Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ СИСТЕМИ БЕЗПЕКИ ЗА ДОПОМОГОЮ ОРКЕСТРАЦІЇ ЛЕГКИХ КОНТЕЙНЕРНИХ ПЛАТФОРМ

IMPLEMENTATION AND RESEARCH OF A SECURITY SYSTEM USING ORCHESTRATION OF LIGHTWEIGHT CONTAINER PLATFORMS

спеціальність 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КНм-21
Ясашний Денис Вікторович
_____
(підпис)
Керівник: к.т.н., доцент
Кошелюк Віктор Андрійович
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Валерій Олександрович
_____
(підпис)

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерних наук*
Ступінь вищої освіти: *магістр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *122 Комп'ютерні науки*
Освітня програма: *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Валерій ЛІЩИНА
«14» травня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Ясашному Денису Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Реалізація та дослідження системи безпеки за допомогою оркестрації легких контейнерних платформ»
Керівник к.т.н., доцент Кошелюк Віктор Андрійович
затверджені наказом закладу вищої освіти від «*14*» *травня* *2025* р. № 255/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«05» грудня 2025 р.*

3. Вихідні дані до роботи: _____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проектування, опис функціонального наповнення об'єкта проектування, розробка й обґрунтування системного наповнення, експерементальне дослідження результативності предмету дослідження.
_____

5. Перелік графічного матеріалу:

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблематики за темою роботи та постановка завдань дослідження	*Кошелюк В.А.*		
Теоретичне дослідження та практична реалізація модулю безпеки інформації	*Кошелюк В.А.*		
Експерементальне дослідження модулю безпеки інформації	*Кошелюк В.А.*		
Показник запозичень тексту	_____ %		
Інструментальна перевірка	*Кошелюк В. А.*		
Нормоконтроль	*Сачук В. О.*		
Гарант ОПП	*Ліщина В. О.*		

7. Дата видачі завдання *«14» травня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	*Провести огляд літературних джерел по темі кваліфікаційної роботи*	*до 30.06.2025 р*	
2	*Провести аналіз загальної проблеми і вибір напрямків дослідження*	*до 01.09.2025 р.*	
3	*Розробити функціональну схему роботи модулю безпеки інформації*	*до 01.10.2025 р*	
4	*Описати засоби розробки об'єкта проектування*	*до 15.10.2025 р.*	
5	*Практична реалізація модулю безпеки інформації*	*до 10.11.2025 р.*	
6	*Провести експерементальне дослідження модулю безпеки інформації*	*до 25.11.2025 р.*	
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*до 05.12.2025 р.*	

Здобувач вищої освіти _____ Денис ЯСАШНИЙ

Керівник роботи _____ Віктор КОШЕЛЮК

АНОТАЦІЯ

Ясашний Д. В. Реалізація та дослідження системи безпеки за допомогою оркестрації легких контейнерних платформ. Рукопис. Кваліфікаційна робота магістра за спеціальністю 122 Комп'ютерні науки. Луцький національний технічний університет. Луцьк, 2025. 55 с.

Кваліфікаційна робота магістра складається з вступу, трьох розділів, висновків, списку використаних джерел, додатків.

У роботі наведено реалізацію та здійснено дослідження системи безпеки за допомогою оркестрації легких контейнерних платформ. Під час виконання поставлених завдань було проаналізовано проблематику системи безпеки оркестрації легких контейнерних платформ; здійснено теоретичне дослідження та практична реалізацію модулю безпеки інформації; запропоновано методику проведення дослідження та виконано обробку отриманих результатів.

Ключові слова: benchmark, kubernetes, аудит, безпека, оркестрація.

ANNOTATION

Denis Yasashny. Implementation and research of a security system using orchestration of lightweight container platforms. Manuscript. Master's Qualification Thesis in the field of 122 Computer Science. Lutsk National Technical University, 2025. 55 pages.

The master's thesis consists of an introduction, three sections, conclusions, a list of used sources, appendices.

The paper presents the implementation and research of a security system using orchestration of lightweight container platforms. During the implementation of the tasks, the issues of the security system of orchestration of lightweight container platforms were analyzed; theoretical research and practical implementation of the information security module were carried out; a research methodology was proposed and the results were processed.

Keywords: benchmark, kubernetes, audit, security, orchestration.

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасних інформаційних технологіях контейнеризація стала одним із ключових підходів до розгортання та підтримки застосунків. Завдяки контейнерам забезпечується гнучкість, портативність та ефективне використання ресурсів. Для управління контейнерними середовищами широко використовуються легкі платформи оркестрації, такі як minikube, microk8s, K3s, K0s та kind. Вони дозволяють швидко створювати та керувати невеликими кластерами, що особливо важливо для тестових, навчальних або ресурсно обмежених систем.

Важливість систем безпеки для оркестрації легких контейнерних платформ зумовлена стрімким розвитком контейнеризації та мікросервісної архітектури в сучасних IT-інфраструктурах. Застосування таких платформ дозволяє розробникам і адміністраторам швидко розгортати, тестувати та масштабувати додатки в ізольованих середовищах, зменшуючи ресурсоємність і час на конфігурацію. Водночас зростає кількість потенційних загроз, пов’язаних із неправильною конфігурацією, вразливими образами контейнерів, недостатнім контролем доступу та відсутністю централізованого моніторингу безпеки.

Забезпечення безпечної оркестрації легких контейнерних систем є критично важливим, адже навіть у тестових або локальних середовищах компрометація може призвести до витоку даних, порушення цілісності інфраструктури чи несанкціонованого доступу до мережевих ресурсів. В умовах переходу до DevSecOps-підходу актуальним стає інтегрування політик безпеки безпосередньо в процеси розгортання контейнерів.

Таким чином, дослідження системи безпеки для оркестрації легких контейнерних платформ є важливим кроком для підвищення надійності, стійкості та відповідності сучасним стандартам захисту інформації у гнучких хмарних середовищах.

Об’єктом дослідження є процеси забезпечення безпеки в контейнерних середовищах оркестрації, побудованих на основі легких рішень типу minikube, microk8s, K3s, K0s і kind.

Предметом дослідження є методи та інструменти реалізації системи безпеки, що забезпечують контроль доступу, моніторинг подій, аудит і шифрування у середовищах оркестрації контейнерів з обмеженими ресурсами.

Метою магістерської роботи є розроблення та дослідження системи безпеки, орієнтованої на підвищення рівня захищеності легких контейнерних платформ за рахунок впровадження комплексних механізмів контролю доступу, моніторингу, журналювання та виявлення загроз.

Завдання дослідження. Для досягнення поставленої мети необхідно виконати такі завдання:

– провести аналіз існуючих підходів до забезпечення безпеки в середовищах Kubernetes та їх спрощених версіях;

– ідентифікувати основні загрози для легких систем оркестрації контейнерів, зокрема несанкціонований доступ до API-сервера, компрометацію секретів, міжконтейнерні атаки та помилки конфігурацій;

– розробити архітектуру системи безпеки, що включатиме модулі автентифікації, авторизації, аудиту та шифрування;

– провести практичне тестування прототипу системи на прикладі однієї або кількох легких платформ (Minikube, K3s) з оцінкою її ефективності та впливу на продуктивність.

