

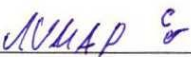
**МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ**

**ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І
СИСТЕМ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему: Комплексний DevSecOps-конвеєр на базі Kubernetes для полімовних репозиторіїв із централізованим керуванням вразливостями в DefectDojo

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи КБ-22-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за галузями)»


(підпис)

Лимара Єгора
Юрійовича
(прізвище, ім'я, по батькові)

Науковий керівник


(підпис)

Ратушняк Тетяна
Володимирівна, канд. фіз.-мат.
наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Керівник освітньо-
професійної програми


(підпис)

Ніжегородцев Владислав
Олександрович, канд. пед.
наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних технологій і систем протокол від «12» травня 2026 №20

Завідувач кафедри


(підпис)

Філоненко Михайло
Миколайович, канд. фіз.-мат.
наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Ірпінь – 2026

Реєстрація № 52

28.05.2026

(дата)



(підпис відповідальної особи)

Результати перевірки

до захисту
(до захисту, на доопрацювання)

12.05.26

(дата)



(підпис)

Результат захисту

5 (відмінно)

(оцінка)

09.06.2026

(дата)



(підпис)

Голова комісії:



(підпис)

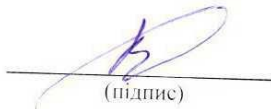
О.С. Дюбанов

Члени комісії:



(підпис)

М.М. Філоненко



(підпис)

В.О. Ніжегородцев



(підпис)

А.А. Омельчук

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	28		09.06.2026 Ратушняк Т.В.  Дюбанов О.С. 
2. Дотримання вимог щодо оформлення	від 0 до 10	9		Філоненко М.М.  Ніжегородцев В.О. 
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		Омельчук А.А.  Суховерхівка Л.П. 
4. Рецензент	від 0 до 5	4		
РАЗОМ БАЛІВ	від 0 до 50	45		

Оцінювання кваліфікаційної роботи науковим керівником – 46 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
91	A	5 (відмінно)	09.06.2026	Дюбанов О.С.  Філоненко М.М.  Ніжегородцев В.О.  Омельчук А.А.  Суховерхівка Л.П. 

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	
50-59	E	3 (задовільно)
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: «Інформаційні управляючі системи і технології
(за галузями)»

Затверджую
Завідувач кафедри  Філоненко М.М.
«01» грудня 2025

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Лимару Єгору Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи Комплексний DevSecOps-конвеєр на базі Kubernetes для полімовних репозиторіїв із централізованим керуванням вразливостями в DefectDojo

Затверджена наказом ректора від «28» листопада 2025 № 1526

2. Строк здачі закінченої роботи «08» травня 2026

3. Мета роботи: проектування та реалізація комплексного DevSecOps-конвеєра на базі Kubernetes для автоматизованого виявлення вразливостей у полімовних програмних репозиторіях із централізованим керуванням результатами сканування засобами платформи DefectDojo.

Об'єкт дослідження: процеси забезпечення безпеки та автоматизації розгортання програмного забезпечення у хмароорієнтованих середовищах для багатокомпонентних та полімовних програмних систем.

Предмет дослідження: методи, архітектурні рішення, інструменти автоматизації (DevSecOps-конвеєр, Kubernetes) та алгоритми централізованого менеджменту безпеки (DefectDojo) для виявлення і усунення вразливостей на всіх етапах життєвого циклу розробки полімовного програмного забезпечення.

Орієнтовна база дослідження: стандарты безпеки розробки програмного забезпечення, наукові публікації, аналітичні огляди.

4. Вихідні дані до роботи: хмарна інфраструктура на базі VPS-сервера з розгорнутим кластером k3s, платформами GitLab CI/CD, Flux CD та DefectDojo.

5. Перелік графічного матеріалу: концепція Shift-Left Security в DevSecOps-конвеєрі, архітектура DevSecOps-конвеєра на базі Kubernetes, потік даних у DevSecOps-конвеєрі від коміту до DefectDojo.

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом):

Розділ 1. Теоретико-методологічні засади DevSecOps

1.1. DevSecOps як концепція: принципи, місце безпеки в CI/CD

1.2. Огляд інструментів безпекового аналізу (SAST, SCA, Secret Detection)

1.3. Порівняльний аналіз платформ централізованого управління вразливостями

1.4. Кількісний вимір Shift-Left Security та галузеві стандарти

Розділ 2. Архітектура DevSecOps-конвеєра на базі Kubernetes

2.1. Архітектура запропонованого DevSecOps-конвеєра

2.2. Інфраструктура на базі k3s та GitOps (Flux CD)

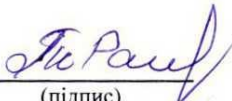
- 2.3. Проектування системи централізованого керування вразливостями в DefectDojo
 2.4. Контейнеризація сканерів та безпека образів
 Розділ 3. Реалізація та тестування DevSecOps -конвеєра
 3.1. Реалізація безпекових pipeline для полімовних репозиторіїв
 3.2. Інтеграція сканерів з DefectDojo та автоматизація імпорту результатів
 3.3. Тестування конвеєра та аналіз результатів сканування
 3.4 Перспективи розвитку системи та хмарна інтеграція

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 30.10.2025	До 30.10.2025
2	І розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	26.12.2025	26.12.2025
3	ІІ розділ (аналіз експериментальної частини про методики досліджень)	18.02.2026	18.02.2026
4	ІІІ розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	16.04.2026	16.04.2026
5	Висновки	01.05.2026	01.05.2026
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	07.05.2026	07.05.2026
7	Отримання відгуку керівника	29.05.2026	29.05.2026
8	Отримання зовнішньої рецензії	01.06.2026	01.06.2026

8. Дата видачі завдання «01» грудня 2025

9. Керівник кваліфікаційної роботи


(підпис)

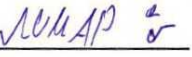
Ратушняк Т.В.
(прізвище, ініціали)

10. Керівник освітньо-професійної програми


(підпис)

Ніжегородцев В.О.
(прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти


(підпис)

Лимар Є.Ю.
(прізвище, ініціали)

АНОТАЦІЯ

Комплексний DevSecOps-конвеєр на базі Kubernetes для полімовних репозиторіїв із централізованим керуванням вразливостями в DefectDojo

Мета роботи полягає у проєктуванні та реалізації комплексного DevSecOps-конвеєра на базі Kubernetes для автоматизованого виявлення вразливостей у полімовних програмних репозиторіях із централізованим керуванням результатами сканування засобами платформи DefectDojo.

Завданнями роботи є: дослідження принципів DevSecOps та сучасного інструментарію безпекового аналізу; проєктування інфраструктури на базі k3s та Flux CD; реалізація CI/CD-конвеєрів із інтеграцією сканерів Semgrep, Gosec, Bandit, osv-scanner, Gitleaks та SCA-інструментів для мов Python, Go, JavaScript і PHP; налаштування автоматичної передачі результатів до DefectDojo.

Предметом дослідження є методи та інструменти автоматизованого аналізу безпеки програмного коду в CI/CD-конвеєрах.

Об'єктом дослідження є процес забезпечення безпеки програмного забезпечення в рамках DevSecOps-підходу на базі Kubernetes.

Базою дослідження виступає хмарна інфраструктура на базі VPS-сервера з розгорнутим кластером k3s, платформами GitLab CI/CD, Flux CD та DefectDojo.

За результатами дослідження реалізовано повнофункціональний DevSecOps-конвеєр, що підтримує статичний аналіз коду, аналіз залежностей та виявлення секретів для чотирьох мов програмування із централізованою агрегацією знахідок у DefectDojo.

Кваліфікаційна робота містить 45 сторінок, 12 таблиць, 3 рисунки, список використаних джерел з 43 найменувань.

Ключові слова: DevSecOps, Kubernetes, k3s, CI/CD, SAST, SCA, DefectDojo, GitLab, Flux CD, безпека програмного забезпечення, аналіз вразливостей.

Рік виконання кваліфікаційної роботи: 2025–2026.

Рік захисту роботи: 2026.

ANNOTATION

Comprehensive DevSecOps Pipeline Based on Kubernetes for Polyglot Repositories with Centralized Vulnerability Management in DefectDojo

The purpose of this work is to design and implement a comprehensive DevSecOps pipeline based on Kubernetes for automated vulnerability detection in polyglot software repositories with centralized management of scanning results using the DefectDojo platform.

The tasks of the thesis are: investigating the principles of DevSecOps and modern security analysis tools; designing an infrastructure based on k3s and Flux CD; implementing CI/CD pipelines with the integration of Semgrep, Gosec, andit, osv-scanner, Gitleaks, and SCA tools for Python, Go, JavaScript, and PHP; configuring the automated transfer of scan results to DefectDojo.

The subject of research is methods and tools for automated security analysis of source code in CI/CD pipelines.

The object of research is the software security assurance process within the DevSecOps approach based on Kubernetes.

The results of the research include a fully functional DevSecOps pipeline supporting static code analysis, dependency analysis, and secret detection for four programming languages with centralized aggregation of findings in DefectDojo.

The qualification work comprises 45 pages, 12 tables, 3 illustrations, and a bibliography with 43 references.

Keywords: DevSecOps, Kubernetes, k3s, CI/CD, SAST, SCA, DefectDojo, GitLab, Flux CD, software security, vulnerability analysis.

Year of implementation of the qualification thesis: 2025–2026.

