптимальної продуктивності. Для моделей навчання з учителем маркування даних є критично важливим кроком. Однак точне маркування великих наборів даних може бути трудомістким та ресурсоємним. Погано марковані дані можуть призвести до упередженості та зниження ефективності моделей. Забезпечення відповідності даних, що використовуються для навчання моделей штучного інтелекту/машинного навчання, правилам та стандартам конфіденційності є надзвичайно важливим. Це включає анонімізацію конфіденційної інформації та впровадження заходів захисту даних. З часом характеристики даних можуть змінюватися, що призводить до дрейфу даних. Моделі, навчені на застарілих даних, можуть стати менш точними, що вимагатиме постійного моніторингу та перенавчання для підтримки продуктивності.

Складність інтеграції ШІ/ML в існуючі конвеєри DevOps. Інструменти та фреймворки ШІ/ML можуть вимагати спеціалізованої інфраструктури та ресурсів, таких як прискорення GPU або TPU. Інтеграція цих вимог з існуючою інфраструктурою DevOps може бути складною та потребувати значних змін. Конвеєри DevOps часто включають різноманітний набір інструментів для CI/CD, моніторингу та розгортання. Інтеграція моделей ШІ/ML в ці інструменти може бути

складною, особливо якщо інструменти не підтримують можливості ШІ/ML безпосередньо. Впровадження рішень ШІ/ML вимагає спеціалізованих навичок та знань. Командам DevOps може знадобитися здобути новий досвід або співпрацювати з фахівцями з обробки даних, щоб ефективно інтегрувати та керувати моделями ШІ/ML. Розгортання моделей ШІ/ML у виробничих середовищах вимагає ретельного планування та управління. Це включає керування версіями моделей, обробку оновлень та забезпечення того, щоб моделі працювали належним чином у реальних умовах.

Моделі штучного інтелекту/машинного навчання (ШІ/МУ) повинні бути масштабованими для обробки великих обсягів даних та вимог до обробки в режимі реального часу. Забезпечення ефективного масштабування моделей в рамках існуючого конвеєра DevOps може бути складним завданням і може вимагати додаткових ресурсів та інфраструктури. Інтеграція ШІ/МУ в DevOps також викликає етичні та безпекові проблеми, які необхідно вирішити:

Моделі ШІ/МО можуть успадковувати упередження, присутні в навчальних даних. Це може призвести до несправедливих або дискримінаційних результатів, особливо в автоматизованих процесах прийняття рішень. Виявлення та пом'якшення упереджень є важливим для забезпечення справедливості та рівності моделей ШІ/МО. Розробка та впровадження моделей ШІ/МО можуть призводити до упереджень, навіть якщо навчальні дані неупереджені. Для мінімізації упереджень та забезпечення справедливості моделі необхідний ретельний розгляд розробки, оцінки та

валідації моделі. Забезпечення прозорості в моделях ШІ/МО важливе для розуміння того, як приймаються рішення. Надання пояснень до прогнозів моделі може допомогти вирішити проблеми щодо упередженості та зміцнити довіру до технології.

Захист конфіденційних даних, що використовуються для навчання моделей ШІ/ML, має вирішальне значення. Це включає впровадження шифрування даних, контролю доступу та безпечних практик обробки даних для захисту від несанкціонованого доступу та витоків даних. Дотримання правил конфіденційності даних, таких як GDPR або CCPA, є важливим під час використання

персональних даних для навчання моделей ШІ/ML. Організації повинні забезпечити дотримання ними законодавчих вимог та отримати необхідну згоду від суб'єктів даних. Самі моделі ШІ/ML можуть бути вразливими до атак, таких як атаки з боку суперника або атаки інверсії моделей. Впровадження заходів безпеки для захисту моделей від експлуатації та забезпечення їхньої стійкості є важливим для підтримки цілісності системи.

Майбутнє DevOps на основі штучного інтелекту. Майбутнє DevOps на основі штучного інтелекту характеризується швидким розвитком та тенденціями, що розвиваються, що обіцяють подальше підвищення автоматизації та ефективності розробки та експлуатації програмного забезпечення:

Майбутній ландшафт штучного інтелекту/машинного навчання в DevOps. Очікується, що інтеграція штучного інтелекту/машинного навчання (AI/ML) призведе до ще більшого рівня автоматизації процесів DevOps. Це включає автоматизацію складніших завдань, таких як інтелектуальне реагування на інциденти, автономне управління інфраструктурою та розширена оптимізація коду. Моделі штучного інтелекту/машинного навчання (AI/ML) продовжуватимуть удосконалюватися у своїй здатності прогнозувати поведінку системи, проблеми з продуктивністю та потенційні збої. Це дозволить приймати більш проактивні та точні рішення, зменшуючи ймовірність збоїв та підвищуючи надійність системи. Зі зростанням поширеності периферійних обчислень, штучний інтелект/машинне навчання (AI/ML) відіграватиме ключову роль в управлінні та оптимізації розгортання периферійних систем. Інструменти на основі штучного інтелекту допоможуть керувати розподіленими системами, оптимізувати розподіл ресурсів та покращити прийняття рішень у режимі реального часу на периферії. Моделі штучного інтелекту/машинного навчання стануть більш досконалими, використовуючи досягнення в глибокому навчанні, навчанні з підкріпленням та обробці природної мови. Це дозволить забезпечити точнішу та контекстно-залежну автоматизацію та оптимізацію в середовищах DevOps.

Новітні технології та їхній потенційний вплив. Квантові обчислення мають потенціал для революціонізування штучного інтелекту/машинного навчання, забезпечуючи складніші обчислення та швидшу обробку. Це може

призвести до проривів у DevOps на основі штучного інтелекту, таких як покращена прогнозна аналітика та алгоритми оптимізації. Федеративне навчання дозволяє навчати моделі ШІ на децентралізованих пристроях, зберігаючи дані локальними. Такий підхід може покращити конфіденційність та безпеку, водночас забезпечуючи спільне навчання моделей у розподілених системах. Інтеграція ШІ в практику DevSecOps підвищить безпеку, автоматизуючи виявлення вразливостей, аналіз загроз та реагування на інциденти. Рішення безпеки на основі ШІ забезпечать захист у режимі реального часу та адаптивні механізми захисту. Бачення повністю автономного DevOps зосереджено на досягненні стану, коли розгортання та обслуговування програмного забезпечення повністю автоматизовані з мінімальним втручанням людини.

Бачення повністю автоматизованого розгортання та обслуговування програмного забезпечення. Повністю автономні конвеєри DevOps використовуватимуть штучний інтелект/машинне навчання (ШІ/МО) для управління всім життєвим циклом розробки програмного забезпечення, від інтеграції коду та тестування до розгортання та моніторингу. Ці конвеєри зможуть приймати рішення в режимі реального часу, оптимізувати робочі процеси та обробляти винятки без втручання людини. Управління інфраструктурою стане повністю автоматизованим, а моделі ШІ/МО оброблятимуть такі завдання, як масштабування, забезпечення та налаштування. Самокерована інфраструктура адаптуватиметься до змінних вимог, забезпечуючи оптимальну продуктивність та використання ресурсів. Бачення включає комплексну автоматизацію всіх процесів DevOps, включаючи аналіз якості коду, стратегії розгортання, управління інцидентами та відновлення системи. Рішення на основі ШІ будуть постійно навчатися та вдосконалюватися, забезпечуючи дедалі складнішу автоматизацію та оптимізацію.

## 2.2 Ключові концепції та фреймворки DevOps (CI/CD, IaC)

Безперервний розвиток включає в себе організацію та кодування продукту. Весь процес удосконалення розділяється на більш скромні цикли розвитку. Ця взаємодія спрощує для групи DevOps пришвидшення загального процесу

розвитку програмування. Цей етап є вирішальним для організації бачення всього циклу удосконалення та дозволяє інженерам повністю зрозуміти припущення проекту. В результаті група починає уявляти його кінцеву мету. Інструменти DevOps не потрібні для планування; натомість використовується багато адаптивних інструментів DevOps, щоб бути в курсі коду. Управління вихідним кодом, часто відоме як система контролю версій вихідного коду, підтримує код. Відомі технології DevOps для підтримки вихідного коду включають SVN, Git, JIRA та Mercurial. Крім того, доступні різні інструменти DevOps, такі як Ant, Gradle та Maven, для об'єднання коду у виконувані документи. Наступний етап життєвого циклу DevOps отримує ці виконувані записи.

Безперервна інтеграція (БІ) включає різні вдосконалення у виконанні тестової взаємодії. Клієнти також надають відгуки/дані для додавання нових елементів до програми. Більшість змін відбуваються у вихідному коді на цьому етапі. БІ стає центром для врегулювання цих регулярних змін щодня або щомісяця. Розробка коду поєднує модульне та узгоджувальне тестування, огляд коду та пакетування. Оскільки розробники впроваджують безперервні вдосконалення, вони можуть швидко виявляти проблеми (якщо такі є) та вирішувати їх на початковому етапі. На цьому етапі з'являються постійні нові функції коду з поточним вихідним кодом. Jenkins є одним з найвідоміших інструментів для безперервної інтеграції. Він допомагає отримати оновлений код та налаштувати виконувану форму.

Наступним етапом життєвого циклу DevOps є етап безперервного тестування, на якому створений код перевіряється на наявність помилок та помилок, які могли поширитися в коді. Саме тут перевірка якості (QA) відіграє значну роль у визначенні зручності використання створеного програмного забезпечення. Плідне завершення взаємодії з QA є важливим для визначення того, чи відповідає продукт вимогам замовника. Автоматизовані інструменти DevOps, такі як JUnit, Selenium та TestNG, використовуються для безперервного тестування, що дозволяє групі QA одночасно перевіряти різні бази коду.

Фреймворки для оркестрації та автоматизації пропонують інструменти та послуги для керування складними робочими процесами та процедурами в

середовищах DevOps та їх автоматизації. Організації можуть використовувати ці фреймворки для автоматизації різних процесів у розподілених та гетерогенних середовищах, включаючи розподіл коду, забезпечення ресурсами, масштабування та моніторинг. Технології автоматизації та оркестрації, такі як Docker Swarm, Apache Mesos та Kubernetes, дозволяють планувати, виявляти сервіси та керувати контейнерами.

Впроваджуючи структури CI/CD, IaC, управління конфігурацією та оркестрації, команди DevOps можуть оптимізувати конвеєри доставки, підвищити загальну гнучкість та надійність у розробці програмного забезпечення та досягти набагато більшого.

Цей огляд літератури зосереджений на використанні автоматизації в DevOps, щоб продемонструвати її ефективність у підвищенні ефективності та надійності розробки та розгортання програмного забезпечення. Використання автоматизації добре підтверджується теоріями, емпіричними дослідженнями та галузевими тенденціями, всі з яких підкреслюють покращення, які автоматизація DevOps принесла процесам розробки.