Методи дослідження. У процесі роботи планується використати методи аналізу архітектури систем оркестрації, експериментального моделювання, а також інструменти контейнеризації та моніторингу безпеки. Буде застосовано засоби Kubernetes CLI та фреймворки для тестування безпеки контейнерів.

Наукова новизна роботи полягає у комплексному підході до розроблення та впровадження системи безпеки для оркестрації легких контейнерних платформ типу minikube, microk8s, K3s, K0s, kind, що поєднує принципи мікросервісної архітектури, автоматизації DevSecOps та політик Zero Trust.

Запропоновано уніфіковану модель безпеки, яка враховує специфіку розподілених середовищ з обмеженими ресурсами, забезпечуючи динамічний контроль доступу, моніторинг подій безпеки та автоматизоване реагування на інциденти в межах контейнерної оркестрації.

Практична цінність дослідження полягає у створенні рекомендацій і прототипів рішень, які можуть бути безпосередньо застосовані в освітніх, наукових та промислових середовищах для побудови безпечних лабораторій, DevOps-процесів і тестових середовищ. Реалізація запропонованої системи підвищує надійність, прозорість та контроль у роботі контейнерних платформ, сприяючи зниженню ризиків кібератак і оптимізації процесів розгортання сервісів. Результати дослідження матимуть практичну цінність для організацій, що впроваджують контейнерні технології у DevOps або навчальних середовищах, де важливі баланс між безпекою, простотою розгортання та ефективним використанням ресурсів.

Апробація результатів дослідження:

– 3 Міжнародна науково-практична конференція «Modern Trends in the Development of Economy, Technology and Industry» (9-11 квітня 2025 р.), Торонто, Канада. International Scientific Unity. 2025 [1].

– XV Міжнародна науково-практична конференція «Scientific research: integration of science and practice for effective development» (15-18 квітня 2025 р.), Флоренція, Італія. International Science Group. 2025 [2].

– Андрущак І., Кошелюк В., Ясашний Д. Підвищення безпеки контейнерів за допомогою розгортання honeypot. International Science Journal of Engineering and Agriculture. 4(3). ISJEA, 2025. Pp. 15-26 [3].

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ СИСТЕМИ БЕЗПЕКИ ТА ПОСТАНОВКА ЗАВДАННЯ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результати існуючих теоретичних та експериментальних досліджень

У сучасному технологічному середовищі розгортання та управління мікросервісами і контейнеризованими додатками перетворилося на складне багатовимірне завдання, що вимагає координації численних компонентів розподіленої системи. Зростаюча складність архітектур, необхідність забезпечення безперервної доступності сервісів та динамічне масштабування створюють унікальні виклики для DevOps-команд та інженерів.

Оркестрація контейнерів відіграє ключову роль у перетворенні складних операційних процесів на керовані та ефективні рішення для команд розробки, експлуатації та DevOps-фахівців. Вона надає декларативний підхід до автоматизації переважної більшості рутинних завдань, що дозволяє командам зосередитися на стратегічних аспектах розробки замість технічних деталей інфраструктури.

Такий підхід особливо цінний для DevOps-команд та організацій, які культивують гнучкість і швидкість у своїй роботі, прагнучи значно перевершити традиційні методології розробки програмного забезпечення за темпами впровадження інновацій. Автоматизація повторюваних операцій, які раніше вимагали постійного ручного втручання, суттєво підвищує продуктивність DevOps-спеціалістів.

Завдяки оркестрації контейнерів, такі процеси як розгортання, масштабування, моніторинг та управління застосунками виконуються автоматично за заздалегідь визначеними правилами. Це не лише скорочує ймовірність людських помилок, але й забезпечує стабільність та передбачуваність у роботі системи, створюючи основу для безперервної інтеграції та доставки програмного забезпечення [4].

Оркестрація виступає ключовим елементом сучасних інфраструктур, спрощуючи цей процес та забезпечуючи ефективне функціонування всієї екосистеми. Вона надає можливість автоматизованого розподілу обчислювальних ресурсів, інтелектуального керування життєвим циклом контейнерів – від створення та запуску до моніторингу, оновлення та завершення роботи. Це робить процеси розробки та експлуатації додатків значно ефективнішими, надійнішими та передбачуванішими.

Управління і оркестрація представляють собою взаємопов’язані процеси, які дозволяють координувати, контролювати та автоматизувати різноманітні компоненти системи, забезпечуючи їх злагоджену роботу як єдиного цілого. Сучасна індустрія програмного забезпечення демонструє стійку тенденцію до використання мікросервісних архітектур, де монолітні системи розбиваються на автономні, слабко пов’язані компоненти мікросервіси, які можуть розроблятися, розгортатися та масштабуватися незалежно один від одного, взаємодіючи через чітко визначені інтерфейси.

Клієнт-серверна архітектура являє собою фундаментальну модель організації розподілених систем, в якій один або декілька клієнтських комп’ютерів встановлюють з’єднання з центральним сервером через мережеву інфраструктуру. У цій моделі клієнти ініціюють запити до сервера, який обробляє їх відповідно до бізнес-логіки та повертає результати назад клієнтам. Центральний сервер виступає головним координатором, що зберігає дані, виконує обчислення та забезпечує спільний доступ до ресурсів.

Така архітектура забезпечує ефективний розподіл обчислювального навантаження між клієнтськими машинами та потужним сервером, що оптимізує використання ресурсів. Вона також значно спрощує процеси розробки, впровадження та подальшого супроводу програмного забезпечення, оскільки основна логіка концентрується в одному місці. Однак централізований підхід може створювати єдину точку відмови та обмежувати горизонтальне масштабування системи.

Мікросервісна архітектура представляє собою сучасний еволюційний підхід, де монолітна система розбивається на безліч невеликих, автономних сервісів. Кожен мікросервіс функціонує незалежно, володіє власною базою даних та відповідає за конкретну бізнес-функцію. Взаємодія між сервісами здійснюється через легковагові протоколи, такі як REST API або повідомлення.

Ключова перевага мікросервісів полягає в можливості незалежного розгортання, масштабування та оновлення кожного компоненту без впливу на всю систему. Це драматично підвищує гнучкість розробки, дозволяючи різним командам працювати паралельно над окремими сервісами. Архітектура забезпечує кращу відмовостійкість, оскільки збій одного сервісу не призводить до краху всієї системи. Обидві архітектури, через свою розподілену природу та наявність множини взаємодіючих компонентів, вимагають впровадження спеціалізованих систем управління та оркестрації для забезпечення координації, моніторингу та ефективної роботи всієї інфраструктури.