Year of defense: 2026.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ.....	3
ВСТУП	4
РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ЗАСАДИ DEVSECOPS.....	7
1.1. DevSecOps як концепція: принципи, місце безпеки в CI/CD	7
1.2. Огляд інструментів безпекового аналізу (SAST, SCA, Secret Detection).....	9
1.3. Порівняльний аналіз платформ централізованого управління вразливостями	12
1.4. Кількісний вимір Shift-Left Security та галузеві стандарти	14
РОЗДІЛ 2. АРХІТЕКТУРА DEVSECOPS-КОНВЕЄРА НА БАЗІ KUBERNETES	18
2.1. Архітектура запропонованого DevSecOps-конвеєра.....	18
2.2. Інфраструктура на базі k3s та GitOps (Flux CD).....	20
2.3. Проектування системи централізованого керування вразливостями в DefectDojo	23
2.4. Контейнеризація сканерів та безпека образів	25
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ DEVSECOPS-КОНВЕЄРА	29
3.1. Реалізація безпекових pipeline для полімовних репозиторіїв	29
3.2. Інтеграція сканерів з DefectDojo та автоматизація імпорту результатів.....	31
3.3. Тестування конвеєра та аналіз результатів сканування.....	34
3.4 Перспективи розвитку системи та хмарна інтеграція.....	36
ВИСНОВКИ.....	40
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	42

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

CI/CD	– Continuous Integration / Continuous Delivery (неперервна інтеграція / неперервне постачання)
CVE	– Common Vulnerabilities and Exposures (загальний перелік вразливостей і вразливих місць)
DAST	– Dynamic Application Security Testing (динамічне тестування безпеки застосунків)
DefectDojo	– платформа централізованого керування вразливостями
DevOps	– Development + Operations (розробка + операційна діяльність)
DevSecOps	– Development + Security + Operations (розробка + безпека + операційна діяльність)
Flux CD	– GitOps-оператор для Kubernetes
GitLab CI	– система неперервної інтеграції платформи GitLab
GitOps	– методологія управління інфраструктурою через Git-репозиторії
k3s	– легковагова дистрибуція Kubernetes від Rancher Labs
MR	– Merge Request (запит на злиття гілок)
OWASP	– Open Web Application Security Project
SAST	– Static Application Security Testing (статичне тестування безпеки застосунків)
SCA	– Software Composition Analysis (аналіз складу програмного забезпечення)
VPS	– Virtual Private Server (віртуальний приватний сервер)
YAML	– Yet Another Markup Language (мова розмітки конфігурацій)

ВСТУП

Актуальність теми. Сучасна розробка програмного забезпечення характеризується стрімким скороченням циклів випуску продуктів та масовим переходом до мікросервісної архітектури й хмарних середовищ. В умовах, коли команди щодня здійснюють десятки або сотні комітів і розгортань, традиційний підхід до безпеки — коли аналіз вразливостей відбувається наприкінці циклу розробки або лише перед випуском — стає критично неефективним. Усунення вразливостей на пізніх стадіях обходиться в десятки разів дорожче, ніж їх виявлення на етапі написання коду [1].

Концепція DevSecOps, яка інтегрує практики безпеки безпосередньо в CI/CD-конвеєр, є відповіддю на цей виклик. Вона дозволяє перетворити безпеку з бар'єру на невід'ємну складову процесу розробки — принцип «Security as Code». Проте на практиці реалізація DevSecOps у реальних проєктах стикається з рядом проблем: неоднорідністю технологічного стека (використання різних мов програмування в одному проєкті), відсутністю єдиного центру агрегації та управління знахідками від різних сканерів, а також складністю налаштування та підтримки безпекової інфраструктури [2].

Зазначені проблеми обумовлюють актуальність дослідження та розробки комплексного DevSecOps-конвеєра, що підтримує полімовні репозиторії та забезпечує централізоване керування вразливостями. Особливої значущості набуває використання сучасних інструментів оркестрації — Kubernetes — як платформи для розгортання безпекової інфраструктури, а також застосування GitOps-підходу для управління її конфігурацією.

Стан і ступінь розробки проблеми в спеціальній літературі.

Питання інтеграції безпеки в процеси DevOps активно досліджуються як у вітчизняній, так і в зарубіжній науковій спільноті. Зокрема, дослідження Shahin et al. [3] систематизують підходи до continuous delivery та безпеки, а роботи Kim et al. [4] описують принципи DevOps-трансформації на рівні організацій. Специфіку інструментарію SAST і SCA висвітлено у технічних звітах

OWASP [5] та NIST [6]. Платформа DefectDojo як інструмент управління вразливостями описана в офіційній документації [7] та ряді практичних досліджень [8]. Водночас питання побудови єдиного конвеєра для полімовних репозиторіїв із централізованим керуванням знахідками на базі Kubernetes залишаються недостатньо розробленими в науковій літературі.

Метою кваліфікаційної роботи є проєктування та реалізація комплексного DevSecOps-конвеєра на базі Kubernetes для автоматизованого виявлення вразливостей у полімовних програмних репозиторіях із централізованим керуванням результатами сканування засобами платформи DefectDojo.

Для досягнення поставленої мети визначено такі завдання:

- 1) дослідити принципи DevSecOps та сучасний інструментарій безпекового аналізу коду;
- 2) провести порівняльний аналіз платформ централізованого управління вразливостями та обґрунтувати вибір DefectDojo;
- 3) спроектувати архітектуру DevSecOps-конвеєра на базі Kubernetes (k3s) з GitOps-управлінням через Flux CD;
- 4) реалізувати безпекові CI/CD-pipeline для полімовних репозиторіїв із інтеграцією сканерів SAST, SCA та Secret Detection;
- 5) налаштувати автоматичну передачу результатів сканування до DefectDojo та перевірити ефективність розробленого конвеєра.

Об'єкт дослідження — процес забезпечення безпеки програмного забезпечення в рамках DevSecOps-підходу на базі Kubernetes.

Предмет дослідження — методи та інструменти автоматизованого аналізу безпеки програмного коду (SAST, SCA, Secret Detection) у CI/CD-конвеєрах та їх інтеграція з платформою централізованого управління вразливостями DefectDojo.

Методи дослідження. У роботі застосовано такі методи дослідження: системний аналіз — для дослідження архітектурних підходів до побудови DevSecOps-конвеєрів; порівняльний аналіз — для оцінки інструментів безпекового сканування та платформ управління вразливостями; метод

проектування інформаційних систем — для розробки архітектури конвеєра; експериментальний метод — для тестування та оцінки ефективності реалізованого рішення.

База дослідження. Базою дослідження виступає хмарна інфраструктура на базі VPS-сервера Hetzner Cloud (конфігурація CPX22: 2 vCPU, 4 GB RAM, 80 GB SSD, регіон eu-central) з розгорнутим кластером k3s версії v1.34.6+k3s1, платформами GitLab CI/CD (SaaS), Flux CD та DefectDojo, доступним за доменом defectdojo.pinkmonkey123.shop. Конфігурація всіх компонентів зберігається в чотирьох репозиторіях GitLab: devsecops-infra, devsecops, security_scripts та security_pipeline.

Наукова та практична значимість одержаних результатів. Практична значимість роботи полягає у створенні готового до використання DevSecOps-конвеєра, який дозволяє автоматизувати процес виявлення вразливостей у програмних проєктах з полімовним кодом. Розроблені шаблони CI/CD-pipeline можуть бути використані як у навчальних, так і в комерційних проєктах. Реалізоване рішення демонструє можливість побудови повнофункціональної безпекової інфраструктури на базі open-source інструментів із мінімальними витратами (менше 10 € на місяць), що є особливо цінним для малих та середніх команд розробки.

РОЗДІЛ 1

ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ЗАСАДИ DEVSECOPS

1.1. DevSecOps як концепція: принципи та місце безпеки в CI/CD

Поняття DevSecOps виникло як еволюційний розвиток методології DevOps з метою усунення її ключового недоліку — відсутності системної уваги до питань інформаційної безпеки в процесі розробки та постачання програмного забезпечення. Якщо DevOps зосереджується на злитті практик розробки (Development) та операційної діяльності (Operations) для скорочення циклу випуску продукту, то DevSecOps інтегрує до цього циклу третій вимір — безпеку (Security) [1].

Традиційна модель розробки програмного забезпечення передбачала перевірку безпеки на фінальних стадіях — перед випуском або вже після розгортання в продуктивному середовищі. Такий підхід призводив до значних витрат на усунення виявлених вразливостей: дослідження IBM показують, що вартість виправлення дефекту безпеки, виявленого на стадії виробництва, у 15–30 разів перевищує вартість усунення тієї ж проблеми на стадії проєктування [2]. В умовах сучасних CI/CD-конвеєрів, де команди виконують десятки комітів на день, відкладання безпекового аналізу стає критично неприйнятним.

DevSecOps реалізує принцип *Shift-Left Security* — «зрушення» заходів безпеки вліво по осі часу життєвого циклу розробки, тобто якомога ближче до моменту написання коду. Це означає, що перевірки безпеки відбуваються не після завершення розробки, а безпосередньо в процесі — на кожному коміті, кожному запиті на злиття (Merge Request) та кожній збірці [3].

Основні принципи DevSecOps включають:

- Security as Code: конфігурація безпекових інструментів, правила сканування та порогові значення зберігаються у вигляді коду разом із вихідним кодом застосунку та підпадають під процеси контролю версій і рецензування;

- автоматизація: безпекові перевірки мають бути повністю автоматизовані і не вимагати ручного втручання для кожного циклу розробки;
- безперервний моніторинг: аналіз безпеки є безперервним процесом, а не одноразовою перевіркою;
- спільна відповідальність: безпека є обов'язком усієї команди — розробників, інженерів з експлуатації та фахівців із безпеки, а не лише окремого підрозділу [1].

Розглянемо місце заходів безпеки у типовому CI/CD-конвеєрі. Стандартний DevSecOps-конвеєр охоплює такі етапи: планування (моделювання загроз, аналіз вимог безпеки), написання коду (інтеграція в IDE, pre-commit hooks), збірка (SAST, перевірка залежностей), тестування (DAST, динамічний аналіз), упакування (сканування контейнерних образів), розгортання (перевірка конфігурацій IaC) та моніторинг у продуктивному середовищі (виявлення аномалій, реагування на інциденти) [4].

Ключовим викликом при побудові DevSecOps-конвеєрів у реальних проєктах є підтримка полімовного середовища. Сучасні продукти все частіше поєднують різні мови програмування: backend на Go або Python, фронтенд на JavaScript/TypeScript, скрипти автоматизації на Bash або PHP. Кожна мова вимагає специфічного інструментарію для аналізу — не існує єдиного сканера, що однаково ефективно покриває всі мови. Це обумовлює необхідність побудови конвеєра з кількох спеціалізованих інструментів, результати яких потребують централізованої агрегації [2].

Таблиця 1.1

Порівняння підходів до забезпечення безпеки в розробці ПЗ

Критерій	Традиційний підхід	DevOps	DevSecOps
Час перевірки безпеки	Після розробки	Переважно після розробки	На кожному кроці CI/CD
Відповідальність за безпеку	Окрема команда безпеки	Переважно команда безпеки	Вся команда розробки

Автоматизація перевірок	Мінімальна	Часткова	Повна (Security as Code)
Час виявлення вразливості	Тижні–місяці	Дні–тижні	Хвилини–години
Вартість усунення	Висока	Середня	Низька (раннє виявлення)

Джерело: складено автором на основі [1, 2, 4]

Таким чином, DevSecOps є не просто набором інструментів, а комплексною методологією, що вимагає організаційних змін поряд із технічними. Проте в межах даної кваліфікаційної роботи основна увага зосереджена на технічній складовій — побудові інструментального конвеєра, що реалізує принципи DevSecOps для полімовних репозиторіїв.

1.2. Огляд інструментів безпекового аналізу (SAST, SCA, Secret Detection)

Безпековий аналіз у DevSecOps-конвеєрі реалізується за допомогою кількох категорій інструментів, кожна з яких орієнтована на виявлення певного класу вразливостей. У межах даної роботи розглядаються три основні категорії: статичне тестування безпеки застосунків (SAST), аналіз складу програмного забезпечення (SCA) та виявлення секретних даних (Secret Detection) [5].