Щоб зрозуміти, як автоматизація може заповнити прогалину між розробкою додатків та IT-операціями, такі теоретики, написали статті, які проливають світло на це питання. Ці рамки висвітлюють культурні та організаційні зміни, які необхідні для ефективного впровадження автоматизації, та натякають на ідею про те, що має бути постійний процес удосконалення та інтеграції.

Дослідження, також можуть підтвердити, що автоматизація позитивно впливає на ефективність організації, оскільки дозволяє часто, швидко та точно доставляти програмне забезпечення. Галузеві аналітичні дані надають реальні випадки, які ілюструють використання інструментів та технологій автоматизації на різних етапах розробки програмного забезпечення. Безперервна інтеграція з такими інструментами, як Jenkins, контейнеризація з Docker та багато інших класифікуються як зброя переходу від лівих, оскільки вони підвищують швидкість та надійність як фактори доставки програмного забезпечення. Однак, цей огляд також визначає перспективи, недоліки та важливі фактори, які слід враховувати під час впровадження автоматизації в DevOps.

Поєднання численних інструментів може стати складним завданням, необхідні зміни культури в організаціях, а питання безпеки залишатимуться актуальними. Дійсно, у роботі азначається, що хоча переваги автоматизації є очевидними, забезпечення їх реалізації не має негативного впливу на швидкість, стабільність та якість відповідної програмної системи. На завершення, цей огляд літератури надає комплексний аналіз автоматизації в контексті DevOps та представляє основу для подальшого дослідження.

У наступних розділах буде глибше розглянуто різні аспекти автоматизації DevOps з посиланнями на конкретні тематичні дослідження, контрольні показники та практичні підходи. Таким чином, метою цього дослідження є надання організаціям інформації та ресурсів, які дозволять їм краще використовувати переваги автоматизації у своїй дисципліні DevOps, сприяючи розвитку та функціональності компаній у сучасній індустрії розробки програмного забезпечення, що розвивається.

### 2.3 Практичні аспекти впровадження DevOps

У цьому підрозділі викладено методологію, використану для цього дослідження, з акцентом на якісному підході, що здійснюється за допомогою обширного огляду літератури. Для розуміння функції та значення автоматизації в DevOps головною метою є методичний аналіз корпусу досліджень з цієї теми, а також галузевих звітів та тематичних досліджень. Цей метод використовує глибину інформації та досвіду в цій галузі, щоб забезпечити поглиблене дослідження теми.

Це дослідження має на меті узагальнити наявну літературу для визначення ключових тенденцій, викликів та можливостей, що виникають внаслідок автоматизації DevOps, та її наслідків для розробки програмного забезпечення та операційних процедур.

Це дослідження ретельно проаналізує емпіричні дані та теоретичні висновки, щоб надати практикам, дослідникам та політикам глибше уявлення про те, як автоматизація змінює процеси DevOps. Воно також надасть уявлення про

дослідження постійно мінливої сфери розробки програмного забезпечення та операційної ефективності.

Дослідницький дизайн цієї роботи базується на якісних даних, зосереджуючись на нечислових даних. Мета полягає в тому, щоб отримати всебічне розуміння ролі автоматизації в DevOps шляхом вивчення задокументованого досвіду, думок та ідей з різних джерел. Такий дизайн сприяє цілісному погляду на тему, враховуючи різні точки зору та контексти.

Підхід систематичного огляду був обраний завдяки його структурованому та комплексному характеру, що ідеально підходить для синтезу існуючих досліджень щодо ролі та впливу автоматизації в DevOps. Цей метод забезпечує ретельну ідентифікацію, оцінку та синтез відповідних досліджень, підвищуючи надійність та валідність результатів. Завдяки систематичному пошуку в кількох базах даних та використанню попередньо визначених критеріїв включення та виключення, цей дизайн дослідження мінімізує упередженість вибору та забезпечує всебічний аналіз літератури. Крім того, систематичні огляди дозволяють інтегрувати різноманітні точки зору та методології, що дозволяє отримати цілісне розуміння теми. Завдяки цій методологічній ретельності дослідження має на меті надати глибоке розуміння того, як автоматизація впливає на практику DevOps, інформуючи як академічний дискурс, так і практичне прийняття рішень у розробці програмного забезпечення та управлінні операціями.

Тип дослідження для роботи – це якісний огляд літератури. Цей метод передбачає систематичний аналіз існуючої літератури з DevOps та автоматизації. Замість збору нових даних, він синтезує та інтерпретує опубліковані матеріали для розуміння тенденцій, передового досвіду та теоретичних перспектив. Цей метод дозволяє ретельно дослідити тему, використовуючи значний корпус актуальної інформації для отримання висновків та ідей щодо застосування автоматизації в процесах DevOps. Це корисний метод для збору та оцінки багатьох точок зору та результатів досліджень з добре відомої теми, що дає повне уявлення про стан знань на даний момент.

Буде створено ретельний план пошуку літератури для пошуку відповідних досліджень з різних джерел. Для максимізації пошуку відповідної літератури

буде застосовано систематичне поєднання ключових слів та пошукових термінів про DevOps, автоматизацію, розробку програмного забезпечення, операційну ефективність та оцінку впливу. Щодо функції автоматизації в процесах DevOps, цей підхід спрямований на зібрання широкого спектру думок та поглядів. Крім того, автоматизований пошук у базі даних буде доповнено ручним пошуком списків літератури з відповідних досліджень та експертних порад.

Основним методом збору даних є ретельний огляд літератури. Це включає пошук джерел: будуть розглянуті академічні журнали, книги, онлайн-публікації тощо. Для пошуку наукових робіт будуть використані пошукові системи, такі як IEEE Xplore, ACM Digital Library та Google Scholar. Критерії відбору: Література буде обрана на основі релевантності автоматизації DevOps, дати публікації (для забезпечення актуальності) та достовірності джерела.

Щоб переконатися в ретельному охопленні літератури, буде проведено пошук у кількох інтернет-базах даних. JSTOR та ScienceDirect є важливими базами даних. Маючи доступ до широкого спектру статей для дослідження, ці бази даних відомі своїм ґрунтовним висвітленням академічної літератури з технологій, програмної інженерії та суміжних тем. Крім того, нерецензована література та галузеві висновки будуть зібрані шляхом звернення до джерел «сірої» літератури, таких як матеріали конференцій, технічні звіти та галузеві публікації.

Будуть встановлені чіткі критерії включення та виключення, які керуватимуть відбором досліджень для систематичного огляду. Критерії включення вимагатимуть, щоб дослідження зосереджувалися на ролі та впливі автоматизації в практиці DevOps, надавали емпіричні докази або теоретичні висновки, а також були опубліковані в рецензованих журналах або на академічних конференціях. Критерії виключення виключатимуть дослідження, які не мають відношення до теми дослідження, не мають емпіричних даних чи теоретичної бази, або опубліковані до 2015 року або після 2024 року. Крім того, дослідження, недоступні англійською мовою, будуть виключені для забезпечення узгодженості мовної інтерпретації.

Назви та анотації визначених досліджень будуть перевірені на відповідність дослідницьким цілям. Потім будуть отримані та переглянуті повнотекстові

версії потенційно релевантних досліджень на основі критеріїв включення та виключення. Будь-які розбіжності під час процесу відбору будуть вирішені шляхом обговорення для забезпечення справедливого результату. Для прозорого документування кожного етапу процесу відбору буде використана блок-схема PRISMA.

Для методичної збору відповідних даних з обраного дослідження буде створено єдину форму вилучення даних. Важливими точками даних, які необхідно вилучити, є рік публікації, автор(и), цілі дослідження, методологія, важливі результати та наслідки. Застосування систематичного підходу до вилучення даних забезпечує надійність та узгодженість в отриманні важливих даних з обраних досліджень, що полегшує подальший аналіз. Крім того, два рецензенти вилучатимуть дані незалежно один від одного, а будь-які суперечки вирішуватимуться шляхом консенсусу або, за необхідності, шляхом обговорення з третім рецензентом.

Використовуючи відповідні інструменти оцінки якості, якість включених досліджень буде оцінена критично. Методологічна точність та достовірність результатів якісних досліджень будуть оцінені за допомогою контрольного списку Програми критичної оцінки (CASP). Для оцінки методологічної якості та ризику систематичної помилки в кількісних дослідженнях буде застосовано шкалу Ньюкасл-Оттава (NOS). Щоб гарантувати достовірність результатів огляду, дослідження, які вважаються низькоякісними або дуже схильними до систематичної помилки, не будуть включені до остаточного аналізу.

Для подальшої оцінки впливу якості дослідження на загальні результати та висновки систематичного огляду буде проведено аналіз чутливості.

Метою емпіричного дослідження цього систематичного огляду є розуміння функції та значення автоматизації в методах DevOps шляхом поєднання та оцінки результатів кількох обраних досліджень. У цій роботі зроблено спробу надати комплексне уявлення про те, як автоматизація впливає на розробку програмного забезпечення та операційну ефективність у контексті DevOps, розглядаючи фактичні дані та теоретичні концепції. Цей емпіричний аналіз має на меті виявити поширені закономірності, тенденції розвитку та ключові проблеми, з якими стикаються підприємства під час інтеграції автоматизації у свої процеси

DevOps, шляхом вивчення тематичних досліджень, опитувань та експериментальних досліджень. Крім того, метою цього емпіричного дослідження є уточнення фундаментальних процесів, що стимулюють інтеграцію автоматизації в DevOps, та тактик, які компанії використовують для оптимізації її переваг для доставки програмного забезпечення та операційної досконалості. Загалом, це емпіричне дослідження сприяє глибшому розумінню ролі автоматизації в практиці DevOps, інформуючи про прийняття стратегічних рішень та сприяючи інноваціям у розробці програмного забезпечення та управлінні операціями.

Метод аналізу охоплює різні підходи до синтезу та інтерпретації результатів вибраних досліджень.

Тематичний аналіз визначає повторювані теми, закономірності та ідеї, пов'язані з роллю та впливом автоматизації в практиці DevOps. Цей підхід включає кодування якісних даних для виявлення ключових концепцій та взаємозв'язків, таких як інтеграція інструментів автоматизації, вплив на співпрацю між командами розробки та експлуатації, а також наслідки для розробки програмного забезпечення та операційної ефективності.

Кількісний аналіз оцінює тенденції, кореляції та статистичну значущість ролі та впливу автоматизації в практиці DevOps. Цей аналіз може включати описову або висновкову статистику, що надає об'єктивне уявлення про вплив автоматизації на процеси розробки програмного забезпечення, частоту розгортання, час виконання та середній час відновлення.