Оркестрація контейнерних додатків представляє собою комплексний процес автоматичного керування життєвим циклом контейнеризованих застосунків у розподіленому середовищі. Цей процес охоплює автоматизоване розгортання, динамічне масштабування, балансування навантаження та координацію роботи контейнерних додатків на кластері серверів.

Впровадження оркестрації надає численні стратегічні переваги для сучасних IT-інфраструктур. Насамперед, вона забезпечує високу доступність сервісів через автоматичне відновлення після збоїв та розподіл навантаження. Система гарантує надійність і стабільну продуктивність застосунків навіть за умов інтенсивного використання. Безпека підвищується завдяки ізоляції контейнерів та централізованому управлінню доступом.

Оркестрація суттєво оптимізує використання обчислювальних ресурсів та інфраструктури, автоматично розподіляючи навантаження між доступними вузлами кластера. Це спрощує процеси управління та моніторингу великої кількості контейнерів, дозволяючи DevOps-командам ефективніше контролювати складні системи.

Управління та оркестрація є ключовими процесами в сучасних інформаційних системах. Управління відповідає за забезпечення стабільного функціонування окремих компонентів системи, включаючи постійний моніторинг стану ресурсів, гнучке налаштування параметрів, надійну безпеку даних, систематичне резервне копіювання інформації та контроль продуктивності. Оркестрація координує взаємодію між різними компонентами системи, забезпечуючи автоматизоване розгортання застосунків, динамічне масштабування ресурсів залежно від навантаження, інтелектуальне балансування трафіку, ефективну маршрутизацію запитів та узгоджену роботу всіх сервісів. Разом ці процеси створюють комплексний підхід до управління складними розподіленими системами, забезпечуючи їхню надійність, масштабованість та ефективність. Робочий процес оркестратора проілюстровано на рисунку 1.1.

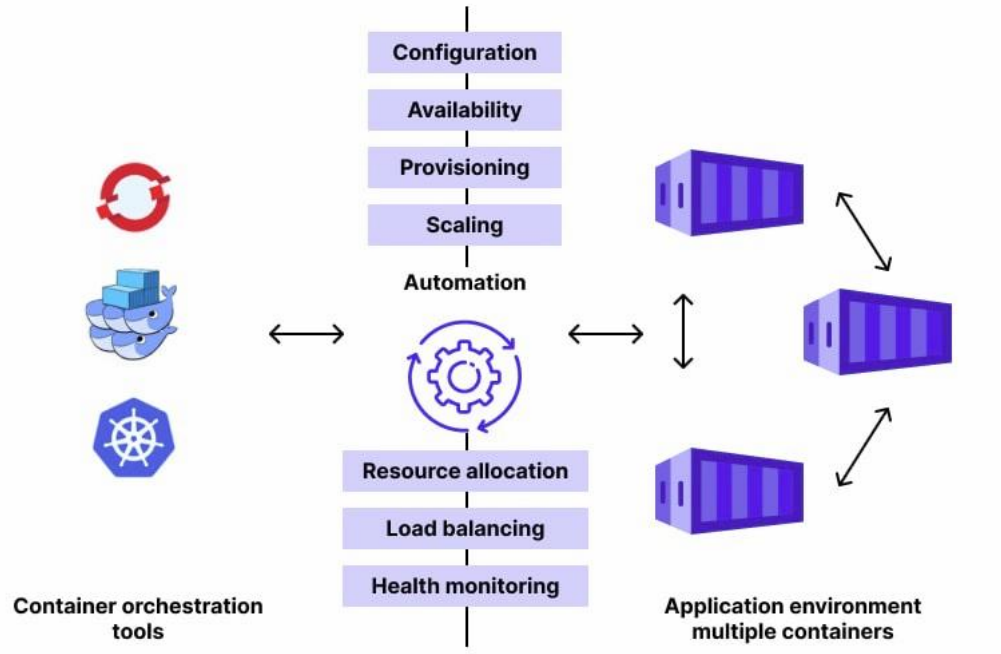


Рисунок 1.1 – Робочий процес оркестратора [5]

Автоматизація розгортання інфраструктури стала критично важливим елементом у сучасному управлінні IT-проектами та їх ефективному розвитку. Infrastructure as Code (IaC) характеризує інноваційний підхід до керування IT-

інфраструктурою, за якого вся конфігурація, параметри та налаштування визначаються через програмний код. Замість трудомістких ручних операцій адміністратори отримують можливість повністю автоматизувати процеси за допомогою спеціалізованих скриптів, написаних різними мовами програмування.

IaC революційно спрощує управління інфраструктурою та прискорює розгортання нових ресурсів. Замість багатогодинного ручного створення серверів, налаштування мереж та конфігурації баз даних, фахівці можуть описати всю необхідну інфраструктуру у вигляді коду. Це робить процес не лише швидшим та ефективнішим, але й повністю відтворюваним, що забезпечує консистентність середовищ та мінімізує людські помилки при розгортанні систем.

Особливо важливою є роль оркестрації у розвитку хмарних та мікросервісних архітектур. Вона забезпечує необхідну гнучкість, модульність та масштабованість, дозволяючи створювати сучасні розподілені застосунки [6].

Серед численних інструментів оркестрації найбільшої популярності набули Kubernetes та OpenShift, які пропонують потужні функціональні можливості для управління контейнерними екосистемами.

Kubernetes – це потужна відкрита система оркестрації контейнерних додатків, яка революціонізувала процес розгортання та управління сучасними застосунками. Платформа забезпечує автоматизоване керування контейнерами на розподілених кластерах, дозволяючи ефективно запускати, масштабувати та моніторити додатки. Kubernetes надає широкий спектр вбудованих сервісів: автоматичне балансування навантаження між екземплярами, забезпечення безперебійної комунікації між компонентами, самовідновлення при збоях, автомасштабування залежно від навантаження. Система підтримує декларативну конфігурацію, що спрощує управління інфраструктурою. Завдяки гнучкій архітектурі Kubernetes став стандартом для хмарних рішень та DevOps-практик, забезпечуючи високу доступність, надійність та ефективність додатків у

виробничому середовищі. На рисунку 1.2 представлено архітектуру та компоненти Kubernetes.