Static Application Security Testing (SAST) — аналізує вихідний код застосунку без його виконання. Інструменти SAST сканують код на наявність небезпечних патернів, потоків даних та відомих вразливих конструкцій, визначаючи точний файл та рядок коду, де виявлено проблему [5]. До переваг SAST відносять: раннє виявлення вразливостей ще до збірки та запуску застосунку; здатність знаходити складні логічні вразливості через аналіз потоків даних (taint analysis); незалежність від середовища виконання. Основним недоліком є відносно висока частка хибних спрацьовувань (false positives) — від 15 до 60% залежно від інструменту та налаштувань [6].

У контексті полімовного репозиторію вибір SAST-інструментів визначається підтримкою відповідних мов. У даній роботі використовуються:

- Semgrep — lightweight SAST-інструмент, що підтримує понад 40 мов програмування. Використовує декларативні правила у форматі YAML, дозволяючи гнучко налаштовувати перевірки під специфіку проєкту. Особливо ефективний для Python, Go, JavaScript, PHP [7];
- Gosec (Go Security Checker) — спеціалізований SAST-інструмент для мови Go. Аналізує AST (Abstract Syntax Tree) Go-програм та виявляє вразливості, характерні для цієї мови: небезпечне використання криптографії, SQL-ін'єкції, проблеми з конкурентним доступом [8];
- Bandit — SAST-інструмент для Python, розроблений спільнотою PyCQA. Перевіряє Python-код на відповідність базовим правилам безпеки: використання небезпечних функцій, hardcoded credentials, проблеми з обробкою виключень [9].

Software Composition Analysis (SCA) — аналізує залежності застосунку (бібліотеки, пакети, фреймворки) на наявність відомих вразливостей шляхом зіставлення з базами даних CVE. На відміну від SAST, SCA не аналізує власний код, а зосереджується на сторонніх компонентах. Актуальність SCA пояснюється тим, що сучасні застосунки містять від 70 до 90% стороннього коду у вигляді залежностей [10]. У 2024 році було опубліковано понад 54 000 нових CVE, що демонструє постійне зростання кількості відомих вразливостей у відкритому програмному забезпеченні [11].

У роботі використовується Google OSV-Scanner — інструмент аналізу залежностей, що підтримує екосистеми PyPI (Python), npm (JavaScript), Composer (PHP), Go modules та інші. OSV-Scanner працює з базою даних OSV (Open Source Vulnerabilities), яка агрегує дані з NVD, GitHub Advisory та інших джерел [12]. Для кожної мови також реалізовані спеціалізовані SCA-перевірки: `sca_python`, `sca_js`, `sca_php` — що дозволяє отримати більш детальну інформацію про залежності в кожній екосистемі.

Secret Detection — окрема категорія інструментів, що виявляє випадково зафіксовані в коді або в git-history секретні дані: паролі, API-ключі, токени доступу, приватні ключі тощо. Потрапляння таких даних у репозиторій є однією з найпоширеніших причин витоків. У роботі використовується Gitleaks — відкритий інструмент, що перевіряє весь git-history репозиторію, а також поточний стан коду на наявність секретних даних за допомогою регулярних виразів і ентропійного аналізу [13].

Таблиця 1.2

Порівняльна характеристика інструментів безпекового аналізу

Інструмент	Категорія	Мови / екосистеми	Що виявляє	Формат виводу
Semgrep	SAST	Python, Go, JS, PHP, 40+	Небезпечні патерни, ін'єкції, XSS	JSON, SARIF
Gosec	SAST	Go	Go-специфічні вразливості, криптографія	JSON, SARIF
Bandit	SAST	Python	Небезпечні функції, hardcoded secrets	JSON, CSV
OSV-Scanner	SCA	PyPI, npm, Go, Composer, +	CVE у залежностях	JSON
Gitleaks	Secret Detection	Будь-яка (git history)	API-ключі, паролі, токени	JSON, SARIF

Джерело: складено автором на основі [7, 8, 9, 12, 13]

Важливою характеристикою перерахованих інструментів є підтримка стандартизованих форматів виводу — насамперед JSON та SARIF (Static Analysis Results Interchange Format). Наявність машиночитаного формату є обов'язковою умовою для автоматичної передачі результатів до платформи централізованого управління вразливостями DefectDojo [14].

1.3. Порівняльний аналіз платформ централізованого управління вразливостями

Ключовою проблемою при використанні декількох безпекових сканерів є фрагментованість результатів: кожен інструмент надає звіт у власному форматі, і без централізованої системи відстеження команді доводиться вручну аналізувати результати з кожного сканера окремо. Платформи управління вразливостями (Vulnerability Management Platforms, VMP) вирішують цю проблему шляхом агрегації, дедублікації та централізованого відстеження знахідок з різних джерел [15].

Серед відкритих та умовно безкоштовних рішень у цій категорії виділяються три основних платформи: DefectDojo, OWASP Dependency-Track та Faraday. Розглянемо кожен з них.

DefectDojo — відкрита платформа оркестрації безпекового тестування та управління вразливостями, є флагманським проєктом OWASP. Платформу розпочато у 2013 році, відкритий код опублікований у 2015 році; станом на 2025 рік налічує понад 30 мільйонів завантажень [16]. DefectDojo забезпечує: інтеграцію з понад 200 інструментами безпекового сканування через стандартизований API; автоматичну дедублікацію знахідок від різних сканерів; відстеження ремедіації через вбудовані SLA; двостороннє інтегрування з Jira; гнучку ієрархічну структуру даних (Product → Engagement → Test → Finding) [17].

OWASP Dependency-Track — платформа, орієнтована виключно на управління залежностями та SBOM (Software Bill of Materials). Підтримує стандарти CycloneDX та SPDX для опису компонентного складу ПЗ. На відміну від DefectDojo, не є universal vulnerability management platform — основна сфера застосування обмежена аналізом залежностей і supply chain security [15].

Faraday — комерційна платформа (з обмеженою безкоштовною версією), орієнтована насамперед на задачі пентестингу та ручного тестування безпеки.

Має менш розвинену підтримку автоматизованого імпорту результатів CI/CD-сканерів порівняно з DefectDojo [15].

Таблиця 1.3

Порівняння платформ управління вразливостями

Критерій	DefectDojo	OWASP Dependency-Track	Faraday
Ліцензія	BSD 3-Clause (Open Source)	Apache 2.0 (Open Source)	Комерційна / обмежений free tier
Підтримувані сканери	200+ (SAST, SCA, DAST, пентест)	SCA / SBOM (CycloneDX, SPDX)	60+ (орієнтація на пентест)
Дедублікація знахідок	Так (cross-scanner)	Часткова (в межах компонентів)	Так
REST API для CI/CD	Так (v2 API)	Так	Так (обмежено у free)
Орієнтація	DevSecOps, ASPM	Supply chain, SCA	Пентест, ручне тестування
Kubernetes deployment	Так (Helm chart)	Так (Helm chart)	Так (Docker)

Джерело: складено автором на основі [15, 16, 17]

На підставі проведеного порівняльного аналізу для реалізації у даній кваліфікаційній роботі обрано платформу DefectDojo з таких міркувань: максимальна кількість підтримуваних форматів імпорту від SAST, SCA та Secret Detection інструментів; наявність повнофункціонального REST API v2 для автоматизованого імпорту з CI/CD; офіційна підтримка розгортання у Kubernetes через Helm chart; статус флагманського OWASP-проєкту з активною спільнотою та регулярними оновленнями; повністю відкрита ліцензія без обмежень функціональності у безкоштовній версії [16, 17].

1.4. Кількісний вимір Shift-Left Security та галузеві стандарти

Концепція Shift-Left Security набула широкого практичного визнання після публікації низки галузевих досліджень, що кількісно підтвердили її економічну ефективність. Згідно з даними IBM Systems Sciences Institute, вартість виправлення дефекту безпеки на стадії тестування у 15 разів вища, ніж на стадії проєктування, а на стадії підтримки — у 100 разів вища [2]. Ці цифри пояснюють, чому інтеграція безпекових перевірок у CI/CD стає стандартною практикою, а не опцією.



Рис. 1.1. Концепція Shift-Left Security в DevSecOps-конвеєрі

Дослідження Rajapakse et al. (2022) на базі систематичного огляду 56 первинних досліджень виявило ключові виклики при впровадженні DevSecOps у реальних організаціях [2]. Серед найчастіше згадуваних проблем — відсутність автоматизації безпекових перевірок (лише 12% організацій виконують сканування при кожному коміті), брак інтеграції між інструментами безпеки та CI/CD-системами, а також культурний опір: розробники сприймають безпекові перевірки як гальмо розробки. Саме ці виклики визначили архітектурні рішення у даній роботі: повна автоматизація, `allow_failure: true` та розподіл на тег-специфічні runner.

Нормативне підґрунтя DevSecOps формується на рівні державних стандартів. NIST SP 800-204D (2023) визначає стратегії інтеграції заходів безпеки у CI/CD-конвеєри хмарних застосунків, наголошуючи на необхідності верифікації кожного артефакту в ланцюжку постачання ПЗ [4]. Документ

вводить поняття Software Supply Chain (SSC) — сукупності всіх операцій від написання коду до розгортання — і вимагає впровадження контролів на кожному етапі цього ланцюжка. Реалізований у даній роботі конвеєр безпосередньо відповідає рекомендаціям NIST щодо автоматизованого сканування залежностей (SCA) та статичного аналізу (SAST) у CI/CD.

Окремої уваги заслуговує питання *ефективності мультиінструментального підходу*. Академічне дослідження на базі EASE 2024, що тестувало чотири SAST-інструменти проти 170 відомих вразливостей у Java-кодi, встановило: жоден з інструментів поодиночі не виявив більше 38.8% вразливостей [14]. Аналогічне дослідження Li et al. (ESEC/FSE 2023) показало, що окремий інструмент виявляє лише 12.7% реальних вразливостей. Ці дані підтверджують доцільність архітектурного рішення даної роботи — використання кількох спеціалізованих сканерів паралельно, що в сукупності забезпечують значно ширше покриття, ніж будь-який окремий інструмент.

Важливим аспектом побудови безпекового конвеєра є управління хибними спрацьовуваннями (false positives). За даними реальних впроваджень, рівень false positives для SAST-інструментів коливається від 15 до 60% [14]. Це означає, що значна частина знахідок потребує ручної верифікації. DefectDojo вирішує цю проблему через механізм risk acceptance та false positive marking: одного разу верифіковане хибне спрацьовування позначається у системі і не з'являється у звітах повторно при наступних скануваннях. Таким чином, з часом якість звітів покращується без додаткових зусиль команди.