Синтезовані якісні та кількісні результати надають комплексний огляд ролі та впливу автоматизації в практиці DevOps. Поєднуючи результати обох аналізів, цей підхід підвищує валідність та надійність результатів, дозволяючи отримати більш цілісне розуміння того, як автоматизація впливає на розробку програмного забезпечення та операційні практики в контексті DevOps.

### 2.4 Висновки по розділу

У другому розділі було досліджено методи та інструменти автоматизації DevOps-процесів. Проведено аналіз популярних платформ та інструментів

DevOps, які забезпечують автоматизацію етапів розробки, тестування, доставки й супроводу програмного забезпечення. Особливу увагу приділено концепціям безперервної інтеграції та безперервного розгортання (CI/CD), а також підходу Infrastructure as Code, що дозволяє автоматизувати управління інфраструктурою. Розглянуто практичні аспекти впровадження DevOps у процес розробки програмного забезпечення, визначено основні переваги та особливості інтеграції DevOps-практик у сучасні IT-проєкти. У результаті дослідження сформовано комплексний підхід до автоматизації життєвого циклу ПЗ із використанням сучасних DevOps-рішень.

# РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ DEVOPS-ПРАКТИК

## 3.1 Життєвий цикл DevOps та його автоматизація

DevOps розглядає повний життєвий цикл розробки програмного забезпечення (Software Development Lifecycle — SDLC), розбиваючи його на чотири основні фази, які, у свою чергу, охоплюють шість ключових областей, як показано на рисунку 3.1:

I. Планування та вимірювання (Plan & Measure)

II. Розробка та тестування (Develop & Test)

III. Випуск та розгортання (Release & Deploy)

IV. Моніторинг та оптимізація (Monitor & Optimize)

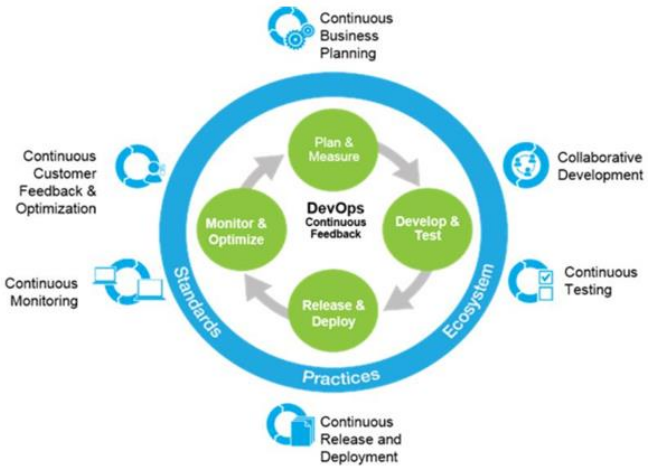


Рисунок 3.1 - Область застосування DevOps у межах SDLC

Безперервне планування, вимірювання та інтеграція бізнес-стратегії й відгуків клієнтів у життєвий цикл розробки.

Безперервне бізнес-планування забезпечує узгодження інвестиційних рішень протягом усього їхнього життєвого циклу з потребами організації. Воно застосовує принципи Lean, щоб команди починали з малого, чітко визначаючи очікувані результати та ресурси, необхідні для перевірки бізнес-бачення та

цінності. Постійно адаптуючись і коригуючи плани на основі реального прогресу та вимог клієнтів, команди отримують можливість швидко змінювати напрямок [9].

Підтримується продуктовий беклог (product backlog), який розбивається на менші релізи та етапи відповідно до пріоритетів, визначених зацікавленими сторонами.

Основні складові безперервного бізнес-планування:

- Управління вимогами (Requirements Management)
- Управління змінами (Change Management)
- Планування проєкту та управління роботами (Project Planning and Work Management)
- Планування ітерацій (Iteration Planning)
- Моніторинг прогресу та звітність (Progress Monitoring and Reporting)

Колаборативна розробка та безперервна інтеграція (Collaborative Development and Continuous Integration). Колаборативна розробка об'єднує бізнес, розробку та QA-організації для швидкої доставки інноваційних і якісних рішень користувачам. Вона кардинально змінює спосіб створення додатків, щоб досягати реальних бізнес-результатів.

Технологічні засоби DevOps формують можливості DevOps шляхом автоматизації різних рутинних і не рутинних завдань [10]. Автоматизація допомагає досягти безперервної доставки та розгортання, надаючи DevOps-пайплайн для доставки всіх змін у виробниче середовище [11].

Безперервне тестування (Continuous Testing). Використовує високий ступінь автоматизації для значного скорочення часу та зусиль, необхідних для отримання якісних результатів [12].

Усуває «вузькі місця» тестування завдяки віртуалізації залежних сервісів. Спрощує створення віртуалізованих тестових середовищ. Безперервне тестування також знижує витрати на розгортання та підтримку різних тестових середовищ.

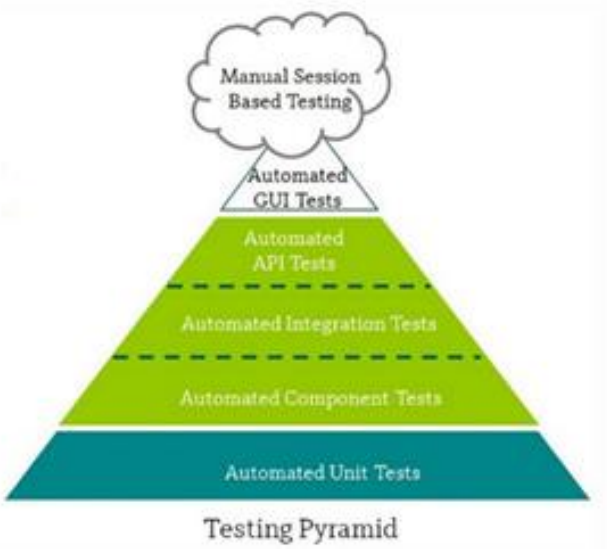


Рисунок 3.2 - Піраміда автоматизації тестування

Більшість активностей тестування програмного забезпечення базуються на практиці тест-driven development (розробка через тестування) [13], яка підтримується практиками безперервної інтеграції. Розробники зобов'язані писати модульні тести для частої інтеграції свого коду [14]. Ці тести виконуються повторно в автоматизованому режимі за допомогою безперервної інтеграції, як показано на рисунку 3.2.

Безперервний випуск та розгортання (Continuous Release & Deployment). Безперервне розгортання має низку важливих переваг. Воно дозволяє отримувати швидкий зворотний зв'язок від клієнтів і користувачів, забезпечує часті та надійні релізи, що підвищує задоволеність клієнтів [15], довіру та якість продукту [16]. Крім того, команди отримують можливість виконувати самостійне розгортання в середовищах Dev/Test за потреби, постійно інтегруючи інновації, керовані програмним забезпеченням.

Технологічні засоби:

- Управління випусками (Release Management)
- Автоматизація збірки (Build Automation)
- Автоматизація розгортання (Deployment Automation)

Безперервний моніторинг (Continuous Monitoring). Безперервний моніторинг автоматизує та оптимізує можливість постійно контролювати та керувати продуктивністю й доступністю додатків та інфраструктури. Він показує,

наскільки добре працюють системи, і чи потрібне будь-яке коригування [17]. Правильний вибір інструменту визначає рівень точності, якого можна досягти в системі


Рисунок 3.3 - Високорівневий цикл збірки та розгортання до продакшн-середовища

.Основні області моніторингу в проєкті з впровадженим DevOps:

- Управління продуктивністю додатків (Application Performance Management)
- Аналітика IT-операцій (IT Operations Analytics)
- Управління операційними проблемами (Operational Problem Management)

Для ефективного моніторингу необхідно використовувати журнали подій (logging) рис. 3.3. Механізм логування працює на кількох рівнях і має бути налаштований таким чином, щоб за потреби можна було проводити детальний аналіз журналів [18].

Аналіз логів забезпечує якісний зворотний зв’язок командам розробки, а потім дозволяє виконати глибокий аналіз проблеми для виявлення її корінної причини. Результати аналізу логів допомагають командам розробки та операцій краще підготуватися до майбутнього.

Безперервний зворотний зв’язок від клієнтів та оптимізація (Continuous Customer Feedback and Optimization). Постійне отримання відгуків клієнтів щодо їхнього досвіду використання дає значно глибші інсайти, які стають основою для покращення прийняття рішень та оптимізації. Швидкий і ранній зворотний

зв’язок від систем допомагає економити час і постійно підвищувати якість досвіду кінцевого користувача [19].

Модель зрілості DevOps сприяє розвитку через безперервне навчання команд розробки та операцій, як показано на рисунку 3.4. Визначені можливості та навички гарантують підвищену здатність працювати з масштабними та складними завданнями.


Рисунок 3.4 - Безперервний зворотний зв’язок

Information Technology Infrastructure Library (ITIL) - це набір найкращих практик управління IT-сервісами, який акцентує увагу на узгодженні IT-сервісів з потребами бізнесу.

Ступінь впровадження DevOps також можна вимірювати за допомогою моделей зрілості за різними критеріями [20].

Я провів дослідження різних інструментів DevOps, які використовуються у провідних організаціях для впровадження DevOps на різних етапах доставки програмного забезпечення.

Рисунок 3.5 нижче показує перелік різних інструментів, що застосовуються в проєктах роздрібної торгівлі (Retail), банківської справи (Banking) та автомобільної промисловості (Automobile). Кожен із цих інструментів, який обслуговує одну з фаз DevOps, унікальний у своєму роді та пропонує різні функції, забезпечуючи оптимальне використання інструментів для конкретних цілей [21].

Ці інструменти використовуються для проєктування, збірки, розгортання, тестування, моніторингу, управління та експлуатації програмного забезпечення та систем, з'єднаних в єдиний інтегрований пайплайн.

Інструменти загалом поділяються на комерційні та відкриті (Open Source). Зі зростанням спільноти Open Source, яка пропонує сучасні функціональні можливості, багато підприємств перейшли з комерційних інструментів на відкриті також через високу вартість ліцензій.

Jenkins — це сервер безперервної інтеграції та автоматизації з відкритим кодом, який працює на основі архітектури плагінів [22]. Він має можливість інтегрувати різноманітні інструменти, забезпечуючи ланцюжок безперервної інтеграції / безперервної доставки (Continuous Integration / Continuous Delivery — CI/CD toolchain). Jenkins може виконувати збірку, розгортання та тестування на кількох платформах [23].

З появою хмарних обчислень (Cloud Computing) з'явилися різноманітні сервіси для виконання завдань автоматизації, які є хмарно-нативними (cloud native) і підтримують платформи, розроблені в хмарі [24]. Додатковою перевагою хмарних провайдерів є вбудовані сервіси безпеки. Вартість цих сервісів залежить від обсягу використання та навантаження на платформу.