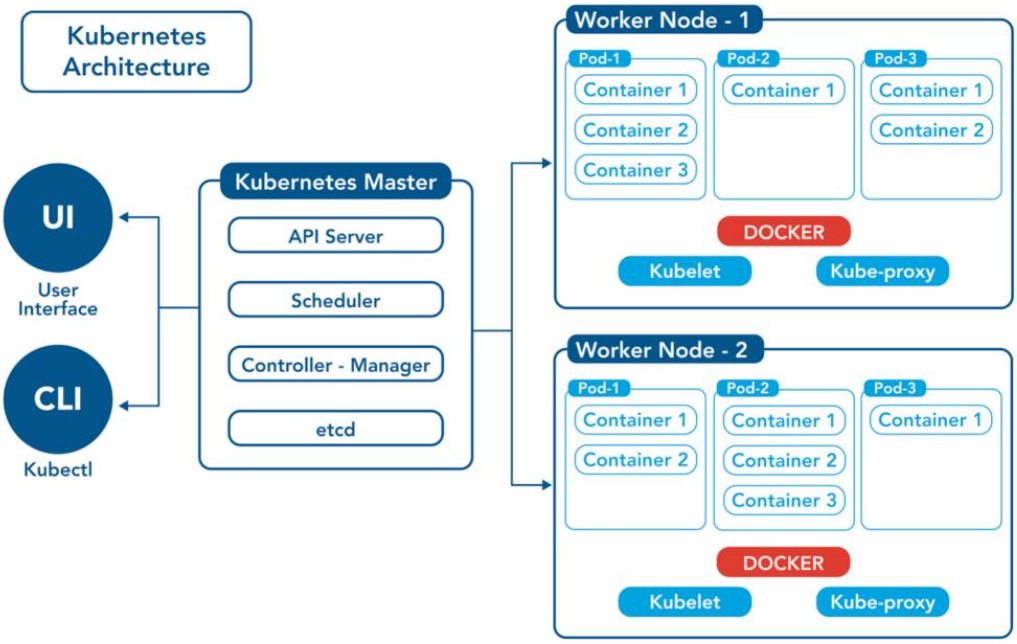


Рисунок 1.2 – Архітектура та компоненти Kubernetes [7]

OpenShift – це корпоративна платформа для розгортання та управління контейнерними додатками, розроблена компанією Red Hat на базі Kubernetes. Ця платформа значно розширює можливості стандартного Kubernetes, додаючи комплексні інструменти для безпеки, моніторингу та автоматизації процесів розробки.

OpenShift забезпечує вбудовані механізми захисту, включаючи контроль доступу та сканування вразливостей. Платформа пропонує потужні засоби моніторингу продуктивності додатків та інфраструктури, автоматизує процеси CI/CD, спрощує масштабування ресурсів. OpenShift seamlessly інтегрується з провідними хмарними провайдерами (AWS, Azure, Google Cloud) та корпоративними системами, що робить її універсальним рішенням для гібридних та мультихмарних середовищ, забезпечуючи гнучкість та надійність enterprise-рівня.

Kubernetes представляє собою відкриту стандартизовану платформу для оркестрації контейнерних застосунків, що забезпечує комплексне управління розподіленими системами. Ця технологія пропонує широкий спектр функціональних можливостей для автоматизованого запуску, динамічного масштабування та ефективного управління контейнерами в межах кластера серверів. Особливою перевагою Kubernetes є його потужна екосистема – велика міжнародна спільнота розробників та користувачів активно сприяє розвитку платформи, надаючи технічну підтримку, консультації, навчальні матеріали та інноваційні рішення.

OpenShift являє собою комерційну корпоративну платформу, побудовану на базі Kubernetes, що розширює його базову функціональність додатковими enterprise-можливостями. Платформа інтегрує вдосконалені механізми безпеки, всебічний моніторинг інфраструктури, автоматизовані процеси розгортання та seamless інтеграцію з провідними хмарними сервісами. OpenShift відзначається інтуїтивно зрозумілим інтерфейсом користувача, що суттєво спрощує роботу з контейнерами, забезпечує гнучкі опції розгортання в різних середовищах та надає можливості для розширення функціоналу відповідно до специфічних бізнес-вимог організації. Обидві платформи мають свої унікальні переваги залежно від конкретних потреб проєкту.

Kubernetes та OpenShift представляють собою два провідних рішення для оркестрації контейнерних додатків у сучасному IT-середовищі. Кожна з цих платформ володіє унікальними характеристиками, перевагами та обмеженнями, які необхідно враховувати при виборі оптимального інструменту.

Kubernetes виступає як відкрита платформа з величезною спільнотою розробників, що забезпечує гнучкість, масштабованість та широку підтримку різноманітних інтеграцій. OpenShift, натомість, надає корпоративне рішення з додатковими функціями безпеки та зручним інтерфейсом.

Вибір між цими технологіями визначається специфічними вимогами проєкту, технічною експертизою команди, бюджетними обмеженнями та довгостроковими бізнес-цілями організації. Враховуючи ширшу популярність,

велику екосистему та гнучкість налаштувань, для даного проєкту було прийнято рішення використовувати саме Kubernetes як основний інструмент оркестрації контейнерних застосунків.

Сучасні підходи до розробки та розгортання програмних систем дедалі більше спираються на контейнеризацію – технологію, що дозволяє ізолювати застосунки та їхні залежності у відокремлених середовищах. Проте із зростанням кількості контейнерів виникла потреба в ефективному управлінні ними, балансуванні навантаження, автоматичному масштабуванні та моніторингу. Для цього використовуються системи оркестрації контейнерів, серед яких Kubernetes став стандартом де-факто. Водночас для навчальних, тестових або невеликих середовищ класичний Kubernetes є занадто ресурсомістким, тому з’явився ряд легких платформ оркестрації: Minikube, MicroK8s, K3s, K0s і Kind.

Minikube це одна з найпопулярніших платформ для локального розгортання Kubernetes-кластерів. Її основна мета полягає в тому, щоб надати розробникам швидкий і зручний спосіб експериментувати з Kubernetes без необхідності створювати складну інфраструктуру. Minikube працює у віртуальній машині або контейнері, забезпечуючи повноцінне середовище з одним вузлом (single-node). Завдяки простим командам CLI, таким як minikube start чи minikube dashboard, користувачі можуть легко розгортати застосунки, тестувати YAML-маніфести та тренуватися у DevOps-практиках.

MicroK8s, розроблений Canonical, позиціонується як легкий, модульний дистрибутив Kubernetes, який можна запустити буквально однією командою (snap install microk8s --classic). Він підтримує додаткові модулі, наприклад Istio, Prometheus чи Helm, які можна активувати за допомогою простих команд microk8s enable. MicroK8s чудово підходить для edge-комп’ютингу, IoT-рішень і невеликих серверів, оскільки має мінімальні вимоги до ресурсів. Його зручно використовувати як у локальному середовищі розробника, так і на невеликих хмарних інстансах.

K3s, створений компанією Rancher Labs, є ще однією надлегкою реалізацією Kubernetes, оптимізованою для роботи в умовах обмежених ресурсів.

Він видаляє непотрібні компоненти, об’єднує ключові служби в один виконуваний файл і підтримує SQLite як вбудовану базу даних для зберігання стану. K3s ідеально підходить для IoT, edge-пристроїв і малих кластерів, але водночас зберігає повну сумісність з основним Kubernetes API, що дозволяє використовувати звичні інструменти kubectl, Helm чи kubectl apply.