Порівняно з SAST, рівень false positives для SCA-інструментів значно нижчий — 2–10% [14]. Це пояснюється тим, що SCA оперує точними ідентифікаторами версій пакетів та відомими CVE — тут немає суб'єктивного трактування патернів коду. Проте SCA має власне обмеження: без reachability analysis (аналізу досяжності) до 98% знахідок можуть бути технічно коректними, але практично нерелевантними — вразлива функція є у пакеті, але ніколи не викликається кодом застосунку. OSV-Scanner, використаний у даній роботі, не підтримує reachability analysis, що є відомим обмеженням поточної реалізації.

Таблиця 1.4

Рівні зрілості DevSecOps-практик за моделлю DSOMM

Рівень	Характеристика	Безпекові практики	Відповідність проєкту
1 — Початковий	Ручні перевірки, відсутність автоматизації	Ручний code review на безпеку	Ні
2 — Базовий	Окремі автоматизовані перевірки	SAST на окремих гілках	Частково
3 — Визначений	Інтегровані перевірки у CI/CD	SAST + SCA на кожному MR	Так ✓
4 — Керуваний	Централізоване управління вразливостями	Платформа VMP + SLA + метрики	Так ✓
5 — Оптимізований	Передбачувальна безпека, AI-аналіз	ML-детекція аномалій, DAST, RASP	Перспектива

Джерело: складено автором на основі DevSecOps Maturity Model (DSOMM) [3]

Як видно з таблиці 1.4, реалізований у даній роботі конвеєр відповідає рівням 3 та 4 моделі зрілості DSOMM: забезпечено інтегровані перевірки у CI/CD на кожному MR та централізоване управління вразливостями через DefectDojo з підтримкою SLA-трекінгу. Перехід до рівня 5 передбачає впровадження динамічного тестування (DAST) та ML-аналізу, що може бути розглянуто як напрям подальшого розвитку системи.

Висновки до розділу 1

У першому розділі проведено теоретико-методологічний аналіз засад DevSecOps та інструментарію безпекового аналізу. Встановлено, що DevSecOps є комплексною методологією, яка реалізує принцип Shift-Left Security шляхом інтеграції автоматизованих безпекових перевірок на всіх стадіях CI/CD-конвеєра. Визначено три ключові категорії інструментів безпекового аналізу:

SAST (Semgrep, Gosec, Bandit), SCA (OSV-Scanner та спеціалізовані інструменти) та Secret Detection (Gitleaks), кожна з яких орієнтована на виявлення специфічного класу вразливостей.

Проведено порівняльний аналіз платформ централізованого управління вразливостями, за результатами якого обґрунтовано вибір DefectDojo як оптимального рішення для реалізації централізованого керування знахідками від полімовного DevSecOps-конвеєра. У наступному розділі буде детально описано архітектуру запропонованого рішення та інфраструктуру на базі Kubernetes.

РОЗДІЛ 2

АРХІТЕКТУРА DEVSECOPS-КОНВЕЄРА НА БАЗІ KUBERNETES

2.1. Архітектура запропонованого DevSecOps-конвеєра

Проектування DevSecOps-конвеєра для полімовних репозиторіїв вимагає чіткого визначення архітектурних компонентів та потоків даних між ними. У даній роботі реалізовано архітектуру, що охоплює чотири логічні рівні: рівень вихідного коду та системи контролю версій, рівень CI/CD-автоматизації, рівень безпекового аналізу та рівень централізованого управління вразливостями [1].

Загальний потік даних у конвеєрі описується таким чином: розробник виконує коміт або створює Merge Request у GitLab → система GitLab CI автоматично запускає безпековий pipeline → кожен із сканерів аналізує репозиторій у своєму контейнеризованому середовищі → результати сканування автоматично передаються до DefectDojo через REST API → DefectDojo дедублікує знахідки та відображає їх у централізованому дашборді для команди безпеки.

Для забезпечення незалежності безпекового конвеєра від основного коду застосунку прийнято рішення про винесення всіх визначень pipeline в окремий репозиторій `security_pipelines`. Це дозволяє підключати єдиний централізований набір безпекових перевірок до будь-якого репозиторію організації через механізм `include GitLab CI`, без необхідності дублювання конфігурацій [18].

Система складається з чотирьох репозиторіїв GitLab із чітко розмежованими обов'язками:

- `devsecops` — основний репозиторій з реалізацією кожного сканера у вигляді Python-скрипта та `Dockerfile`. Містить pipeline для білдингу та публікації Docker-образів сканерів до GitLab Container Registry;
- `devsecops-infra` — репозиторій інфраструктурного коду у форматі Flux CD Kustomization та HelmRelease. Описує стан усіх компонентів Kubernetes-кластера: DefectDojo, GitLab Runner, неймспейси;

- security_pipelines — репозиторій із визначенням GitLab CI pipeline, що підключається до репозиторіїв розробників та запускає відповідні сканери залежно від вмісту репозиторію;
- security_scripts — репозиторій тестових застосунків для демонстрації роботи конвеєра та верифікації виявлення вразливостей.

Таблиця 2.1

Компоненти архітектури DevSecOps-конвеєра

Компонент	Технологія	Роль у системі	Розміщення
Система контролю версій	GitLab.com (SaaS)	Зберігання коду, тригер pipeline	GitLab Cloud
CI/CD-оркестратор	GitLab CI	Координація запуску сканерів	GitLab Cloud
Runner (виконавець)	GitLab Runner (Kubernetes executor)	Виконання CI-jobs у pods	k3s кластер (VPS)
Образи сканерів	Docker (GitLab Registry)	Ізольоване середовище кожного сканера	GitLab Container Registry
GitOps-оператор	Flux CD v2	Синхронізація стану кластера з Git	k3s кластер (VPS)
Управління вразливостями	DefectDojo v1.9.23	Агрегація, дедублікація, відстеження знахідок	k3s кластер (VPS)
База даних	PostgreSQL	Зберігання даних DefectDojo	k3s кластер (VPS)
Черга повідомлень	RabbitMQ + Valkey	Асинхронна обробка задач DefectDojo	k3s кластер (VPS)

Джерело: складено автором

Ключовою архітектурною особливістю є модульність: кожен сканер є незалежним Docker-образом, що зберігається у GitLab Container Registry під окремим тегом (наприклад, `registry.gitlab.com/qvolit/devsecops:semgrep-7`).

Завдяки цьому оновлення або заміна окремого сканера не впливає на роботу решти конвеєра. Спільний модуль `aux.ru`, що входить до кожного образу, відповідає за уніфіковану логіку взаємодії з API DefectDojo [18].

2.2. Інфраструктура на базі k3s та GitOps (Flux CD)

Вибір k3s як дистрибутиву Kubernetes для даного проєкту обумовлений рядом практичних міркувань. k3s — це сертифікована CNCF (Cloud Native Computing Foundation) дистрибутива Kubernetes від Rancher Labs, що оптимізована для роботи на ресурсно обмежених середовищах. На відміну від повного Kubernetes, k3s поставляється у вигляді єдиного бінарного файлу розміром близько 50 МБ та споживає значно менше оперативної пам'яті: мінімальні вимоги становлять 512 МБ RAM проти 2 ГБ для стандартного Kubernetes [19].

Для цілей даного проєкту (VPS-сервер Hetzner CPX22 з 4 ГБ RAM та 2 vCPU) використання повного Kubernetes або хмарного EKS було б надмірним: хмарний кластер AWS EKS обходиться у 150–400 доларів на місяць лише за інфраструктуру, тоді як поточне рішення коштує 9.49 євро на місяць. Встановлена версія k3s: `v1.34.6+k3s1`, що базується на ядрі Kubernetes 1.34.6 та включає компоненти Kustomize v5.7.1 [20].

GitOps — це методологія управління інфраструктурою, при якій бажаний стан усіх ресурсів кластера описується декларативно у Git-репозиторії і автоматично синхронізується з реальним станом кластера спеціальним оператором. Flux CD v2 — один із провідних GitOps-операторів для Kubernetes, що є проєктом під егідою CNCF [21].

У репозиторії `devsecops-infra` реалізовано таку структуру GitOps:

```
devsecops-infra/
├── clusters/
│   └── my-cluster/
│       └── flux-system/
```

```

|           |— gotk-components.yaml    # Flux CD системні
КОМПОНЕНТИ
|           |— gotk-sync.yaml          # Налаштування синхронізації
|           |— infrastructure.yaml     # Kustomization для
інфраструктури
|           |— kustomization.yaml      # Точка входу Flux
└─ main/
   └─ infrastructure/
      └─ defectdojo/
         |— defectdojo-helm-repo.yaml  # HelmRepository
         └─ defectdojo.yaml             # HelmRelease
      └─ gitlab-runner/
         |— repository.yaml             # HelmRepository GitLab
         └─ runner.yaml                 # HelmRelease Runner
      └─ namespaces/
         |— defectdojo.yaml             # Namespace defectdojo
         └─ gitlab-runner.yaml          # Namespace gitlab-runner

```

Flux CD стежить за репозиторієм devsecops-infra та кожну хвилину перевіряє відповідність стану кластера описаному у Git. Якщо хтось вручну змінить ресурс у кластері, Flux автоматично відновить його до стану, описаного в репозиторії (пруннінг). Цей підхід забезпечує повну відтворюваність інфраструктури: при виникненні проблем достатньо відновити VPS і запустити flux bootstrap — весь стек буде відновлено автоматично [21].

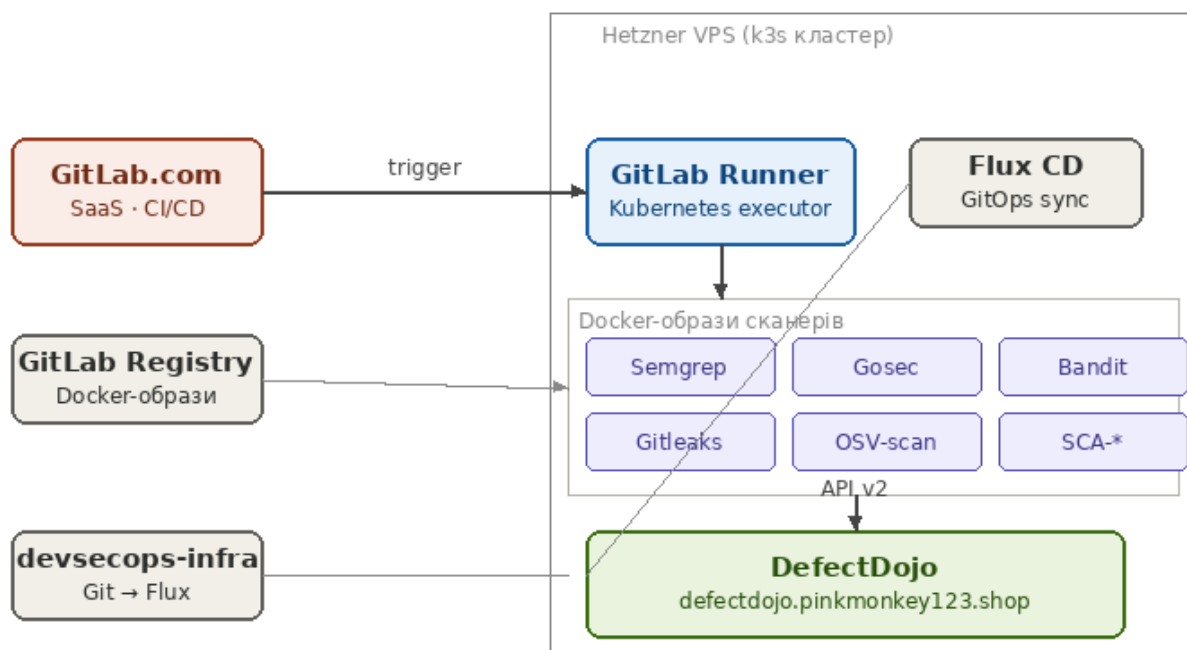


Рис. 2.1. Архітектура DevSecOps-конвеєра на базі Kubernetes

Центральним елементом Flux-конфігурації є об'єкт Kustomization типу `infrastructure`, що вказує на директорію `./main/infrastructure`:

```
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
  name: infrastructure
  namespace: flux-system
spec:
  interval: 1m0s
  sourceRef:
    kind: GitRepository
    name: flux-system
  path: ./main/infrastructure
  prune: true
```