### 3.2 Виклики та проблеми впровадження DevOps

DevOps - це набір практик, які поєднують процеси розробки та експлуатації (Development and Operations) . Для виконання функцій поєднання та інтеграції DevOps потребує набору інструментів. Іншими словами, DevOps - це єдина команда, яка відповідає за розробку, тестування та експлуатацію. У DevOps весь цикл створення продукту не переривається на жодному етапі.

DevOps має чотири виміри:

1. Колаборація (Collaboration)
2. Автоматизація (Automation)
3. Вимірювання (Measurement)
4. Моніторинг (Monitoring)

DevOps є розширенням гнучкої методології розробки програмного забезпечення (Agile). DevOps акцентує увагу на безперервній доставці програмного забезпечення поряд із безперервною інтеграцією. Автоматизація також відіграє ключову роль у зменшенні затримок при випуску продукту. DevOps не лише покращує колаборацію та комунікацію, а й забезпечує швидку та безперервну доставку, регулярні оновлення, підвищення надійності тощо.

Хмарні обчислення відіграють важливу роль у безперервній доставці (Continuous Delivery. Хмарні інструменти допомагають усунути розриви між потребою та доставкою, а також забезпечують швидший зворотний зв'язок . У разі швидкого процесу розгортання тестування має бути автоматизованим, щоб зменшити затримки.

У дослідженні, проведеному серед 19 компаній, було виявлено, що 11 із 19 компаній застосовують стратегію безперервного розгортання. Традиційні методи, такі як інкрементальний/ітеративний та ad-hoc підходи, не задовольняли потреби компаній з розробки програмного забезпечення.

У роботі розглянуто, як підготувати компанію до впровадження безперервної доставки. Було зазначено, що безперервну доставку слід впроваджувати невеликими кроками, а повторювані завдання автоматизувати. Переваги впровадження безперервної доставки включають: прискорення виходу на ринок, створення правильного продукту, підвищення продуктивності та ефективності, надійні релізи, покращення якості продукту та задоволення клієнтів. Інфраструктура як код (Infrastructure as Code) сприяє швидким релізам.

До появи DevOps у компаніях не існувало спільної інфраструктури для розробки програмного забезпечення, тому співробітники працювали кожен у своєму середовищі. Через це процес доставки затримувався через збільшений час інтеграції.

Сам DevOps уособлює інтеграцію між командами розробки та експлуатації. DevOps забезпечує безперервну інтеграцію всіх процесів, пов'язаних із розробкою продукту, тому всі процеси виконує одна команда протягом усього циклу.

Недостатня комунікація є ключовою проблемою, яка зумовлює необхідність впровадження DevOps. DevOps дозволяє компаніям з розробки ПЗ швидко додавати нові функції та постійно вдосконалювати продукт на основі зворотного зв'язку. DevOps інтегрує не лише команди, а й інструменти, які використовуються на різних етапах розробки продукту .

У роботі було запропоновано нову рамкову модель для впровадження безперервної інтеграції та розгортання в існуючий цикл продукту Scrum. Версії інструментів програмного забезпечення повинні підтримуватися сумісними з іншими інструментами. Стратегію контролю версій було запропоновано для керування різними артефактами, такими як вихідний код, конфігураційні файли, скрипти розгортання та бінарний код програми.

Автоматизація є необхідною для безперервної інтеграції. Для відстеження змін використовуються інструменти, такі як Git-репозиторій. HARNESS — це багатосторонній дослідницький проєкт, метою якого є інтеграція неоднорідних ресурсів за допомогою технік контролю версій.

У роботі було запропоновано новий інструмент під назвою Filling Gap (FG), який усуває розрив між командами розробки та експлуатації та забезпечує швидкий зворотний зв'язок щодо продуктивності для команди розробки.

Хмара є ключовим елементом методології DevOps. Методологія, інструменти та культура Cloud DevOps показані на рисунку 3.5.

Хмарна екосистема дозволяє мати велику кількість взаємопов'язаних компонентів, які легко доступні та керовані. Було запропоновано спрощену хмарну екосистему. Серверлес-модель (serverless model) використовує хмару у своїй архітектурі . Якісні дані показали, що практики DevOps сильно залежать від нової моделі хмарних обчислень.

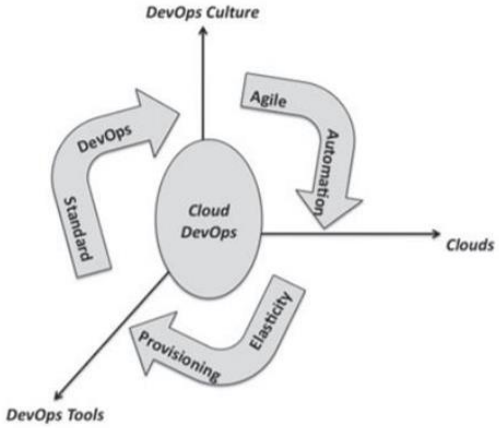


Рисунок 3.5 - Cloud DevOps

Для хмарних додатків використовувалися легковагові мови програмування. Хмара сприяє ітеративній розробці програмного забезпечення, моніторингу запущених додатків на стороні клієнта та отриманню зворотного зв'язку щодо процесу. Використання хмарного середовища також підвищує безпеку програмних додатків .

Артефакти DevOps класифікуються як вузлово-орієнтовані (Node centric) та середовищно-орієнтовані (Environmental centric) артефакти. Для забезпечення інтероперабельності артефактів у хмарних додатках використовується TOSCA.

DevSecOps або SecDevOps — терміни, що пов'язані із захищеним DevOps. У роботі було запропоновано модель безперервної безпеки, яка використовує програмне забезпечення з відкритим кодом у хмарі для забезпечення безпеки протягом усього циклу розробки продукту.

Практики безпеки, які застосовуються в DevOps, включають:

1. Автоматизовані дії: автоматизоване тестування, моніторинг, code review, програмно-визначений фаєрвол та ліцензування програмного забезпечення.
2. Посилена колаборація між командами розробки та безпеки.
3. Неавтоматизовані дії з безпеки: аналіз вимог безпеки, налаштування конфігурацій безпеки, застосування політик безпеки, дизайн-рев'ю, валідація введення, аналіз ризиків тощо.

У роботі  було запропоновано динамічну модель для усунення проблем безпеки як у процесах розробки, так і в операційній діяльності багатохмарних додатків. Модель була перевірена на реальному додатку. Фреймворк динамічної моделі MUSA показано на рисунку 3.6.

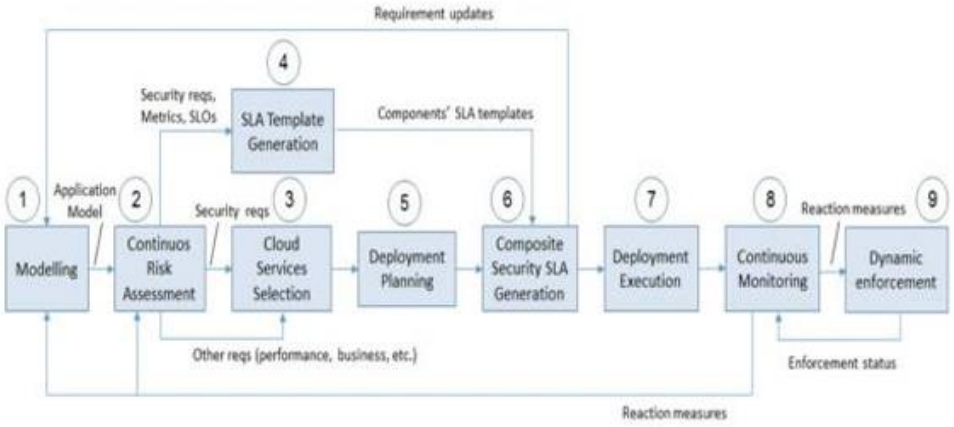


Рисунок 3.6 - Фреймворк MUSA DevOps

Хмарні опції безпеки включають серверлес-обчислення, інфраструктуру як код та централізацію безпеки.

Деякі заходи безпеки, які слід виконувати в DevOps: збирання вимог безпеки, моделювання загроз, конфігурація середовища, статичний аналіз безпеки, code review з акцентом на безпеку, пентестинг, тестування середовища та огляд безпеки.

У разі швидкого розгортання ручне тестування стає неможливим, тому тестування має бути автоматизованим. Остаточне завдання автоматизації — виявлення помилок у системі та їх виправлення.

Окрім тестування, автоматизація допомагає масштабувати продукти . У запропонованій моделі робочий процес поділяється на backend і frontend. Модель намагається автоматизувати процеси, що відбуваються протягом життєвого циклу розробки. Цикл починається зі збору даних, на основі яких будується додаток, потім результати збірки тестуються і розгортаються через хмару.

Автоматизація випуску програмного забезпечення призводить до

зниження операційних витрат, підвищення продуктивності, доступності, надійності та оптимізації продуктивності.

Роботизована автоматизація процесів (Robotic Process Automation - RPA) є перспективним методом автоматизації повторюваних завдань. Машинне навчання можна застосовувати в завданнях автоматизації, таких як тестування, забезпечення якості, виявлення відмов тощо.

Автоматизація - єдиний спосіб скоротити цикл випуску. Архітектура була побудована на основі надійних існуючих інструментів для автоматизованого тестування та забезпечення якості. Основні будівельні блоки запропонованої архітектури: хмарна платформа, віртуальна машина управління та інструмент управління конфігурацією.

Не всі процеси можна автоматизувати - деякі мають виконуватися вручну. Автоматизація також забезпечує швидкий зворотний зв'язок від клієнтів.

Інструменти потрібні для розробки та інтеграції процесів. Вони класифікуються як інструменти збірки та інструменти безперервної інтеграції. Серед інструментів для покращення практик DevOps: JIRA, GIT, Jenkins та Docker продемонстрованона рис. 3.7.

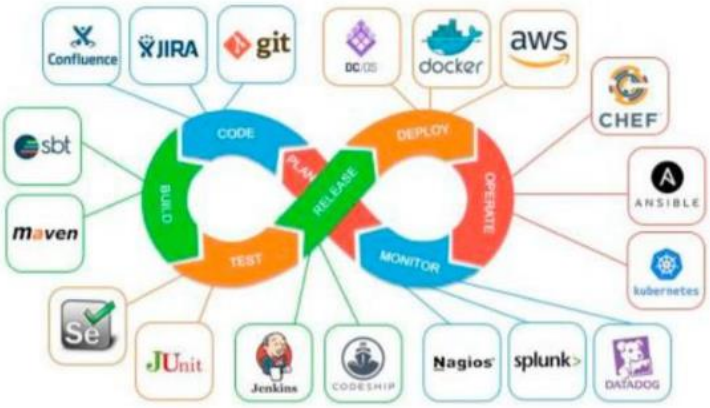


Рисунок 3.7 - Інструменти, що використовуються в DevOps

Інструменти повинні використовуватися відповідно до ієрархії. Chef Cookbooks, Puppet Modules, Saltstack modules, Docker images, Juju charms, Bundles та templates - це приклади інструментів управління конфігурацією.