K0s, розроблений компанією Mirantis, позиціонується як «zero friction» Kubernetes-дистрибутив. Його можна запустити без root-доступу, що робить його безпечнішим і зручнішим у середовищах з підвищеними вимогами до безпеки. K0s відзначається простою інсталяцією, мінімальними залежностями та автоматичною конфігурацією компонентів кластера, що дозволяє швидко розгорнути повноцінну Kubernetes-інфраструктуру навіть на віртуальних машинах або невеликих серверах.

Kind (Kubernetes in Docker) – це інструмент, який дозволяє запускати Kubernetes-кластери безпосередньо в Docker-контейнерах. Такий підхід робить його надзвичайно зручним для CI/CD-процесів, оскільки кластери можна створювати та знищувати динамічно під час автоматичних тестів. Kind широко використовується для розробки й тестування Kubernetes-компонентів або інтеграції застосунків у контрольованому середовищі.

Підсумовуючи, легкі системи оркестрації контейнерів відіграють ключову роль у навчанні, розробці та тестуванні сучасних розподілених застосунків. Вони забезпечують швидкий доступ до функціональності Kubernetes без складних налаштувань і високих вимог до ресурсів. Кожна з них має свою нішу – від локального навчання (Minikube, Kind) до edge-комп’ютингу (MicroK8s, K3s, K0s) – але всі вони спрямовані на спрощення переходу до повноцінних DevOps- та хмарних рішень.

Вибір між цими платформами залежить від конкретних потреб: Minikube для навчання, MicroK8s для IoT, K3s/K0s для edge-computing, Kind для CI/CD. Усі вони демократизують доступ до Kubernetes, дозволяючи розробникам експериментувати та тестувати без потреби в хмарній інфраструктурі, що прискорює цикл розробки та зменшує витрати.

1.2 Огляд і аналіз методів та засобів безпеки оркестрації контейнерних платформ для вирішення проблеми дослідження

Сучасний технологічний ландшафт характеризується безпрецедентним рівнем складності інформаційних систем та постійною еволюцією кіберзагроз, які проникають у найглибші шари інфраструктури. У контексті контейнерних технологій це означає, що комплексний підхід до безпеки має охоплювати абсолютно всі рівні технологічного стеку – від фундаментального апаратного забезпечення та мікропрограмного рівня до програмних компонентів найвищого рівня абстракції. Саме така всеохоплююча стратегія формує основу для розподіленої моделі довірених обчислень, яка гарантує максимально захищене середовище для створення, розгортання, оркестрації та управління контейнеризованими застосунками.

Фундаментом надійної обчислювальної моделі слугує механізм вимірюваного та захищеного завантаження системи, що забезпечує верифікацію цілісності системної платформи з першого моменту ініціалізації. Цей процес створює безперервний ланцюг довіри, який бере початок на рівні апаратних компонентів і послідовно поширюється через завантажувачі операційної системи, ядро та системні компоненти. Така архітектура уможливлює криптографічну верифікацію усіх критичних елементів: механізмів завантаження, системних образів, середовищ виконання контейнерів та власне контейнерних образів. У практичному застосуванні до контейнерних технологій ці безпекові методології інтегруються на рівні апаратної платформи, гіпервізора та хостової операційної системи, забезпечуючи надійний захист специфічних контейнерних компонентів.

Управління ризиками представляє собою комплексний підхід, що ґрунтується на систематичній ідентифікації потенційних загроз, їх структуруванні та класифікації за рівнем критичності, а також виборі оптимальних стратегій для мінімізації можливих негативних наслідків. Цей

процес вимагає постійного моніторингу та аналізу всіх можливих ризиків, що можуть вплинути на безпеку системи.

Моделювання загроз являє собою структурований процес виявлення, документування та аналізу всіх потенційних загроз, які можуть вплинути на інформаційну систему. Цей методичний підхід передбачає ретельне дослідження всіх компонентів системи, аналіз можливих сценаріїв атак та визначення потенційних векторів проникнення. Завдяки такому всебічному аналізу організації можуть проактивно ідентифікувати найбільш вразливі елементи своєї інфраструктури та зосередити ресурси на їх захисті.

Важливо розуміти, що універсальної моделі загроз, яка б підходила абсолютно всім, просто не існує. Кожна організація має унікальні особливості свого технологічного середовища, специфічні бізнес-процеси та індивідуальний набір додатків. Тому модель загроз повинна розроблятися з урахуванням конкретних умов функціонування. Водночас, можна виокремити певні типові загрози, характерні для багатьох контейнерних середовищ, що дозволяє створити базову основу для аналізу безпеки.

Протягом дослідницького періоду, що тривав з жовтня 2020 року до лютого 2021 року, команда фахівців з кібербезпеки Unit 42 [8] проводила систематичне сканування та детальний аналіз незахищених кластерів Kubernetes (скорочено k8s), що знаходилися у відкритому доступі в мережі Інтернет. Хоча платформа Kubernetes передбачає можливість та необхідність налаштування численних параметрів безпеки для захисту інфраструктури, дослідники виявили значну кількість кластерів, які не мали базових засобів захисту. Це створювало критичну вразливість, оскільки будь-який зловмисник, володіючи інформацією про IP-адреси, номери портів та точки доступу до API, міг отримати повний анонімний доступ до цих систем.

Результати дослідження виявилися вражаючими: було ідентифіковано 2100 незахищених кластерів Kubernetes, які об’єднували 5300 обчислювальних вузлів, 31 340 процесорних ядер та 75 270 активних модулів. Ці вразливі системи використовувалися організаціями з критично важливих секторів економіки,

включаючи електронну комерцію, фінансові установи та заклади охорони здоров'я.

Особливу привабливість для потенційних зловмисників становили масштабні обчислювальні потужності та величезні обсяги конфіденційної інформації, що зберігалася в цих застосунках: токени доступу до API, облікові дані систем баз даних, вихідні коди програмного забезпечення та персональні дані користувачів (PII). Найпотужніший виявлений кластер налічував понад 500 вузлів та 2000 активних капсул. Рисунок 1.3 ілюструє порівняльний аналіз виявлених ресурсів Kubernetes.

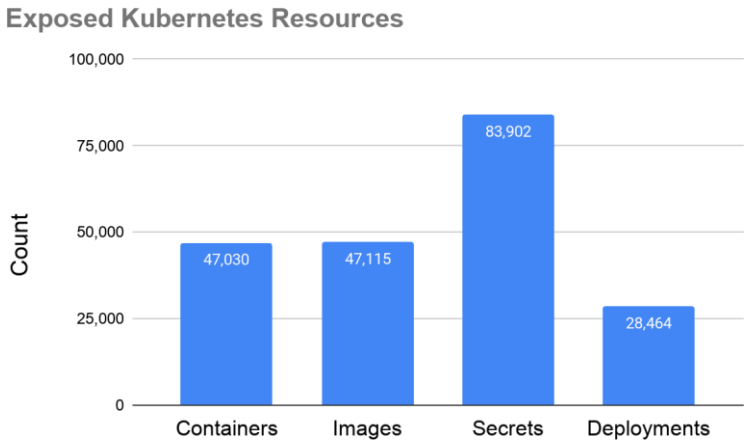


Рисунок 1.3 – Порівняльний аналіз виявлених ресурсів Kubernetes [9]

Kubernetes, як складна система оркестрації контейнерів, характеризується численними викликами в сфері інформаційної безпеки, з якими регулярно стикаються фахівці з адміністрування систем та DevOps-інженери. Розглянемо детальніше найбільш критичні вразливості та векторі атак.