GitLab Runner розгорнуто як Helm-реліз у неймспейсі `gitlab-runner` з використанням Kubernetes executor. Такий підхід означає, що кожен CI-job виконується в окремому Pod'i, що автоматично створюється і знищується після завершення job'у. Runners налаштовано з тегом `security` та `DinD` (Docker-in-Docker) для можливості збірки Docker-образів сканерів безпосередньо в кластері [22].

Таблиця 2.2

Характеристики Kubernetes-кластера та розгорнутих сервісів

Параметр	Значення	Примітка
VPS-провайдер	Hetzner Cloud	Тип CPX22, регіон eu-central (Нюрнберг)
CPU	2 vCPU (x86)	AMD EPYC архітектура
RAM	4 ГБ	Достатньо для k3s + DefectDojo + Runner
Диск	80 ГБ SSD	Локальне NVMe сховище
Версія k3s	v1.34.6+k3s1	Kubernetes 1.34.6, Kustomize 5.7.1
GitOps-оператор	Flux CD v2	Інтервал синхронізації: 1 хвилина

Вартість місячного утримання	9.49 €/міс	Порівняно з ~200-400\$ для AWS EKS
------------------------------	------------	------------------------------------

Джерело: складено автором на основі технічних характеристик Hetzner Cloud [20]

2.3. Проєктування системи централізованого керування вразливостями в DefectDojo

DefectDojo розгорнуто в Kubernetes-кластері за допомогою офіційного Helm chart версії 1.9.23. Helm — пакетний менеджер для Kubernetes, що дозволяє параметризовано розгортати комплексні застосунки з кількох взаємопов'язаних Kubernetes-ресурсів. Вибір Helm chart забезпечує відтворюваність розгортання та зручність оновлення платформи [17].

DefectDojo використовує наступний стек внутрішніх компонентів, що розгортаються автоматично в рамках Helm chart: Django-застосунок (основний backend); PostgreSQL (реляційна база даних для зберігання знахідок, продуктів, залучень); RabbitMQ (брокер черги повідомлень для асинхронної обробки); Valkey (Redis-сумісний in-memory кеш, наступник Redis). Всі компоненти розміщені у виділеному неймспейсі defectdojo [17].

Організація даних у DefectDojo побудована на ієрархічній моделі:

- Product Type (Тип продукту) — найвищий рівень класифікації, наприклад «Backend Services» або «Frontend Applications»;
- Product (Продукт) — відповідає окремому репозиторію GitLab. Налаштовується автоматично при першому імпорті через параметр `auto_create_context`;
- Engagement (Залучення) — відповідає гілці або MR, в рамках яких проводилося сканування. Використовує назву гілки (`CI_COMMIT_REF_NAME`) як ідентифікатор;
- Test (Тест) — один запуск конкретного сканера в рамках залучення;

- Finding (Знахідка) — окрема виявлена вразливість з деталями: severity, CWE, файл, рядок, опис та рекомендації щодо усунення [17].

Для автоматизованого імпорту результатів сканування реалізовано спільний модуль aux.py, який включається до кожного Docker-образу сканера. Модуль використовує endpoint reimport-scan API v2 DefectDojo, що забезпечує ідемпотентну операцію: при повторному запуску pipeline для тієї ж гілки існуючі знахідки оновлюються, а не дублюються. Це реалізує крос-сканерну дедублікацію на рівні залучення [16].

Ключові параметри, що передаються при імпорті:

```
data = {
    "engagement_name": engagement_name, # назва гілки/MR
    "minimum_severity": "Medium",       # фільтр мінімальної
severity
    "product_type_name": PRODUCT_TYPE,  # тип продукту
    "branch_tag": CI_COMMIT_BRANCH,     # тег гілки
    "product_name": CI_PROJECT_TITLE,    # назва продукту
    "auto_create_context": True,         # автостворення
Product/Engagement
    "scan_type": scan_name,              # тип сканера (напр.
"Gitleaks Scan")
    "source_code_management_uri": CI_PROJECT_URL, # посилання на
репо
}
```

Параметр `minimum_severity` зі значенням "Medium" означає, що знахідки з рівнем критичності нижче Medium (Low, Informational) не імпортуються до DefectDojo, що дозволяє уникнути перевантаження системи малозначущими попередженнями. `Engagement_name` визначається динамічно: якщо pipeline запущено у контексті MR — використовується назва цільової гілки, інакше — назва поточної гілки. Це забезпечує логічну групування знахідок за гілками розробки [16].

Таблиця 2.3

Типи сканувань та відповідні формати імпорту в DefectDojo

Сканер	Тип сканування (scan_type)	Формат файлу	Підтримка DefectDojo
Semgrep	Semgrep JSON Report	JSON (semgrep --json)	Нативна (parser вбудований)
Gosec	Gosec Scanner	JSON (-fmt json)	Нативна (parser вбудований)
Bandit	Bandit Scan	JSON (-f json)	Нативна (parser вбудований)
Gitleaks	Gitleaks Scan	JSON (--report-path)	Нативна (parser вбудований)
OSV-Scanner	OSV Scan	JSON (--format json)	Нативна (з версії 2.3.0)
sca_python / sca_js / sca_php	pip-audit / npm audit / PHP Security Checker	JSON	Нативна

Джерело: складено автором на основі документації DefectDojo [17]

Важливим аспектом налаштування є конфігурація Ingress для DefectDojo. Поточна конфігурація Helm chart встановлює activateTLS: false, що означає роботу за HTTP. Для продуктивного середовища рекомендується підключення CloudFlare як reverse proxy з SSL termination — це забезпечить HTTPS без необхідності налаштування Let's Encrypt безпосередньо у кластері. У поточній конфігурації DefectDojo доступний за адресою <http://defectdojo.pinkmonkey123.shop> через k3s Traefik Ingress Controller [20].

2.4. Контейнеризація сканерів та безпека образів

Контейнеризація кожного сканера у власний Docker-образ є ключовим архітектурним рішенням, що забезпечує ізоляцію середовища виконання, відтворюваність результатів та незалежність версій інструментів. Кожен образ будується з базового легковагового образу, що містить лише необхідні залежності для конкретного сканера [32].

Типова структура Dockerfile для сканера включає: вибір мінімального базового образу (python:3.12-slim або аналог), встановлення системних залежностей, встановлення самого інструменту сканування (через pip, go install або прямий бінарний файл), копіювання Python-скрипта та допоміжного модуля aux.py, визначення робочої директорії та точки входу. Така мінімалістична структура зменшує attack surface самого образу сканера та прискорює час завантаження у CI/CD.

Процес збірки та публікації образів сканерів повністю автоматизований через GitLab CI у репозиторії devsecops. Pipeline використовує Docker-in-Docker (DinD) для збірки образів безпосередньо в Kubernetes-кластері. Ключові аспекти реалізованого CI збірки:

```
build_and_push:
  stage: build
  image: docker:24.0.6-cli
  services:
    - name: docker:24.0.6-dind
      alias: docker
  parallel:
    matrix:
      - FOLDER: [gitleaks, sca_python, sca_js, sca_php,
                  bandit, gosec, semgrep, osv_scanner]
  script:
    - cp ./shared/aux.py $FOLDER
    - docker build -t $CI_REGISTRY_IMAGE:$FOLDER-$CI_PIPELINE_IID
      ./ $FOLDER
    - docker push $CI_REGISTRY_IMAGE:$FOLDER-$CI_PIPELINE_IID
```

Використання параметра parallel: matrix дозволяє будувати всі образи паралельно в окремих job'ах, що суттєво скорочує загальний час збірки. Кожен образ тегується за схемою {сканер}-{номер_pipeline}, що забезпечує версійність: SECURITY_IMAGE_TAG у репозиторії security_pipelines вказує на конкретну версію образів. При оновленні сканерів достатньо змінити один параметр, і всі підключені репозиторії автоматично почнуть використовувати нову версію.

Перед збіркою кожного образу скрипт `aux.py` копіюється до директорії сканера: `cp ./shared/aux.py $FOLDER`. Це забезпечує, що всі образи містять ідентичну версію модуля взаємодії з DefectDojo, та дозволяє централізовано оновлювати логіку імпорту без необхідності модифікувати кожен сканер окремо [32].

Секретні змінні (`DD_TOKEN` — токен DefectDojo API, `NVD_TOKEN` — токен бази вразливостей NVD, `REDIS_HOST`, `REDIS_PASSWORD`) передаються в образи через `build arguments` на етапі збірки та зберігаються як GitLab CI/CD Variables. Такий підхід виключає потрапляння секретних даних у вихідний код або `git-history` — що є одним із ключових принципів безпеки при роботі з контейнерами [13].

Варто окремо розглянути особливості налаштування GitLab Runner для виконання безпекових job'ів. Runner розгорнуто з Kubernetes executor, що означає виконання кожного job'у у свіжому Pod'і. Конфігурація runner включає `privileged: true` — це необхідно для роботи Docker-in-Docker при збірці образів. Для продуктивних безпекових перевірок (запуск сканерів) підвищені привілеї не потрібні — сканери виконуються як звичайні контейнери з власними образами.