Раніше для зберігання конфігураційних даних і відстеження змін використовувався репозиторій Mercurial.

Інструмент управління проєктами необхідний IT-компанії для керування даними різних проєктів. Одним із таких інструментів є GZ-Agile Project Management Consolidator, який зберігає та відстежує діяльність DevOps і Agile.

У системі виявлення шахрайства, розробленій Netfective Technology, для оцінки метрик використовувався SimTool.

Серед хмарних сервісів, що використовуються в DevOps: Amazon Web Services (AWS), Microsoft Azure, IBM Cloud.

Для уніфікації процесу розробки програмного забезпечення встановлення та конфігурація не повинні бути схильними до помилок. Інструменти конфігурації та управління IT-системами використовуються для впровадження DevOps, оскільки вони автоматизують процес конфігурації системи. Конфігурація системи керується як набір правил, організованих у модулі та класи.

ITIL Framework також є інструментом управління IT-сервісами, що застосовується в DevOps. Архітектуру ITIL показано на рисунку 3.8.


Рисунок 3.8 - Архітектура ITIL

Впровадження DevOps в IT-компанії є непростим завданням. У компанії важко змінити організаційну культуру, а практики DevOps не підходять для всіх обставин.

Виклики можна класифікувати як: відсутність обізнаності, відсутність підтримки, технологічна здійсненність та адаптація до змін. Крім організаційно-культурних викликів, існують також: впровадження в існуючі процеси,

архітектурні виклики, відсутність автоматизації безперервного тестування та застарілі системи (Legacy systems).

Інфраструктура також відіграє важливу роль у впровадженні DevOps - вона повинна бути сумісною та легковаговою.

Виклики, виявлені за допомогою методу Fuzzy TOPSIS: неоднорідність даних, інтеграція даних, помилки та неузгоджені дані, орфографічні помилки при введенні, відсутність інформації, трасувальність даних, гармонізація даних, візуалізація даних тощо.

Конфлікти між командами розробки та експлуатації виникають під час розгортання нових функцій і виправлення проблем. Команда розробки створює нову функцію, не знаючи проблем зі старою версією. Команда операцій повинна виправляти проблеми разом із розробниками, але розробники вже готові розгортати нову версію.

Деякі перешкоди в розробці ПЗ для високо регульованого середовища:

1. оператор не знає коду;
2. усі артефакти зберігаються в одному ізольованому контейнері;
3. проєкт повинен бути схвалений перед початком;
4. слабка колаборація.

5. Запропоновані моделі

Трьохетапну модель впровадження DevOps було запропоновано. Модель була оцінена шляхом практичного впровадження. Для успішного впровадження DevOps потрібні взаємозв’язки між категоріями: гнучкість, автоматизація, культура колаборації, безперервне вимірювання, забезпечення якості, стійкість, обмін інформацією та прозорість.

Підхід iObserve для подолання викликів впровадження DevOps розглянуто. Він базується на циклі керування MAPE (Monitor, Analyse, Plan, Execute). Фреймворк підходу iObserve показано на рисунку 3.9.

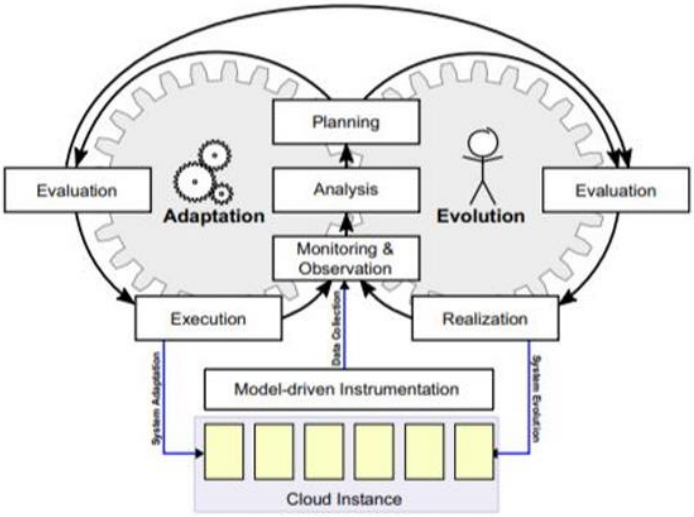


Рисунок 3.9 - Модель iObserve

Модель трансферу технологій, показану на рисунку 3.10, було запропоновано в  для масштабованих багатосторонніх організацій, академії та промисловості. Було зазначено, що продуктивність DevOps зростає зі збільшенням інтеграції між DevOps і управлінням знаннями. У компаніях слід заохочувати добровільний обмін знаннями.

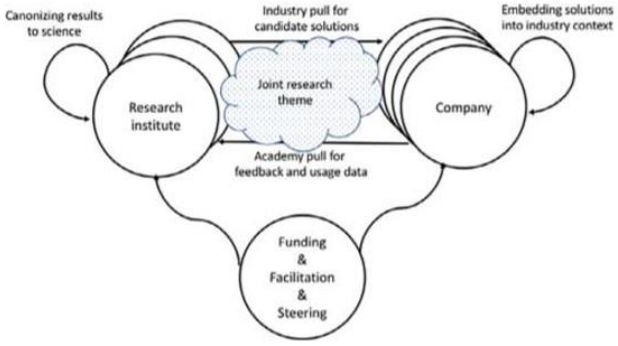


Рисунок 3.10 - Модель трансферу технологій

Модель зрілості DevOps було запропоновано. Вона має 5 рівнів зрілості та 4 виміри оцінки. Модель базується на CMMI і показана на рисунку 3.11.

Було запропоновано концепцію Composable DevOps architecture.

Компонована архітектура передбачає розбиття системи на менші фрагменти з подальшою інтеграцією всього пайплайну розробки. Composable DevOps - це ітеративний спосіб забезпечення колаборації між розробкою та операціями.

Під час розробки програмного забезпечення за допомогою нових методик якість готового продукту повинна підтримуватися на належному рівні. Забезпечення якості продукту покращується завдяки використанню автоматизованого DevOps-пайплайну. Релізи випускаються частіше для вдосконалення функцій продукту.

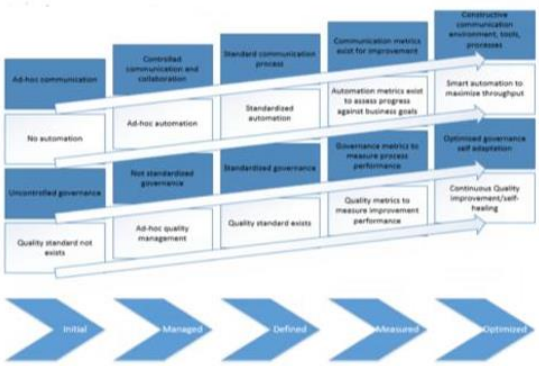


Рисунок 3.11 - Модель зрілості DevOps

Набір метрик для оцінки практик DevOps було розглянуто. Запропоновані організаційні метрики для відстеження безперервного покращення: Truck-Factor, Socio-Technical Congruence, Core-Periphery Ratio, Community Member Turnover та Smelly-Quitters.

Технічні метрики включають: кількість рядків коду, зв’язність між класами об’єктів (Coupling Between Object Classes), процес зміни коду, фактори, пов’язані з розробниками, показники підтримуваності під час виконання та фактори операцій.

У системі виявлення шахрайства для оцінки метрик використовувався Simtool. Для оцінки якості в DevOps застосовувався метод Fuzzy TOPSIS.

### 3.3 Аналіз результатів впровадження DevOps

У цьому розділі всебічно аналізуються результати дослідження, що стосуються дослідницьких питань, що лежали в основі цього дослідження. Він поєднує поглиблений аналіз зібраних даних із критичною оцінкою результатів,

прагнучи забезпечити цілісне розуміння ролі автоматизації в практиці DevOps. Мета полягає в систематичному аналізі того, як автоматизація впливає на процеси DevOps, включаючи ефективність, масштабованість та командну співпрацю, а також у визначенні найкращих практик та проблем, пов'язаних із впровадженням автоматизації.

У цьому розділі обговорюються результати дослідження у порівнянні з існуючою літературою, підкреслюючи їх відповідність попереднім дослідженням та унікальний внесок цього дослідження. У розділі будуть дані відповіді на основні дослідницькі питання та синтезовано значення цих результатів для фахівців та організацій DevOps. До кінця цього розділу буде надано чіткий виклад ключових висновків, що пропонуватиме практичні рекомендації та визначатиме потенційні напрямки для майбутніх досліджень.

Аналіз даних виявив кілька ключових тем щодо впливу автоматизації на практики DevOps. Ці теми включають підвищення ефективності, покращення масштабованості та сприяння командній співпраці.

Підвищення ефективності: Автоматизація значно підвищує ефективність процесів DevOps, мінімізуючи ручне втручання та оптимізуючи робочі процеси. Дані показали, що інструменти автоматизації, такі як конвеєри безперервної інтеграції (CI) та безперервної доставки (CD), скорочують час випуску оновлень програмного забезпечення. Це узгоджується з дослідженнями, які підкреслюють, як автоматизація прискорює цикли розгортання, скорочуючи час виходу на ринок.

Покращення масштабованості: Інтеграція автоматизації дозволяє організаціям безперешкодно масштабувати свої DevOps-операції. Фреймворки автоматизації, такі як Інфраструктура як код (IaC), забезпечують послідовне та повторюване розгортання, що спрощує керування великомасштабними середовищами. За даними автоматизовані інструменти керування налаштуванням та конфігурацією (наприклад, Terraform, Ansible) допомагають організаціям обробляти зростаючі робочі навантаження з мінімальними витратами ресурсів.

Сприяння командній співпраці: Ще однією ключовою визначеною темою є роль автоматизації у покращенні співпраці між командами розробки та

операцій. Дані свідчать про те, що автоматизація зменшує точки тертя, встановлюючи стандартизовані процеси, сприяючи кращій комунікації та спільній відповідальності. Це підтверджує висновки які стверджують, що автоматизація руйнує ізоляцію та сприяє культурі спільної відповідальності в умовах DevOps.

Ці теми демонструють, що автоматизація є ключовою для оптимізації робочих процесів DevOps, підвищення операційної ефективності та сприяння спільній роботі.

В аналізі додатково досліджено різні інструменти автоматизації, що зазвичай використовуються в DevOps, та їхній вплив на розробку програмного забезпечення та IT-операції.