Лавиноподібне зростання внутрішнього трафіку (Explosion of East-West Traffic). Ця проблема виникає через специфіку розподіленої архітектури Kubernetes. Контейнери можуть бути динамічно розгорнуті в різних географічно віддалених хмарних середовищах, що призводить до експоненційного збільшення обсягу міжсервісної комунікації всередині логічного кластера. Така розподілена природа створює сприятливі умови для зловмисників, які можуть

використовувати множинні точки входу для організації масштабних DDoS-атак. Крім того, ускладнюється моніторинг та виявлення аномалій у трафіку, оскільки необхідно аналізувати комунікації між численними вузлами в різних дата-центрах.

Розширена поверхня атаки (Increased Attack Surface). Гетерогенність контейнерного середовища створює унікальні виклики безпеки. Кожен окремий контейнер у кластері може містити власний набір вразливостей, специфічних для конкретних версій програмного забезпечення, бібліотек або конфігурацій. Зловмисники активно експлуатують відомі проблеми безпеки в популярних технологіях, таких як Docker runtime, системи авторизації хмарних провайдерів (AWS IAM, Azure AD), або вразливості в базових образах операційних систем. Множинність компонентів значно ускладнює централізоване управління безпекою та своєчасне застосування патчів.

Компрометація контейнерів (Container Compromise). Найпоширеніший вектор атак на Kubernetes-інфраструктуру ґрунтується на використанні помилок у конфігурації безпеки. Неправильні налаштування можуть включати надмірні привілеї контейнерів, відсутність обмежень ресурсів, або застосування застарілих образів з відомими вразливостями. Успішна компрометація надає зловмисникам можливість маніпулювати внутрішніми процесами, отримувати доступ до файлової системи хосту, перехоплювати мережевий трафік або ескалювати привілеї для компрометації всього кластера.

Неавторизовані з’єднання між подами (Unauthorized Pod-to-Pod Connections). Скомпрометовані контейнери можуть встановлювати латеральні з’єднання з іншими подами в межах кластера, створюючи ланцюги атак. Хоча традиційна фільтрація на мережевому рівні (L3 ACL) забезпечує базовий захист, вона виявляється недостатньою для виявлення складних атак на рівні застосунків. Ефективний захист вимагає глибокої інспекції пакетів на рівні L7 моделі OSI, що дозволяє аналізувати контент взаємодії та виявляти аномальну поведінку застосунків.

Захист Kubernetes-кластерів можна реалізувати через два основні методологічні напрямки. Перша стратегія полягає у ретельному конфігуруванні параметрів захисту та впровадженні перевірених галузевих стандартів безпеки на всіх критичних компонентах Kubernetes-архітектури, включаючи вузли, контейнери та мережеві з'єднання. Альтернативний підхід передбачає інтеграцію спеціалізованих зовнішніх інструментів, які забезпечують безперервний моніторинг загроз, автоматизований контроль політик безпеки та централізоване управління рівнем захищеності середовища. Далі представлено універсальні практичні поради щодо налаштування механізмів безпеки, які можуть бути успішно адаптовані та застосовані в різноманітних Kubernetes-застосунках незалежно від їхньої специфіки чи масштабу розгортання.

До ключових принципів безпеки інфраструктури Kubernetes відносять:

– впровадження криптографічного захисту на основі TLS. Протокол Transport Layer Security необхідно активувати для всіх компонентів інфраструктури Kubernetes, які технічно підтримують цю технологію. Така конфігурація забезпечує комплексний захист від несанкціонованого перехоплення мережевого трафіку, обов'язкову верифікацію ідентичності серверної сторони та, у випадку використання взаємного TLS (Mutual TLS), додаткову аутентифікацію клієнтських запитів;

– застосування RBAC-моделі з принципом мінімальних привілеїв. Система контролю доступу на основі ролей (Role-Based Access Control) дозволяє створювати гнучкі політики безпеки, через які користувачі отримують строго обмежений доступ до конкретних ресурсів кластера, зокрема до просторів імен. За замовчуванням слід надавати лише мінімально необхідні права для виконання робочих завдань;

– інтеграція зовнішніх систем аутентифікації для API Server. Для ефективного управління правами доступу співробітників організації рекомендується централізована система автентифікації та авторизації на корпоративному рівні, зокрема через LDAP-каталоги або технологію єдиного входу (Single Sign-On), що спрощує процеси надання та відкликання привілеїв;

– захист кластеру etcd через мережеву ізоляцію. Кластер etcd є однією з найважливіших і найбільш критичних складових архітектури Kubernetes, оскільки він виконує функцію розподіленого сховища конфігураційних даних. У цьому кластері зберігається вся інформація про поточний стан системи Kubernetes, включаючи метадані про поди, сервіси, розгортання, а також найбільш чутливі дані безпеки – токени автентифікації, криптографічні сертифікати та інші секрети доступу. Враховуючи критичність цієї інформації, кластер etcd потребує багаторівневого захисту. Найефективнішим підходом є його розташування за потужним мережевим екраном (Firewall), який має бути налаштований на дозвіл лише необхідного трафіку від авторизованих компонентів Kubernetes. Більш того, рекомендується розміщувати etcd в ізольованій віртуальній приватній хмарі (VPC), що створює додатковий рівень мережевої сегментації та унеможливлює несанкціонований доступ з інших мережевих сегментів інфраструктури;

– стратегія ротації криптографічних ключів. Систематична та регулярна зміна ключів шифрування є фундаментальною практикою забезпечення інформаційної безпеки в будь-якій сучасній IT-системі. Процес ротації ключів має здійснюватися за двома принципами: часовим (за заздалегідь визначеним розкладом) та подієвим (у відповідь на певні події безпеки). Така практика дозволяє суттєво мінімізувати потенційні наслідки компрометації ключів доступу, обмежуючи так званий «радіус ураження». Якщо зловмисник отримає доступ до ключа, його використання буде обмежене часовим вікном до наступної ротації, що значно зменшує можливості для несанкціонованого доступу до системи та її ресурсів;

– безпека конфігураційних файлів YAML. Конфігураційні файли у форматі YAML відіграють ключову роль у декларативному управлінні ресурсами Kubernetes. Критично важливо забезпечити, щоб конфіденційна інформація, що міститься в цих файлах, ніколи не зберігалася в незашифрованому вигляді на подах або в системах контролю версій. Всі чутливі конфігурації, секрети доступу, паролі та інші критичні дані мають бути