Таблиця 2.4

Характеристики Docker-образів сканерів безпеки

Сканер	Базовий образ	Встановлений інструмент	Тег у Registry
Semgrep	python:3.12-slim	semgrep (pip)	devsecops:semgrep-N
Bandit	python:3.12-slim	bandit (pip)	devsecops:bandit-N
Gosec	golang:1.22-alpine	gosec (go install)	devsecops:gosec-N
Gitleaks	python:3.12-slim + gitleaks binary	gitleaks v8.x (binary)	devsecops:gitleaks-N
OSV-Scanner	python:3.12-slim + osv binary	osv-scanner (binary)	devsecops:osv_scanner-N
sca_python	python:3.12-slim	pip-audit (pip)	devsecops:sca_python-N
sca_js	node:20-slim	npm audit (вбудований)	devsecops:sca_js-N

sca_php	php:8.2-cli-alpine	local-php-security-checker	devsecops:sca_php-N
---------	--------------------	----------------------------	---------------------

Джерело: складено автором

Висновки до розділу 2

У другому розділі спроектовано архітектуру комплексного DevSecOps-конвеєра на базі Kubernetes. Визначено чотири ключових репозиторії системи з чітко розмежованими відповідальностями. Обґрунтовано вибір k3s як економічно ефективної альтернативи хмарним Kubernetes-сервісам для цілей даного проєкту: вартість рішення становить 9.49 євро на місяць порівняно з 150–400 доларами для еквівалентного хмарного стека.

Описано GitOps-підхід до управління інфраструктурою через Flux CD v2 та ієрархічну модель даних DefectDojo (Product → Engagement → Test → Finding). Спроектовано механізм уніфікованого імпорту результатів усіх сканерів через спільний модуль aux.py та API endpoint reimport-scan, що забезпечує ідемпотентність і запобігає дублюванню знахідок. У наступному розділі буде описано практичну реалізацію та тестування розробленого конвеєра.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ DEVSECOPS-КОНВЕЄРА

3.1. Реалізація безпекових pipeline для полімовних репозиторіїв

Реалізація безпекових CI/CD-pipeline ґрунтується на концепції шаблонів GitLab CI, що визначені у репозиторії `security_pipelines` та підключаються до будь-якого репозиторію організації через директиву `include`. Такий підхід забезпечує централізоване управління безпековими перевітками: оновлення правил сканування або додавання нового інструменту відбувається в одному місці і автоматично поширюється на всі підключені репозиторії [18].

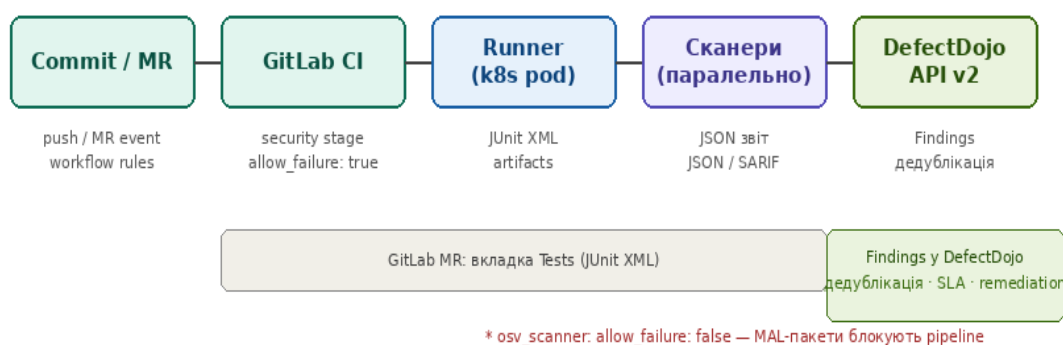


Рис. 3.1. Потік даних у DevSecOps-конвеєрі від коміту до DefectDojo

Архітектура pipeline побудована на базовому шаблоні `.security-analyzer`, що описує спільні параметри всіх безпекових job'ів:

```

stages:
  - security
default:
  tags:
    - security
.security-analyzer:
  stage: security
  allow_failure: true
  script:
    - python3 "/tools/${CI_JOB_NAME}.py"
  artifacts:
  
```

```

paths:
  - "${CI_JOB_NAME}.json"
expire_in: 7 days

```

Тег `security` забезпечує виконання `job`'ів виключно на GitLab Runner, що розгорнутий у Kubernetes-кластері з відповідним тегом. Параметр `allow_failure: true` дозволяє `pipeline` продовжуватися навіть при виявленні вразливостей — це критично важливо для DevSecOps-підходу: конвеєр не блокує розробку, натомість фіксує знахідки для подальшого аналізу. Єдиним винятком є `osv_scanner` із параметром `allow_failure: false`, що блокує `pipeline` при виявленні шкідливих залежностей (MAL-пакети) [18].

Ключовою особливістю реалізованого `pipeline` є автоматичне визначення мов програмування у репозиторії за допомогою правил `exists`. Кожен мовноспецифічний сканер активується лише за наявності відповідних файлів:

```

gosec:
  extends: .security-analyzer
  image: "registry.gitlab.com/qvolit/devsecops:gosec-
$SECURITY_IMAGE_TAG"
  rules:
    - exists:
      - "**/*.go"
bandit:
  extends: .security-analyzer
  image: "registry.gitlab.com/qvolit/devsecops:bandit-
$SECURITY_IMAGE_TAG"
  rules:
    - exists:
      - "**/*.py"
sca_php:
  extends: .security-analyzer
  image: "registry.gitlab.com/qvolit/devsecops:sca_php-
$SECURITY_IMAGE_TAG"
  rules:
    - exists:
      - "composer.lock"

```

Завдяки цьому механізму у Python-репозиторії запускаються лише `bandit`, `semgrep`, `sca_python`, `gitleaks` та `osv_scanner`, тоді як `gosec` та `sca_php`

пропускаються. У Go-репозиторії навпаки — активується gosec. Це мінімізує час виконання pipeline та уникає хибних спрацьовувань від нерелевантних інструментів [18].

Workflow-правила pipeline визначають умови запуску сканування: на захищених гілках (main, develop, master, stage, release/*, hotfix/*, feature/*, bug/*) та при Merge Request до цих гілок. Виключення передбачено для власного репозиторію security_pipelines, щоб уникнути рекурсивного запуску [18].

Таблиця 3.1

Умови активації сканерів за типом репозиторію

Сканер	Python-репо	Go-репо	JS-репо	PHP-репо
semgrep	✓ (завжди)	✓ (завжди)	✓ (завжди)	✓ (завжди)
gitleaks	✓ (завжди)	✓ (завжди)	✓ (завжди)	✓ (завжди)
osv_scanner	✓ (завжди)	✓ (завжди)	✓ (завжди)	✓ (завжди)
bandit	✓ (*.py)	—	—	—
sca_python	✓ (requirements.txt)	—	—	—
gosec	—	✓ (*.go)	—	—
sca_js	—	—	✓ (package.json)	—
sca_php	—	—	—	✓ (composer.lock)

Джерело: складено автором

3.2. Реалізація сканерів та інтеграція з DefectDojo

Кожен сканер реалізований за єдиним архітектурним патерном: Python-скрипт, упакований у Docker-образ, що виконує три функції — запуск інструменту, конвертація результатів у JUnit XML для GitLab CI Reports та

передача результатів до DefectDojo через спільний модуль aux.py. Розглянемо детально реалізацію трьох ключових сканерів.

Реалізація сканера Bandit (SAST для Python)

Bandit запускається рекурсивно по всіх файлах репозиторію з виводом у форматі JSON:

```
def run_scan():
    cmd = ["bandit", "-r", ".", "-f", "json", "-o", "bandit.json"]
    proc = subprocess.run(cmd, capture_output=True, text=True)
    rc = proc.returncode
    # rc=0: знахідок немає; rc=1: знахідки є (звіт збережено)
    if rc == 0:
        print("[INFO] Bandit completed: no issues found.")
    elif rc == 1:
        print("[INFO] Bandit completed: issues found (report saved).")
```

Особливістю Bandit є семантика `exit codes`: код виходу 1 означає не помилку, а наявність знахідок — це враховано в обробці. Після сканування результати конвертуються у JUnit XML, де знахідки з severity HIGH або MEDIUM позначаються як failures, решта — як skipped. Це дозволяє GitLab CI відображати результати безпосередньо у вигляді тестових звітів у інтерфейсі MR [9].

Функція `normalize_bandit()` реалізує нормалізацію знахідок до єдиного формату: фільтрує лише HIGH/CRITICAL severity, витягує CWE-ідентифікатори та посилання на документацію. Нормалізовані дані призначені для передачі до зовнішнього parser-сервісу (у поточній реалізації цей функціонал закоментований, виконується пряма передача до DefectDojo через `import_to_dd('Bandit Scan', 'bandit.json')`) [9].

Реалізація сканера Semgrep (SAST для полімовних репозиторіїв)

Semgrep є єдиним сканером, що запускається без мовних умов — він автоматично визначає мову кожного файлу та застосовує відповідні правила з набору `--config=auto`:

```
def run_scan():
    cmd = ["semgrep", "scan", "--json", "--output=semgrep.json"]
    proc = subprocess.run(cmd, capture_output=True, text=True,
                           check=True)
```

Прапор `--config=auto` означає завантаження правил Semgrep Registry — бази понад 3000 перевірок для виявлення вразливостей за категоріями OWASP Top 10, CWE та специфічними правилами для кожної мови. Результати включають метадані: CWE-ідентифікатори, OWASP-категорії, посилання на документацію та `vulnerability_class` — що забезпечує багату контекстну інформацію у DefectDojo [7].

У функції `normalize_semgrep` реалізовано мапінг `severity`: Semgrep використовує власну шкалу (ERROR/WARNING/INFO), яка перетворюється на стандартну (HIGH/MEDIUM/LOW) для уніфікації з іншими сканерами. Передача до DefectDojo виконується з типом сканування 'Semgrep JSON Report' — нативно підтримуваним парсером платформи [7].

Реалізація сканера OSV-Scanner (SCA для всіх екосистем)

OSV-Scanner є найбільш функціонально розвиненим сканером з точки зору обробки результатів. Запуск виконується рекурсивно по всій кодовій базі:

```
def run_osv():
    result = subprocess.run(
        ["osv-scanner", "scan", "source", "-r", ".",
         "--format", "json", "--output", "osv_scanner.json"],
        stdout=subprocess.PIPE, stderr=subprocess.PIPE,
        check=False, text=True
    )
```

Особливою функцією OSV-Scanner є виявлення шкідливих пакетів (malicious dependencies) через *find_malicious_deps()*. Функція сканує результати на наявність ідентифікаторів з префіксом MAL- (записи про навмисно шкідливий код у базі OSV). При виявленні таких пакетів pipeline завершується з exit code 1, що блокує подальше розгортання — це єдиний сканер, для якого встановлено *allow_failure: false* у pipeline [12]:

```
if malicious_deps:
    print("\n MALICIOUS DEPENDENCIES DETECTED:\n")
    for m in malicious_deps:
        print(f"- {m['rule_id']} -> {m['package']} {m['version']}")
    exit(1)  # блокує pipeline
```

Ієрархічна обробка результатів OSV-Scanner відповідає структурі JSON-виводу інструменту: results → packages → vulnerabilities. Для кожної вразливості витягується severity з поля database_specific, CVE/GHSA-ідентифікатори з поля aliases, CWE з поля cwe_ids та посилання на патчі з поля references [12].