Інструменти безперервної інтеграції (CI): Jenkins, GitLab CI та CircleCI були визначені як популярні інструменти CI, що сприяють автоматизованим процесам збірки та тестування. Ці інструменти дозволяють раннє виявляти проблеми інтеграції, тим самим покращуючи якість коду. Однак, керування складними конвеєрами збірки та забезпечення масштабованості є проблемами, на які звернули увагу.

Управління конфігурацією та інфраструктура як код (IaC): Такі інструменти, як Ansible, Puppet та Chef, широко використовуються для автоматизації управління конфігурацією та забезпечення інфраструктури. Ці інструменти підтримують узгодженість у різних середовищах, що є важливим для підтримки стабільності у виробничих системах. Незважаючи на їхні переваги, інтеграція цих інструментів з існуючими застарілими системами може бути складною через проблеми сумісності.

Моніторинг та спостереження: Prometheus, Grafana та стек ELK є важливими для автоматизованого моніторингу та спостереження. Ці інструменти дозволяють проактивно керувати інцидентами, надаючи інформацію про продуктивність системи в режимі реального часу. Однак командам часто потрібна допомога в управлінні великими обсягами генерованих даних, що вимагає автоматизованих систем оповіщення та виявлення аномалій.

Порівняння інструментів: інструменти неперервної інтеграції (CI) зосереджені на покращенні інтеграції коду, інструменти інфраструктури контролю

(IAC) наголошують на управлінні інфраструктурою, а інструменти моніторингу підвищують надійність системи. Вибір інструментів залежить від пріоритетів організації — чи є фокусом швидкість, стабільність чи видимість. Збалансований підхід, що поєднує CI, IaC та інструменти моніторингу, може оптимізувати конвеєр DevOps.

Впровадження автоматизації в середовищах DevOps має свої труднощі. Аналіз виявив кілька перешкод, з якими стикаються організації під час впровадження автоматизації:

Проблеми інтеграції інструментів: Інтеграція різноманітних інструментів автоматизації в цілісний робочий процес DevOps може бути складною, особливо для організацій з існуючими застарілими системами. Проблеми сумісності між новими інструментами автоматизації та існуючими платформами можуть перешкоджати безперебійному впровадженню, вимагаючи додаткових зусиль з налаштування.

Опір команди: Опір змінам є значною перешкодою, особливо під час впровадження нових методів автоматизації, які змінюють існуючі робочі процеси. Дані показали, що відсутність належного навчання та обізнаності серед членів команди часто призводить до небажання впроваджувати автоматизацію.

Проблеми масштабованості: Хоча автоматизація підвищує масштабованість, управління автоматизованими процесами у великомасштабних середовищах може бути вимогливим. Складність масштабування конвеєрів CI/CD та автоматизації інфраструктури часто призводить до проблем, пов'язаних з продуктивністю, розподілом ресурсів та обслуговуванням системи.

Проблеми безпеки: Ще однією зазначеною проблемою є наслідки автоматизації для безпеки. Автоматизація процесів розгортання та інфраструктури створює ризики, якщо заходи безпеки не інтегровані в робочі процеси автоматизації. Організації повинні вирішувати потенційні вразливості, впроваджуючи автоматизовані методи безпеки, такі як постійне сканування вразливостей та перевірки на відповідність.

Аналіз показує, що автоматизація є трансформаційним фактором у підвищенні ефективності, масштабованості та співпраці практик DevOps.

Автоматизація спрощує повторювані та ручні завдання, такі як інтеграція коду, тестування та розгортання, значно скорочуючи час виходу на ринок релізів програмного забезпечення. Інтеграція інструментів автоматизації, таких як Jenkins, GitLab CI та Ansible, забезпечує безперервну інтеграцію (CI) та безперервну доставку (CD), що, у свою чергу, прискорює процес розробки та забезпечує стабільний, високоякісний результат.

Більше того, автоматизація сприяє масштабованості, використовуючи фреймворки «Інфраструктура як код» (IaC), які дозволяють організаціям керувати великомасштабними середовищами з мінімальним ручним втручанням. Такі інструменти, як Terraform та Chef, допомагають автоматизувати надання інфраструктури, тим самим підтримуючи швидке масштабування та оптимізацію ресурсів. Спільний характер автоматизації також руйнує розмежування між командами розробки та експлуатації, сприяючи культурі спільної відповідальності та постійного вдосконалення. Як зазначали автоматизація покращує співпрацю в команді, що призводить до ефективніших робочих процесів та покращення загальної продуктивності в середовищах DevOps [33].

У дослідженні було визначено кілька найкращих практик, які організації можуть застосувати для ефективного впровадження автоматизації у свої DevOps-процеси:

Застосуйте стратегію поступової автоматизації: Замість того, щоб автоматизувати все одразу, організаціям слід застосовувати поетапний підхід, починаючи з найбільш трудомістких та схильних до помилок завдань [29]. Такий підхід дозволяє командам поступово адаптуватися та зменшує опір змінам.

Використовуйте інструменти безперервної інтеграції (CI) та безперервної доставки (CD): інструменти CI/CD, такі як Jenkins, GitLab CI та CircleCI, мають вирішальне значення для автоматизації процесів збірки, тестування та розгортання. Ці інструменти дозволяють виявляти проблеми на ранній стадії, покращувати якість коду та пришвидшувати цикли випуску.

Впровадження інфраструктури як коду (IaC): Організаціям слід використовувати інструменти IaC, такі як Ansible та Terraform, щоб забезпечити узгодженість між середовищами розробки, тестування та виробництва. IaC сприяє

автоматизованому наданню та управлінню інфраструктурою, зменшуючи кількість помилок, внесених вручну, та покращуючи масштабованість.

Пріоритет автоматизованого тестування: інтеграція платформ автоматизованого тестування, таких як Selenium та JUnit, у конвеєр CI/CD гарантує ретельну перевірку змін коду перед розгортанням. Така практика допомагає підтримувати високу якість програмного забезпечення та мінімізує кількість помилок у виробництві.

Сприяти розвитку культури співпраці: Автоматизація не повинна обмежуватися інструментами та технологіями, а також повинна охоплювати культурні аспекти. Заохочення міжфункціональної співпраці між командами розробки, експлуатації та безпеки є важливим для максимізації переваг автоматизації.

Зосередження на автоматизації безпеки: інтеграція перевірок безпеки в автоматизовані робочі процеси, такі як сканування вразливостей та аудит відповідності, допомагає зменшити ризики безпеки на ранніх етапах розробки. Ця практика відповідає принципам DevSecOps, гарантуючи, що безпека не є другорядною, а постійним аспектом конвеєра [25].

Ці найкращі практики можуть допомогти організаціям оптимізувати свої стратегії автоматизації DevOps, підвищуючи ефективність, гнучкість та стійкість.

Незважаючи на численні переваги автоматизації, кілька проблем можуть перешкоджати її впровадженню в середовищах DevOps:

Інтеграція складних інструментів: Інтеграція кількох інструментів автоматизації в єдиний робочий процес може бути складною, особливо для організацій з існуючими застарілими системами. Аналіз показав, що проблеми сумісності часто виникають під час узгодження нових рішень автоматизації з уже існуючою інфраструктурою.

Опір змінам: Однією з найсуттєвіших перешкод для впровадження автоматизації є опір членів команди, які звикли до традиційних робочих процесів. Небажання використовувати нові інструменти та процеси може перешкоджати успішному впровадженню стратегій автоматизації. Навчання та ініціативи з управління змінами є важливими для подолання цього опору.

Проблеми масштабованості: Хоча автоматизація призначена для підвищення масштабованості, управління автоматизованими процесами у великомасштабних середовищах створює певні труднощі. До них належать оптимізація продуктивності, управління розподілом ресурсів та забезпечення надійності конвеєрів CI/CD у сценаріях високого навантаження.

Ризики безпеки та відповідності: Автоматизація процесів розгортання та інфраструктури створює потенційні вразливості безпеки, якщо ними не керувати належним чином. Організації повинні інтегрувати заходи безпеки, такі як автоматизоване сканування вразливостей, виявлення загроз та механізми реагування на інциденти, у свої автоматизовані робочі процеси для зменшення ризиків.

Прогалини у навичках та вимоги до навчання: Ефективне впровадження автоматизації в DevOps вимагає спеціалізованих навичок [28]. Багатьом організаціям потрібна допомога у підвищенні кваліфікації своїх співробітників для використання передових інструментів та фреймворків автоматизації. Програми постійного навчання та сертифікації мають вирішальне значення для подолання цієї перешкоди.

Вирішення цих проблем вимагає стратегічного підходу, який включає вибір правильних інструментів, інвестування в навчання персоналу та сприяння культурі, що підтримує автоматизацію. Розуміючи та пом'якшуючи ці проблеми, організації можуть розкрити весь потенціал автоматизації у своїх DevOps-практиках.

Результати цього дослідження мають значне значення для фахівців з DevOps, дослідників та організацій, які прагнуть оптимізувати свої процеси DevOps за допомогою автоматизації. Використовуючи автоматизацію, організації можуть підвищити ефективність та масштабованість розробки програмного забезпечення та IT-операцій, а також сприяти спільній та інноваційній організаційній культурі. Наслідки цих висновків обговорюються нижче.

Для фахівців з DevOps дослідження підкреслює критичну роль автоматизації в оптимізації робочих процесів та пришвидшенні розробки програмного забезпечення. Впровадження таких інструментів, як Jenkins для безперервної інтеграції (CI), Ansible для управління конфігураціями та Terraform для

інфраструктури як коду (IaC), може значно зменшити кількість помилок, виконуваних вручну, та покращити узгодженість розгортання. Автоматизація також дозволяє фахівцям зосередитися на завданнях вищої цінності, таких як оптимізація продуктивності системи та підвищення заходів безпеки, а не загрузнути в повторюваних ручних процесах.

Підвищена операційна ефективність: Впроваджуючи конвеєри CI/CD, фахівці можуть автоматизувати процеси тестування та розгортання, що призводить до швидших циклів випуску та скорочення часу виходу на ринок. Це узгоджується з галузевими звітами, які підкреслюють роль автоматизації у підвищенні продуктивності та операційної ефективності.

Проактивне вирішення проблем: Інструменти автоматизації з вбудованими можливостями моніторингу та сповіщень, такі як Prometheus та Grafana, дозволяють командам виявляти та вирішувати проблеми до того, як вони проактивно вплинуть на кінцевих користувачів. Це призводить до підвищення надійності системи та задоволеності клієнтів.

Розвиток навичок: Фахівців заохочують постійно підвищувати свою кваліфікацію в технологіях автоматизації, щоб залишатися конкурентоспроможними в умовах постійного розвитку DevOps-середовища. Навчання інструментам та фреймворкам автоматизації підвищує технічну майстерність і позиціонує фахівців як цінних активів для своїх організацій.