зашифровані за допомогою спеціалізованих інструментів, таких як git-crypt, sealed-secrets або зовнішніх систем управління секретами. Регулярний статичний аналіз YAML-конфігурацій дозволяє встановити та підтримувати базові стандарти безпеки, виявляти потенційні вразливості до їх експлуатації. Статистика показує, що саме неправильна конфігурація є причиною більшості успішних атак на інфраструктуру Kubernetes, тому автоматизований аналіз конфігурацій має стати невід'ємною частиною CI/CD-конвеєра;

– принцип мінімальних привілеїв для контейнерів. Запуск контейнерів під обліковим записом root є поширеною, але небезпечною практикою. Контейнери, що працюють з привілеями суперкористувача, зазвичай володіють значно більшими правами доступу, ніж фактично потрібно для виконання їхніх робочих завдань. Така надлишкова привілейованість створює серйозні ризики безпеки: у разі компрометації контейнера зловмисник отримує розширені можливості для подальшого проникнення в систему, латерального переміщення мережею та ескалації привілеїв.

Хоча дотримання всіх попередньо описаних рекомендацій і практик дозволяє забезпечити фундаментальні потреби у сфері інформаційної безпеки, варто зауважити, що процес ручного налаштування параметрів захисту застосунку або проведення комплексного аудиту вже розгорнутого продукту вимагає значних витрат робочого часу спеціалізованого інженера з безпеки та потребує глибоких технічних знань.

Відповідно до результатів дослідження [10], наведених у згаданому звіті, більше ніж 86% сучасних організацій активно використовують платформу Kubernetes для оркестрації та управління хоча б частиною своїх контейнеризованих робочих навантажень. Однак рівень забезпечення безпеки в цих середовищах часто залишається недостатнім і не відповідає сучасним вимогам. Критичною проблемою є те, що більше половини цих компаній не здійснили належних інвестицій у захист контейнерної інфраструктури, що призвело до негативних наслідків. Це або суттєво уповільнило темпи впровадження Kubernetes у виробниче середовище, або спричинило серйозні

інциденти, пов’язані з порушенням безпеки. Особливо тривожним є той факт, що у багатьох організацій відсутня чітка стратегія для ефективного вирішення таких інцидентів та запобігання їм у майбутньому.

У контексті обмежених часових ресурсів та необхідності оптимізації процесів розробки критично важливим стає вибір адекватної стратегії перевірки застосунків, яка відповідатиме специфічним вимогам проєкту. Для визначення найбільш ефективного підходу до ідентифікації вразливостей контейнеризованих застосунків необхідно провести ґрунтовне дослідження існуючих методик виявлення вразливостей, порівняти їхні можливості та обмеження. Наступним кроком має стати формування об’єктивних кількісних характеристик ефективності кожної методики, що дозволить здійснити обґрунтований вибір оптимального рішення для конкретного середовища розгортання.

1.3 Постановка завдання на кваліфікаційну роботу магістра

Сучасні тенденції у сфері DevOps і хмарних обчислень дедалі частіше базуються на контейнеризації, яка дозволяє швидко розгортати, масштабувати й оновлювати застосунки незалежно від середовища. Легкі платформи оркестрації контейнерів, такі як minikube, microk8s, K3s, K0s та kind, відіграють важливу роль у навчальних, тестових і виробничих середовищах, забезпечуючи мінімальну складність та низькі вимоги до ресурсів. Однак разом із зручністю та мобільністю зростають ризики, пов’язані з безпекою даних, управлінням доступом, а також збереженням цілісності системи. Саме тому постає необхідність розроблення системи безпеки, здатної ефективно захищати такі легкі платформи оркестрації контейнерів.

Разом із тим спрощення архітектури таких платформ часто призводить до зниження рівня безпеки. Неконтрольований доступ до компонентів Kubernetes, відсутність централізованої системи журналювання подій або моніторингу безпеки, а також вразливості у контейнерах створюють значні ризики витоку

даних, компрометації вузлів чи контейнерів. Тому дослідження, спрямоване на розроблення системи безпеки для оркестрації легких контейнерних платформ, є надзвичайно актуальним і практично значущим.

Проблематика безпеки в контейнерних середовищах є багаторівневою. З одного боку, контейнери надають певну ізоляцію процесів, але ця ізоляція не є повною. Вразливості ядра, некоректні налаштування мережевих політик, відсутність контрольованого доступу до секретів або невідповідне керування правами користувачів можуть призвести до серйозних порушень безпеки. З іншого боку, легкі оркестратори на кшталт minikube чи K3s часто застосовуються у середовищах розробки, де увага до безпеки знижується, що створює додаткові ризики при подальшій інтеграції таких рішень у промислові кластери.

Метою даної роботи є розроблення та дослідження системи безпеки, орієнтованої на забезпечення комплексного контролю доступу, моніторингу подій безпеки та інтеграції із системами журналювання для легких платформ оркестрації контейнерів. Система повинна враховувати особливості архітектури таких рішень, як minikube, microk8s, K3s, K0s та kind, зокрема спрощені моделі кластеризації, використання вбудованих компонентів Kubernetes (kubectl, kubelet, API-server), а також обмеження щодо ресурсів і масштабованості.

Для досягнення поставленої мети необхідно вирішити такі основні завдання:

– проаналізувати існуючі підходи до захисту контейнерних середовищ Kubernetes-подібного типу;

– визначити основні загрози, що виникають у легких оркестраторах, включно з атаками на API-сервер, несанкціонованим доступом до pods, компрометацією секретів і мережевими атаками між контейнерами;

– розробити модель безпеки, яка забезпечуватиме автентифікацію, авторизацію, аудит і шифрування на рівні вузлів і контейнерів;

– провести експериментальне впровадження системи безпеки на прикладі однієї або кількох платформ (наприклад, minikube і K3s), оцінити її ефективність та продуктивність.

Таким чином, дослідження сприятиме розвитку підходів до безпечної оркестрації контейнерів у середовищах із обмеженими ресурсами, забезпечуючи баланс між продуктивністю, масштабованістю та захистом інформації. Це дозволить зробити технології контейнеризації більш надійними та готовими до використання у широкому спектрі завдань від розробки до розгортання критично важливих застосунків.

Висновки до розділу 1

У межах цього розділу здійснено комплексний аналіз ключових аспектів інформаційної безпеки оркестрації легких контейнерних платформ Kubernetes. Дослідження дозволило ідентифікувати критичні вектори атак, класифікувати групи потенційних загроз та сформувати детальні моделі загроз для цих технологій.