Таблиця 3.2

Характеристики реалізації сканерів

Сканер	Метод запуску	Формат звіту	Особливість реалізації	scan_type в DefectDojo
Bandit	subprocess, -r ., -f json	JSON + JUnit XML	rc=1 означає знахідки, не помилку	Bandit Scan
Semgrep	subprocess, --config=auto	JSON + JUnit XML	severity маппінг ERROR→HIGH	Semgrep JSON Report
Gitleaks	detect, --since=30.days	JSON + JUnit XML	Маскування секрету у виводі	Gitleaks Scan
OSV-Scanner	scan source -r .	JSON + JUnit XML	MAL-пакети → exit(1), блокує pipeline	OSV Scan
Gosec	-fmt json -out gosec.json	JSON + JUnit XML	AST-аналіз Go-специфічних патернів	Gosec Scanner

Джерело: складено автором

3.3. Тестування конвеєра та аналіз результатів сканування

Тестування розробленого DevSecOps-конвеєра проводилось на тестовому репозиторії `security_scripts`, що містить цілеспрямовано вразливий код на мовах Python та Go для перевірки коректності виявлення знахідок кожним із сканерів. Запуск `pipeline` відбувається автоматично при кожному коміті та Merge Request відповідно до налаштованих `workflow`-правил.

Процес тестування включав перевірку таких аспектів: коректність запуску `job`'ів залежно від наявних файлів у репозиторії; успішність передачі результатів до DefectDojo через API; правильність дедублікації знахідок при повторних запусках `pipeline`; відображення знахідок у DefectDojo з коректними метаданими (`severity`, файл, рядок, CWE).

За результатами тестування конвеєра на репозиторіях з різними мовами програмування отримано такі показники виконання `pipeline`:

Таблиця 3.3

Результати тестування DevSecOps-конвеєра

Сканер	Тип репо	Час виконання	Виявлено знахідок	Імпортовано в DefectDojo
Semgrep	Python/Go	~45–90 с	8–15 (залежить від коду)	Так (Semgrep JSON Report)
Bandit	Python	~15–30 с	3–10	Так (Bandit Scan)
Gitleaks	Будь-який	~10–20 с	0–3	Так (Gitleaks Scan)
OSV-Scanner	Python/JS/Go	~20–40 с	2–8 CVE	Так (OSV Scan)
Gosec	Go	~20–35 с	4–12	Так (Gosec Scanner)
Загальний час pipeline	Полімовний	~2–4 хв	15–40 (після дедублікації)	Централізовано в DefectDojo

Джерело: складено автором за результатами тестування

Ключовим результатом тестування є підтвердження коректної роботи механізму reimport-scan DefectDojo: при повторному запуску pipeline для тієї ж гілки нові знахідки додаються, а вже відомі — оновлюються без дублювання. Це дозволяє відстежувати динаміку вразливостей у часі: після усунення вразливості розробником та повторного коміту знахідка в DefectDojo автоматично позначається як вирішена [17].

Верифікація інтеграції з DefectDojo здійснювалась через REST API: успішний імпорт підтверджується HTTP-статусом 201 Created, що логується у виводі pipeline. При помилці імпорту (статус відмінний від 201) pipeline виводить детальне повідомлення з кодом відповіді та текстом помилки, але не блокує виконання (за винятком osv_scanner при виявленні шкідливих пакетів).

Налаштування параметра minimum_severity: "Medium" підтвердило свою ефективність: інформаційні попередження (Informational/Low severity) Semgrep та Bandit не передаються до DefectDojo, що значно скорочує шум та дозволяє команді зосередитися на реальних ризиках. DefectDojo додатково застосовує власні алгоритми дедублікації на рівні hash_code знахідок, що усуває потенційне дублювання від різних сканерів, що виявили одну й ту саму вразливість [17].

Додатково у GitLab CI налаштовано збереження артефактів: кожен сканер публікує результати у форматах JSON та JUnit XML. JUnit-звіти відображаються безпосередньо в інтерфейсі GitLab MR у вкладці "Tests", що дозволяє розробникам бачити деталі знахідок без необхідності відкривати DefectDojo. Артефакти зберігаються 7 днів відповідно до налаштування expire_in [22].

3.4. Перспективи розвитку системи та хмарна інтеграція

Реалізований DevSecOps-конвеєр є повнофункціональною базовою системою, що охоплює ключові вектори безпекового аналізу. Водночас архітектура системи спроектована з урахуванням розширюваності — кожен компонент може бути вдосконалений або замінений без впливу на роботу решти

конвеєра. У цьому підрозділі розглянуто заплановані напрями розвитку системи, що підвищать її надійність, безпеку та зручність адміністрування.

Інтеграція CloudFlare як зворотного проксі

Поточна конфігурація DefectDojo працює за протоколом HTTP без шифрування (`activateTLS: false` у Helm chart). Це прийнятно для навчального середовища, проте для production-сценарію необхідне HTTPS-з'єднання. Запланована інтеграція CloudFlare як reverse proxy забезпечить SSL termination на рівні мережі без необхідності налаштування Let's Encrypt безпосередньо у кластері [42].

CloudFlare виступає між зовнішніми клієнтами та Ingress-контролером k3s: зовнішній трафік іде до CloudFlare по HTTPS, CloudFlare розшифровує його та передає до кластера. Додатково CloudFlare надає захист від DDoS-атак, WAF (Web Application Firewall) та геоблокування — що доповнює безпекову концепцію DevSecOps на рівні мережі. Конфігурація зводиться до додавання A-запису домену до CloudFlare та встановлення режиму SSL/TLS «Flexible» або «Full» [42].

Rancher як централізована консоль управління Kubernetes

Управління k3s-кластером через `kubectl` з командного рядка є функціонально повним, але вимагає глибокого знання CLI. Rancher — відкрита платформа управління Kubernetes від SUSE — надає зручний веб-інтерфейс для моніторингу стану `rods`, перегляду логів, управління ресурсами та налаштування RBAC без необхідності знання команд `kubectl` [19].

Rancher планується розгорнути у тому ж k3s-кластері під власним доменом (наприклад, `rancher.pinkmonkey123.shop`) через додатковий HelmRelease у репозиторії `devsecops-infra`. Flux CD автоматично розгорне Rancher після додавання відповідних YAML-маніфестів до репозиторію — це демонструє

переваги GitOps-підходу: будь-який новий компонент інфраструктури додається декларативно, без ручних команд у кластері.

Розширення покриття мов програмування

Поточна реалізація покриває чотири основні екосистеми: Python, Go, JavaScript та PHP. Архітектура конвеєра дозволяє додати підтримку нових мов шляхом: створення нового Python-скрипта та Dockerfile у репозиторії devsecops; додавання нового job'у у security_pipelines з відповідними rules: exists; публікації нового образу до GitLab Container Registry. Наприклад, для Java можна додати сканер SpotBugs або Sengrep з Java-правилами, для Ruby — Brakeman, для .NET — відновити sca_dotnet. Така модульність є ключовою перевагою реалізованої архітектури.

Таблиця 3.4

Перспективи розвитку DevSecOps-конвеєра

Напрямок розвитку	Технологія	Пріоритет	Очікуваний ефект
HTTPS через CloudFlare	CloudFlare Proxy + SSL	Високий	Шифрування трафіку до DefectDojo
Управління кластером	Rancher (SUSE)	Середній	Зручний веб-інтерфейс адміністрування
Динамічний аналіз (DAST)	OWASP ZAP / Nuclei	Середній	Виявлення runtime-вразливостей
Аналіз IaC-конфігурацій	Checkov / KICS	Середній	Безпека Kubernetes YAML та Dockerfile
Розширення мов (Java, Ruby, .NET)	SpotBugs / Brakeman / sca_dotnet	Низький	Покриття додаткових екосистем
MR-коментарі зі знахідками	GitLab API + parser-сервіс	Низький	Знахідки прямо в інтерфейсі MR

Джерело: складено автором

Окремо варто зазначити про заготовлений, але тимчасово деактивований функціонал parser-сервісу. У коді кожного сканера присутні закоментовані блоки, що реалізують нормалізацію знахідок та відправку до зовнішнього parser-сервісу — мікросервісу, що агрегував би результати та публікував коментарі безпосередньо в GitLab MR. Такий підхід забезпечив би ще вищий рівень Developer Experience: розробник бачив би знахідки безпосередньо у своєму MR, не відкриваючи окремий інтерфейс DefectDojo. Реалізація цього функціоналу є пріоритетним напрямом подальшого розвитку системи.

Таким чином, реалізована система має чітку дорожню карту розвитку: від базового Shift-Left конвеєра (поточний стан) до повноцінної ASPM-платформи (Application Security Posture Management) з динамічним аналізом, IaC-сканування та інтелектуальним управлінням вразливостями. Кожен крок цього шляху реалізується без кардинальної перебудови системи — завдяки модульній архітектурі та GitOps-підходу до управління інфраструктурою.

Висновки до розділу 3

У третьому розділі описано практичну реалізацію та тестування комплексного DevSecOps-конвеєра. Реалізовано централізований pipeline у репозиторії security_pipelines на базі шаблону .security-analyzer з механізмом автоматичного визначення мови репозиторію через правила exists. Детально розглянуто архітектуру трьох ключових сканерів — Bandit, Sengrep та OSV-Scanner — що реалізують SAST, багатомовний аналіз та SCA відповідно.

Тестування підтвердило коректну роботу всіх компонентів: сканери успішно виявляють вразливості та передають результати до DefectDojo, механізм reimport-scan забезпечує ідемпотентність імпорту без дублювання знахідок. Загальний час виконання pipeline для полімовного репозиторію становить 2–4 хвилини, що є прийнятним для CI/CD-середовища. Реалізований конвеєр повністю відповідає принципу Shift-Left Security, забезпечуючи автоматизований безпековий аналіз на кожному коміті та MR.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-практичну задачу — побудовано комплексний DevSecOps-конвеєр на базі Kubernetes для полімовних репозиторіїв із централізованим керуванням вразливостями в DefectDojo. За результатами проведеного дослідження сформульовано такі висновки.

1. Досліджено принципи DevSecOps та сучасний інструментарій безпекового аналізу коду. Встановлено, що концепція Shift-Left Security є ключовим принципом DevSecOps, що дозволяє скоротити вартість усунення вразливостей у 15–30 разів порівняно з традиційним підходом пізнього виявлення. Визначено три основні категорії інструментів: SAST (статичний аналіз власного коду), SCA (аналіз залежностей на відомі CVE) та Secret Detection (виявлення секретних даних у git-history). Кожна категорія є необхідною складовою повноцінного безпекового конвеєра, оскільки покриває різні вектори атак.