Результати дослідження відкривають нові шляхи для дослідження впливу автоматизації на практику DevOps. Хоча існуюча література здебільшого зосереджена на технічних перевагах автоматизації, все ще існують прогалини в розумінні її соціально-технічних наслідків.

Новітні технології: Дослідники можуть досліджувати, як новітні технології, такі як штучний інтелект (ШІ) та машинне навчання (МН), можуть бути інтегровані в автоматизацію DevOps для оптимізації процесів прийняття рішень, розподілу ресурсів та прогнозної аналітики.

Соціально-технічна динаміка: Дослідження визначає необхідність подальшого дослідження соціально-технічних аспектів автоматизації, таких як її вплив на динаміку команди, організаційну культуру та добробут співробітників.

Розуміння того, як автоматизація впливає на співпрацю, комунікацію та задоволення від роботи, може дати уявлення про найкращі практики впровадження автоматизації з урахуванням потреб людини.

Безпека та відповідність: Майбутні дослідження можуть дослідити наслідки автоматизації робочих процесів DevOps для безпеки, особливо у високорегульованих галузях. Досліджуючи автоматизовані заходи безпеки, такі як постійне сканування вразливостей та перевірки на відповідність, дослідники можуть надати рекомендації щодо безпечних практик автоматизації.

Результати дослідження підкреслюють стратегічну важливість автоматизації для організацій у досягненні конкурентної переваги. Автоматизація підвищує операційну ефективність та стимулює інновації, дозволяючи командам зосередитися на креативному вирішенні проблем та стратегічних ініціативах.

Оптимізація процесів DevOps: Організації можуть використовувати автоматизацію для оптимізації своїх робочих процесів DevOps, тим самим зменшуючи вартість і складність розробки програмного забезпечення. Впровадження інфраструктури як коду (IaC) дозволяє організаціям безперешкодно масштабувати свою інфраструктуру, підтримуючи зростання та гнучкість бізнесу [31]. Автоматизуючи повторювані завдання, організації можуть досягти швидших циклів розгортання, покращити якість програмного забезпечення та підвищити гнучкість реагування на зміни ринку.

Покращення динаміки та співпраці команди: Автоматизація сприяє культурі співпраці, руйнуючи розрізненість між командами розробки, операцій та безпеки. Стандартизуючи процеси, автоматизація гарантує, що всі команди відповідають своїм цілям, тим самим сприяючи прозорості та спільній відповідальності. Організації повинні заохочувати міжфункціональну співпрацю та постійні цикли зворотного зв'язку, щоб максимізувати переваги автоматизації.

Розвиток культури, орієнтованої на автоматизацію: Щоб повною мірою використати потенціал автоматизації, організаціям необхідно розвивати у своїх співробітників менталітет, орієнтований на автоматизацію [34]. Це передбачає інвестування у правильні інструменти та пріоритетність програм навчання та розвитку для розвитку знань з автоматизації. Підтримка керівництва має

вирішальне значення для просування цього культурного зрушення, оскільки вона заохочує команди використовувати автоматизацію як ключовий компонент своєї стратегії DevOps.

Вирішення соціально-технічних наслідків: Впровадження автоматизації також пов'язане з соціально- технічними викликами, такими як опір змінам та необхідність постійного навчання. Організації повинні проактивно вирішувати ці виклики, забезпечуючи належне навчання, створюючи сприятливе робоче середовище та пропагуючи культуру інновацій. Заохочення членів команди до експериментування з інструментами та методами автоматизації може сприяти залученості та підвищувати задоволення від роботи.

Міркування щодо безпеки та відповідності: Оскільки організації автоматизують свої процеси DevOps, важливо інтегрувати безпеку в робочі процеси автоматизації. Впровадження практик DevSecOps, таких як автоматизоване тестування безпеки, оцінка вразливостей та перевірки відповідності, може допомогти організаціям зменшити ризики та захистити свої системи від потенційних загроз.

Підсумовуючи, результати цього дослідження демонструють, що автоматизація — це не просто технічне вдосконалення, а стратегічний фактор трансформації діяльності організацій. Автоматизація може стимулювати сталий розвиток та інновації в середовищах DevOps шляхом оптимізації процесів, сприяння співпраці та врахування соціально-технічної динаміки. Організації, які цілісно сприймають автоматизацію, інтегруючи її у свою культуру та стратегічні цілі, ймовірно, досягнуть більшої гнучкості, стійкості та конкурентної переваги в цифрову епоху [27].

Рекомендації для майбутніх досліджень. Результати цього дослідження виявили кілька прогалин та можливостей для подальшого дослідження в галузі автоматизації DevOps. Усунення цих прогалин може поглибити наше розуміння ролі автоматизації в удосконаленні практик DevOps та сприяти постійному розвитку більш ефективних, масштабованих та безпечних процесів доставки програмного забезпечення. Нижче наведено цільові рекомендації для майбутніх досліджень, засновані на виявлених прогалинах:

Однією з нових сфер інтересу є інтеграція технологій штучного інтелекту та машинного навчання в автоматизацію DevOps. Майбутні дослідження можуть зосередитися на тому, як ці технології можуть покращити прийняття рішень, оптимізувати розподіл ресурсів та покращити прогнозну аналітику в середовищах DevOps.

Автоматизація на основі штучного інтелекту: дослідження потенціалу штучного інтелекту в автоматизації складних завдань DevOps, таких як виявлення аномалій, налаштування продуктивності та автоматизоване виправлення. Дослідження можуть включати використання алгоритмів штучного інтелекту для прогнозування системних збоїв, оптимізації конвеєрів CI/CD та забезпечення інтелектуальної автоматизації, яка адаптується до змін робочих навантажень.

Машинне навчання для постійного вдосконалення: Дослідіть, як моделі машинного навчання можна інтегрувати в процеси DevOps, щоб забезпечити постійне навчання на основі минулих розгортань, збоїв та показників продуктивності. Це може включати розробку моделей машинного навчання, які аналізують історичні дані для рекомендації оптимізації для майбутніх релізів.

Рекомендації для майбутніх досліджень. Результати цього дослідження виявили кілька прогалин та можливостей для подальшого дослідження в галузі автоматизації DevOps. Усунення цих прогалин може поглибити наше розуміння ролі автоматизації в удосконаленні практик DevOps та сприяти постійному розвитку більш ефективних, масштабованих та безпечних процесів доставки програмного забезпечення. Нижче наведено цільові рекомендації для майбутніх досліджень, засновані на виявлених прогалинах:

Однією з нових сфер інтересу є інтеграція технологій штучного інтелекту та машинного навчання в автоматизацію DevOps. Майбутні дослідження можуть зосередитися на тому, як ці технології можуть покращити прийняття рішень, оптимізувати розподіл ресурсів та покращити прогнозну аналітику в середовищах DevOps [30].

Автоматизація на основі штучного інтелекту: дослідження потенціалу штучного інтелекту в автоматизації складних завдань DevOps, таких як виявлення аномалій, налаштування продуктивності та автоматизоване виправлення.

Дослідження можуть включати використання алгоритмів штучного інтелекту для прогнозування системних збоїв, оптимізації конвеєрів CI/CD та забезпечення інтелектуальної автоматизації, яка адаптується до змін робочих навантажень.

Машинне навчання для постійного вдосконалення: Дослідіть, як моделі машинного навчання можна інтегрувати в процеси DevOps, щоб забезпечити постійне навчання на основі минулих розгортань, збоїв та показників продуктивності. Це може включати розробку моделей машинного навчання, які аналізують історичні дані для рекомендації оптимізації для майбутніх релізів.

Хоча автоматизація широко впроваджується в невеликих, гнучких середовищах, її масштабування у великих, складних організаціях створює унікальні труднощі. Майбутні дослідження можуть розглянути стратегії впровадження автоматизації в великих масштабах, враховуючи такі фактори, як складність інфраструктури, розмір команди та організаційна структура.

Автоматизація у великих масштабах: Дослідіть найкращі практики масштабування автоматизації у великих, розподілених командах та складних IT-середовищах. Це може включати вивчення проблем управління кількома конвеєрами CI/CD, забезпечення узгодженості в різних середовищах та оптимізацію використання ресурсів.

Гібридна та багатохмарна автоматизація: Оскільки організації впроваджують гібридні та багатохмарні стратегії, необхідні дослідження для вивчення автоматизації розгортання на різних хмарних платформах. Дослідження можуть зосередитися на проблемах автоматизації робочих процесів у багатохмарних середовищах, зокрема щодо сумісності, безпеки та управління витратами.

Швидка еволюція інструментів та фреймворків автоматизації відкриває можливості для дослідження їхньої ефективності, адаптивності та майбутніх тенденцій.

Порівняльний аналіз інструментів автоматизації: Проведіть поглиблені порівняльні дослідження популярних інструментів автоматизації (наприклад, Jenkins, GitLab, Ansible, Kubernetes), щоб оцінити їхні сильні та слабкі сторони, а також придатність для різних випадків використання. Це може допомогти організаціям приймати обґрунтовані рішення під час вибору інструментів для своїх

DevOps-конвеєрів.

Новітні технології автоматизації: Дослідіть потенціал нових технологій, таких як безсерверні обчислення, контейнери та мікросервіси, для покращення автоматизації DevOps. Дослідження можуть зосередитися на використанні цих технологій для покращення масштабованості, зниження витрат та збільшення швидкості розгортання.

Майбутні дослідження можуть охопити ці напрямки дослідження та зробити свій внесок у постійний розвиток практик DevOps, гарантуючи, що автоматизація підвищує ефективність та відповідає вимогам безпеки, масштабованості та соціально-технічним аспектам. Ці рекомендації спрямовані на підтримку організацій, практиків та дослідників у подолання викликів та можливостей, що виникають у швидкозмінному ландшафті автоматизації DevOps.

У цьому дослідженні всебічно проаналізовано вплив автоматизації на практики DevOps, зосереджуючись на її ролі у підвищенні ефективності, масштабованості та співпраці в команді. Результати дослідження показують, що автоматизація значно оптимізує процеси DevOps, оптимізуючи повторювані завдання, збільшуючи швидкість розгортання та забезпечуючи безперервну інтеграцію та доставку. Інструменти автоматизації, такі як Jenkins, Ansible та Terraform, показали, що вони зменшують кількість помилок, що виникають вручну, покращують узгодженість між середовищами та підвищують якість випусків програмного забезпечення.

У дослідженні було виділено кілька найкращих практик впровадження автоматизації, включаючи прийняття поступової стратегії автоматизації , інтеграцію автоматизованого тестування, використання інфраструктури як коду (IaC) та сприяння культурі співпраці між міжфункціональними командами. Ці стратегії можуть допомогти організаціям подолати такі труднощі, як проблеми інтеграції інструментів, опір змінам та проблеми безпеки, тим самим максимізуючи переваги автоматизації.