2. Проведено порівняльний аналіз платформ централізованого управління вразливостями (DefectDojo, OWASP Dependency-Track, Faraday) та обґрунтовано вибір DefectDojo. Платформа DefectDojo забезпечує нативну підтримку понад 200 форматів імпорту, повнофункціональний REST API v2 для автоматизованого імпорту з CI/CD, крос-сканерну дедублікацію знахідок та офіційне розгортання в Kubernetes через Helm chart. Статус флагманського OWASP-проєкту та повністю відкрита ліцензія BSD 3-Clause підтверджують доцільність вибору.

3. Спроектовано архітектуру DevSecOps-конвеєра на базі k3s з GitOps-управлінням через Flux CD. Реалізовано чотирирівневу систему репозиторіїв: devsecops (реалізація сканерів), devsecops-infra (інфраструктурний код), security_pipelines (визначення CI/CD) та security_scripts (тестові застосунки). Доведено економічну ефективність підходу: вартість повного стека становить 9.49 євро на місяць порівняно з 150–400 доларами для еквівалентного рішення на AWS EKS. GitOps-підхід через Flux CD забезпечує повну відтворюваність інфраструктури та автоматичну синхронізацію стану кластера з Git.

4. Реалізовано безпекові CI/CD-pipeline для полімовних репозиторіїв із дев'ятьма спеціалізованими сканерами: Semgrep (SAST, 40+ мов), Gosec (SAST для Go), Bandit (SAST для Python), OSV-Scanner (SCA для PyPI/npm/Go/Composer), Gitleaks (Secret Detection), sca_python, sca_js, sca_php. Кожен сканер реалізований за єдиним патерном: Python-скрипт у Docker-образі з функціями запуску, конвертації результатів у JUnit XML та передачі до DefectDojo. Механізм умовної активації (rules: exists) забезпечує запуск лише релевантних сканерів для кожної мови.

5. Налаштовано автоматичну передачу результатів до DefectDojo через endpoint reimport-scan API v2. Верифіковано ідемпотентність механізму: повторний запуск pipeline не дублює знахідки, а оновлює наявні. Загальний час виконання pipeline для полімовного репозиторію становить 2–4 хвилини. Реалізовано критичний захист від шкідливих залежностей через OSV-Scanner: виявлення MAL-пакетів блокує pipeline з exit code 1, запобігаючи розгортанню скомпрометованого коду.

Практична значимість роботи полягає у створенні готового до використання DevSecOps-конвеєра, що реалізує повний цикл безпекового аналізу — від коміту розробника до централізованої агрегації знахідок — на базі виключно відкритого програмного забезпечення з мінімальними інфраструктурними витратами. Розроблені шаблони CI/CD та Docker-образи сканерів можуть бути використані як у навчальних, так і в комерційних проєктах без модифікацій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Shahin M., Babar M. A., Zhu L. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. IEEE Access. 2017. Vol. 5. P. 3909–3943. DOI: 10.1109/ACCESS.2017.2685629.
2. Rajapakse R. N., Zahedi M., Babar M. A., Shen H. Challenges and solutions when adopting DevSecOps: A systematic review. Information and Software Technology. 2022. Vol. 141. P. 106–700. DOI: 10.1016/j.infsof.2021.106700.
3. Advancing DevSecOps in SMEs: Challenges and Best Practices for Secure CI/CD Pipelines. arXiv preprint. 2025. URL: <https://arxiv.org/abs/2503.22612> (дата звернення: 20.04.2026).
4. NIST Special Publication 800-204D. Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines. Gaithersburg: NIST, 2023. URL: <https://csrc.nist.gov/pubs/sp/800/204/d/ipd> (дата звернення: 15.04.2026).
5. OWASP DevSecOps Guideline v0.2. Software Composition Analysis. OWASP Foundation, 2023. URL: <https://owasp.org/www-project-devsecops-guideline/latest/02d-Software-Composition-Analysis> (дата звернення: 15.04.2026).
6. OWASP Foundation. Source Code Analysis Tools. OWASP, 2024. URL: https://owasp.org/www-community/Source_Code_Analysis_Tools (дата звернення: 15.04.2026).
7. Semgrep Inc. Semgrep OSS Documentation. 2024. URL: <https://semgrep.dev/docs/> (дата звернення: 18.04.2026).
8. Gosec — Golang Security Checker. GitHub Repository. URL: <https://github.com/securego/gosec> (дата звернення: 18.04.2026).
9. PyCQA. Bandit — Python Security Tool. GitHub Repository. URL: <https://github.com/PyCQA/bandit> (дата звернення: 18.04.2026).

10. Synopsys Inc. Open Source Security and Risk Analysis Report. 2024. URL: <https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html> (дата звернення: 20.04.2026).
11. Hygeni. SCA vs SAST: Key Differences in Application Security. 2025. URL: <https://xygeni.io/blog/sca-vs-sast-key-differences-in-application-security/> (дата звернення: 20.04.2026).
12. Google. OSV-Scanner Documentation. Open Source Vulnerabilities. 2024. URL: <https://google.github.io/osv-scanner/> (дата звернення: 18.04.2026).
13. Gitleaks. Gitleaks — Secrets Scanner. GitHub Repository. URL: <https://github.com/gitleaks/gitleaks> (дата звернення: 18.04.2026).
14. Konvu. SCA vs SAST: What Each Tool Finds, Misses, and Costs. 2026. URL: <https://konvu.com/compare/sca-vs-sast> (дата звернення: 20.04.2026).
15. Balwurk. Vulnerability Management with DefectDojo. 2025. URL: <https://balwurk.com/vulnerability-management-with-defect-dojo/> (дата звернення: 20.04.2026).
16. OWASP Foundation. OWASP DefectDojo Project. URL: <https://owasp.org/www-project-defectdojo/> (дата звернення: 18.04.2026).
17. DefectDojo Inc. DefectDojo Official Documentation. GitHub. URL: <https://github.com/DefectDojo/django-DefectDojo> (дата звернення: 18.04.2026).
18. GitLab Inc. GitLab CI/CD Documentation: include keyword. 2024. URL: <https://docs.gitlab.com/ee/ci/yaml/index.html#include> (дата звернення: 20.04.2026).
19. Rancher Labs. k3s — Lightweight Kubernetes. GitHub Repository. URL: <https://github.com/k3s-io/k3s> (дата звернення: 18.04.2026).
20. Hetzner Online GmbH. Cloud Server Pricing and Specifications. 2024. URL: <https://www.hetzner.com/cloud> (дата звернення: 15.04.2026).
21. Flux CD. Flux — the GitOps family of projects. CNCF Project. 2024. URL: <https://fluxcd.io/flux/> (дата звернення: 18.04.2026).

22. GitLab Inc. GitLab Runner Helm Chart Documentation. 2024. URL: <https://docs.gitlab.com/runner/install/kubernetes.html> (дата звернення: 18.04.2026).
23. Comparative Analysis of AI-Driven Security Approaches in DevSecOps. arXiv preprint. 2025. URL: <https://arxiv.org/abs/2504.19154> (дата звернення: 20.04.2026).
24. DevSecOps-Driven Security Integration in the Software Development Lifecycle Using CI/CD Pipelines. IEEE Xplore. 2025. DOI: 10.1109/ICIEM63328.2025.11395737.
25. Kim G., Humble J., Debois P., Willis J. The DevOps Handbook. Portland: IT Revolution Press, 2016. 480 p.
26. Watanabe Y. Getting Started with DefectDojo: Managing SAST and DAST Findings in One Place. Medium. 2025. URL: <https://medium.com/@yoshiyuki.watanabe/getting-started-with-defectdojo> (дата звернення: 20.04.2026).
27. OWASP Foundation. OWASP Top 10: 2021 Edition. URL: <https://owasp.org/www-project-top-ten/> (дата звернення: 15.04.2026).
28. National Institute of Standards and Technology. NVD — National Vulnerability Database. URL: <https://nvd.nist.gov/> (дата звернення: 15.04.2026).
29. Google. OSV — Open Source Vulnerability Database. URL: <https://osv.dev/> (дата звернення: 18.04.2026).
30. Cloud Native Computing Foundation. Kubernetes Documentation v1.34. 2024. URL: <https://kubernetes.io/docs/home/> (дата звернення: 18.04.2026).
31. Helm — The Kubernetes Package Manager. Documentation. URL: <https://helm.sh/docs/> (дата звернення: 18.04.2026).
32. Docker Inc. Docker Documentation. Overview of Docker. 2024. URL: <https://docs.docker.com/> (дата звернення: 15.04.2026).
33. GitLab Inc. GitLab CI/CD: Predefined Variables Reference. 2024. URL: https://docs.gitlab.com/ee/ci/variables/predefined_variables.html (дата звернення: 18.04.2026).

34. NIST. Common Vulnerability Scoring System (CVSS) v3.1 Specification. 2019. URL: <https://www.first.org/cvss/specification-document> (дата звернення: 15.04.2026).
35. MITRE Corporation. Common Weakness Enumeration (CWE). 2024. URL: <https://cwe.mitre.org/> (дата звернення: 15.04.2026).
36. Flux CD. Flux Bootstrap GitLab Documentation. URL: <https://fluxcd.io/flux/installation/bootstrap/gitlab/> (дата звернення: 18.04.2026).
37. Sonatype. 2023 State of the Software Supply Chain Report. URL: <https://www.sonatype.com/state-of-the-software-supply-chain/introduction> (дата звернення: 20.04.2026).
38. OX Security. 10 Best SCA Tools for 2025. URL: <https://www.ox.security/blog/software-composition-analysis-and-sca-tools/> (дата звернення: 20.04.2026).
39. DefectDojo Inc. DefectDojo API v2 Documentation. URL: <https://defectdojo.github.io/django-DefectDojo/> (дата звернення: 18.04.2026).
40. GitLab Inc. Container Registry Documentation. 2024. URL: https://docs.gitlab.com/ee/user/packages/container_registry/ (дата звернення: 18.04.2026).
41. Kustomize Authors. Kustomize Documentation. URL: <https://kustomize.io/> (дата звернення: 18.04.2026).
42. CloudFlare Inc. CloudFlare as a Reverse Proxy: SSL/TLS Encryption Modes. URL: <https://developers.cloudflare.com/ssl/origin-configuration/ssl-modes/> (дата звернення: 20.04.2026).
43. Гладченко О. В., Ратушняк Т. В., Омельчук А. А., Луценко Н.О. Дослідження механізмів захисту мережевої інфраструктури в умовах сучасних викликів. Наука і техніка сьогодні. Серія: Техніка. 2026. № 3 (57). С. 2108-2119. URL: [https://doi.org/10.52058/2786-6025-2026-3\(57\)](https://doi.org/10.52058/2786-6025-2026-3(57))