Більше того, дослідження визначило ключові проблеми, пов'язані з впровадженням автоматизації, включаючи складність інтеграції інструментів, опір команди, проблеми масштабованості та ризики безпеки. Вирішення цих проблем

вимагає стратегічного підходу, що охоплює сумісні інструменти, інвестування в навчання та сприяння культурі безперервного навчання та інновацій.

Це дослідження доповнює існуючі знання про автоматизацію DevOps, підтверджуючи її технічні переваги та висвітлюючи її соціально-технічні наслідки, такі як вплив на динаміку команди та організаційну культуру. Дослідження підкреслює важливість цілісного підходу до автоматизації, який інтегрує технічні та культурні аспекти для досягнення стійких трансформацій DevOps. Надаючи практичні рекомендації та визначаючи напрямки для майбутніх досліджень, це дослідження є цінним ресурсом для практиків, дослідників та організацій, які прагнуть оптимізувати свої практики DevOps за допомогою автоматизації.

На завершення, автоматизація — це не просто технічне вдосконалення DevOps, а стратегічний фактор, що сприяє ефективності, інноваціям та співпраці. Організації, які цілісно застосовують автоматизацію, мають кращі можливості для адаптації до швидкозмінного цифрового середовища, досягаючи більшої гнучкості, стійкості та конкурентної переваги.

### 3.4 Висновки по розділу

У третьому розділі було розглянуто практичну реалізацію та оцінку ефективності DevOps-практик у процесі автоматизації життєвого циклу програмного забезпечення. Проаналізовано етапи життєвого циклу DevOps та механізми їх автоматизації за допомогою сучасних інструментів і технологій. Досліджено основні виклики та проблеми, що виникають під час впровадження DevOps, зокрема питання інтеграції, безпеки, масштабованості та організації командної взаємодії. Також проведено аналіз результатів впровадження DevOps-практик, що підтвердив підвищення швидкості розробки, стабільності роботи системи та якості програмного забезпечення. Отримані результати демонструють ефективність використання DevOps для оптимізації процесів розробки та підтримки сучасних програмних продуктів.

# ВИСНОВКИ

У ході виконання бакалаврської роботи було досліджено використання DevOps-практик для автоматизації життєвого циклу програмного забезпечення як комплексного підходу, спрямованого на підвищення ефективності розробки, тестування, розгортання та супроводу програмних систем. У роботі проаналізовано сучасні підходи до DevOps, досліджено переваги та недоліки основних інструментів автоматизації, а також розглянуто практичну реалізацію процесів безперервної інтеграції та доставки. У результаті було сформовано цілісне уявлення про організацію повністю автоматизованого життєвого циклу програмного забезпечення.

Ми розпочали з аналізу DevOps-методології, визначивши її основні принципи, зокрема співпрацю між командами розробки та супроводу, автоматизацію процесів та безперервність поставки програмного продукту. Було встановлено, що впровадження DevOps дозволяє значно скоротити час розробки та підвищити якість програмного забезпечення за рахунок інтеграції всіх етапів життєвого циклу в єдиний автоматизований процес.

Моделювання життєвого циклу DevOps дозволило структурувати ключові етапи - від написання коду до його тестування, збирання, розгортання та моніторингу. Було виявлено основні виклики, серед яких складність налаштування CI/CD-процесів, інтеграція різних інструментів та забезпечення стабільності автоматизованих пайплайнів. Розглянуті концепції Infrastructure as Code та контейнеризації дозволили підвищити керованість та повторюваність середовищ розгортання.

Практична частина охопила аналіз та застосування інструментів DevOps для автоматизації процесів розробки. Було забезпечено безперервну інтеграцію коду, автоматичне тестування та розгортання застосунків у тестове та продукційне середовище. Використання CI/CD-пайплайнів дозволило мінімізувати людський фактор, зменшити кількість помилок та прискорити випуск нових версій програмного забезпечення.

Особлива увага приділялася автоматизації тестування та моніторингу, що забезпечило своєчасне виявлення помилок і контроль стану системи в режимі реального часу. Це дало змогу підвищити стабільність роботи програмного забезпечення та швидкість реагування на інциденти.

Загалом, результати роботи підтверджують, що використання DevOps-практик є ефективним підходом до автоматизації життєвого циклу програмного забезпечення. Це дозволяє підвищити швидкість розробки, покращити якість продукту, забезпечити стабільність релізів та зменшити витрати на супровід систем. Водночас існують певні виклики, пов’язані зі складністю впровадження DevOps у великі системи, необхідністю налаштування інфраструктури та постійного вдосконалення процесів автоматизації.

У перспективі подальші дослідження можуть бути спрямовані на використання штучного інтелекту для оптимізації CI/CD-процесів, автоматичне виявлення помилок у пайплайнах, а також розширення можливостей моніторингу та аналітики для підвищення ефективності DevOps-середовищ.

## СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА


1. **Angular Documentation.** (2025). *angular.io*. https://angular.io/docs
2. **Azure Application Insights Overview.** (2025). *learn.microsoft.com*. https://learn.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview
3. **Azure Cosmos DB Documentation.** (2025). *learn.microsoft.com*. https://learn.microsoft.com/en-us/azure/cosmos-db
4. **Azure DevOps.** (2025). CI/CD Pipeline. https://learn.microsoft.com/en-us/azure/devops/pipelines/
5. **Bhatt, P., & Patel, S.** (2024). IoT integration for green energy management systems. *Journal of Renewable Energy Technology*, 12(3), 245–260. https://doi.org/10.1016/j.jret.2024.03.007
6. **Bootstrap.** (2025). Documentation. *getbootstrap.com*. https://getbootstrap.com/docs/
7. **Clean Code Principles in .NET.** (2023). *devblogs.microsoft.com*. https://devblogs.microsoft.com/dotnet/clean-code-principles/
8. **Cypress End-to-End Testing.** (2025). *docs.cypress.io*. https://docs.cypress.io/guides/overview/why-cypress
9. **Deloitte.** (2025). Digital solutions for renewable energy operations. *deloitte.com*. https://www2.deloitte.com/global/en/pages/energy-and-resources/articles/digital-solutions-renewable-energy.html
10. **Fowler, M.** (2002). *Patterns of Enterprise Application Architecture* (pp. 5–10). https://www.martinfowler.com/eaaCatalog/
11. **Freeman, E., & Robson, E.** (2021). *Head First Design Patterns*. *oreilly.com*. https://www.oreilly.com/library/view/head-first-design/9781492077992/
12. **Gamma, E., Helm, R., Johnson, R., & Vlissides, J.** (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. *pearson.com*. https://www.pearson.com/store/p/design-patterns-elements-of-reusable-object-oriented/P100000252058/
13. **Google Maps Platform Documentation.** (2025). developers.google.

*com*. https://developers.google.com/maps/documentation

14. **Green Software Foundation.** (2024). Software Carbon Intensity Specification. *greensoftware.foundation*. https://greensoftware.foundation/projects/software-carbon-intensity-specification

15. **Hassan, M., & Khan, A.** (2023). Scalable web architectures for renewable energy monitoring. *IEEE Transactions on Sustainable Computing*, 8(2), 134–145. https://doi.org/10.1109/TSUSC.2023.3245678

16. **ImpactQA.** (2024). The crucial role of software testing in the renewable energy sector. *impactqa.com*. https://www.impactqa.com/the-crucial-role-of-software-testing-in-the-renewable-energy-sector/

17. **Martin, R. C.** (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. *pearson.com*. https://www.pearson.com/store/p/clean-architecture-a-craftmans-guide-to-software-structure-and-design/P100001465201

18. **Microsoft Azure Documentation.** (2025). *learn.microsoft.com*. https://learn.microsoft.com/en-us/azure/

19. **MongoDB.** (2025). Documentation. https://www.mongodb.com/docs/

20. **.NET Core Best Practices.** (2023). *learn.microsoft.com*. https://learn.microsoft.com/en-us/dotnet/core/extensions/best-practices

21. **Osherove, R.** (2019). *The Art of Unit Testing*. *manning.com*. https://www.manning.com/books/the-art-of-unit-testing-third-edition

22. **OWASP Top Ten Security Risks.** (2024). *owasp.org*. https://owasp.org/www-project-top-ten/

23. **Postman API Testing Guide.** (2025). *learning.postman.com*. https://learning.postman.com/docs/

24. **Renewable Energy Data Management with Cloud Solutions.** (2024). *dataversity.net*. https://www.dataversity.net/renewable-energy-data-management-with-cloud-solutions/

25. **Resnick, M.** (2023). Continuous deployment with Azure. *learn.microsoft.com*. https://learn.microsoft.com/en-us/azure/architecture/example-scenario/devops/continuous-deployment

26. **SASS Basics.** (2024). *sass-lang.com*. https://sass-lang.com/guide

27. **Security Best Practices for Azure Cosmos DB.** (2024). *learn.microsoft.com*. https://learn.microsoft.com/en-us/azure/cosmos-db/secure-access-data

28. **Selenium Documentation.** (2025). *selenium.dev*. https://www.selenium.dev/documentation/

29. **Siemens.** (2024). Software for renewable energy asset management. *siemens.com*. https://www.siemens.com/global/en/products/energy/renewable-energy/software.html

30. **Wang, L., & Zhang, Y.** (2025). Real-time analytics for green energy web applications. *ACM Transactions on Web*, 19(1), 1–20. https://doi.org/10.1145/3647890

31. **Chen, J., & Li, X.** (2024). Web-based monitoring systems for solar energy facilities. *Renewable Energy*, 218, 119345. https://doi.org/10.1016/j.renene.2024.119345

32. **Gartner.** (2024). Trends in software for renewable energy management. *gartner.com*. https://www.gartner.com/en/information-technology/insights/ renewable -energy-software-trends

33. **Kumar, S., & Sharma, R.** (2023). Microservices for green energy web applications: Design and deployment. *Journal of Software Engineering Research and Development*, 11(2), 67–82. https://doi.org/10.1186/s40411-023-00189-9

34. **Schneider Electric.** (2024). EcoStruxure for renewable energy monitoring. *se.com*. https://www.se.com/ww/en/solutions/renewable-energy-monitoring

35. **Taylor, M., & Brown, A.** (2025). Security frameworks for green energy web platforms. *IEEE Security & Privacy*, 23(1), 45–53. https://doi.org/10.1109/MSEC.2024.347890

**БІБЛІОГРАФІЧНА ДОВІДКА**

**Тема бакалаврської роботи:** " Використання DevOps-практик для автоматизації життєвого циклу програмного забезпечення"

Обсяг пояснювальної записки: 69 аркушів

Дата закінчення бакалаврської роботи 10 червня 2026р.

Підпис студента _____