Окрему увагу приділено експериментальному дослідженню, проведеному за участю фахівців з кібербезпеки, яке мало на меті визначити оптимальний підхід до сканування вразливостей. Результати експерименту переконливо продемонстрували, що гібридний метод, який поєднує автоматизоване сканування з мануальною верифікацією, забезпечує найвищу ефективність виявлення загроз при мінімальних часових витратах.

На основі отриманих висновків прийнято рішення розробити спеціалізоване програмне забезпечення, яке максимально автоматизує процеси аудиту безпеки, зменшуючи навантаження на інженерів та мінімізуючи ризики, пов’язані з людським фактором. Система здійснюватиме пасивний моніторинг та аудит безпеки досліджуваних технологій у стислі терміни, після чого виявлені вразливості проходитимуть обов’язкову мануальну валідацію експертами.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ МОДУЛЮ БЕЗПЕКИ ІНФОРМАЦІЇ

2.1 Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання

У сучасних умовах розвитку DevOps і хмарних обчислень безпека контейнерних платформ стає одним із ключових чинників стабільної роботи інформаційних систем. Легкі платформи для оркестрації контейнерів – такі як minikube, microk8s, K3s, kind – забезпечують спрощене розгортання кластерів Kubernetes, однак разом із цим відкривають нові вектори атак через недосконалі налаштування, відсутність політик контролю доступу або вразливі образи контейнерів. Тому дослідження системи безпеки цих платформ потребує комплексного підходу з використанням перевірених технологій аналізу й аудиту. Для дослідження системи безпеки легких контейнерних платформ доцільно застосувати комбінований підхід, що поєднує теоретичний аналіз, емпіричне тестування та моделювання загроз. Такий багатовекторний метод дозволяє охопити всі рівні безпеки контейнерної екосистеми: від ізоляції процесів на рівні ядра операційної системи до управління доступом на рівні оркестрації [11].

Для всебічного аналізу безпеки контейнерних платформ необхідний багатовимірний підхід, що охоплює різні аспекти захисту: від відповідності стандартам до активного пошуку вразливостей. Це зумовлює вибір комплексу інструментів, кожен з яких спеціалізується на окремому аспекті безпеки. Обрання саме комплексного підходу зумовлене складністю архітектури контейнерних платформ, де безпека залежить від взаємодії численних компонентів: container runtime, kernel namespaces, cgroups, networking stack, image registries та систем оркестрації. Дослідження лише одного аспекту не дасть повної картини реального стану безпеки системи.

Для реалізації цього дослідження обрано інструментарій Kube-bench, Kubesec, Kubeaudit, Kube-hunter та Kube-scan. Кожен із них доповнює інші, формуючи цілісну систему оцінки безпеки контейнерного середовища.

Kube-bench використовується як базовий засіб перевірки відповідності конфігурації Kubernetes рекомендаціям CIS Benchmark. Цей інструмент дозволяє визначити відхилення від безпечних налаштувань у компонентах кластера (API Server, Kubelet, Controller Manager тощо). Його алгоритм базується на тестових сценаріях, які перевіряють ключові параметри конфігураційних файлів, прав доступу та використання TLS-сертифікатів. Використання Kube-bench дає змогу побудувати основу для подальшого вдосконалення політик безпеки [12].

Kubesec забезпечує аналіз маніфестів Kubernetes на рівні YAML-файлів. Його алгоритм виконує статичну перевірку політик безпеки, виявляючи ризикові налаштування, як-от відсутність readOnlyRootFilesystem, використання привілейованих контейнерів чи відсутність мережевих політик. Це дозволяє забезпечити превентивний контроль на етапі розгортання [13].

Kubeaudit орієнтований на динамічний аудит конфігурацій кластера. Він автоматично сканує ресурси Kubernetes і виявляє порушення принципів безпеки, таких як неправильне керування доступом, невірно налаштовані service accounts або невідповідні правила RBAC. Завдяки автоматизації звітності kubeaudit спрощує моніторинг безпеки у процесі життєвого циклу контейнерних сервісів [14].

Kube-hunter виконує активне тестування безпеки, моделюючи дії потенційного зловмисника. Алгоритми інструменту шукають відкриті порти, неконтрольований доступ до API, слабкі автентифікаційні механізми та інші експлойтабельні вразливості. Таким чином, kube-hunter дозволяє перевірити ефективність захисних механізмів у реальних умовах [15].

Нарешті, Kube-scan забезпечує інтегровану оцінку ризиків для всіх компонентів кластера, надаючи візуалізований звіт про рівень загроз та рекомендації щодо їх усунення. Завдяки алгоритмам кореляції результатів

сканування kube-scan допомагає пріоритезувати виправлення найбільш критичних проблем [16].

Комбінація цих п’яти інструментів створює багаторівневу систему аналізу безпеки, що охоплює весь життєвий цикл контейнерного додатку:

– етап розробки та проектування забезпечується Kubesec, який інтегрується в CI/CD pipeline та запобігає потраплянню небезпечних конфігурацій у продакшн;

– етап розгортання та конфігурації контролюється Kube-bench, що перевіряє відповідність базових налаштувань кластера індустріальним стандартам;

– етап експлуатації моніториться Kubeaudit та Kube-scan, які виявляють відхилення від безпечних практик у реальному часі;

– етап тестування захисту реалізується Kube-hunter, який активно перевіряє ефективність впроваджених механізмів безпеки.

Важливо визнати обмеження обраного підходу. Kube-bench перевіряє лише відповідність стандартам, але не виявляє логічні вразливості в архітектурі додатків. Статичні аналізатори (Kubesec, Kubeaudit) не можуть виявити runtime-вразливості, такі як експлуатація вразливостей у самих додатках. Kube-hunter в активному режимі може потенційно порушити роботу кластера, тому його використання в продакшн-середовищі вимагає обережності. Kube-scan вимагає встановлення додаткових компонентів у кластер, що збільшує площу атаки. Ці обмеження мінімізуються комплексним використанням всіх інструментів та доповненням їх ручним аналізом критичних компонентів.

Легкі контейнерні платформи мають свої особливості, які впливають на методологію тестування безпеки:

– обмеження ресурсів вимагають оптимізації процесу сканування. Інструменти повинні використовуватися послідовно, а не одночасно, щоб не перевантажувати систему;

– спрощена архітектура деяких легких платформ може означати відсутність певних компонентів (наприклад, etcd може бути замінено на SQLite в K3s). Це потребує адаптації тестових сценаріїв;

– edge-computing контекст передбачає потенційно менш захищене фізичне розміщення, тому особливу увагу слід приділити захисту API-сервера та автентифікації.

Застосування зазначених технологій дозволяє комплексно дослідити безпеку легких контейнерних платформ з різних аспектів – від статичного аналізу конфігурацій до активного тестування та візуалізації ризиків. Їхня комбінація забезпечує системність і достовірність результатів, що є необхідною умовою для формування практичних рекомендацій з підвищення безпеки контейнерних середовищ.

Таким чином, вибір цих інструментів обґрунтований їхньою взаємодоповнюваністю, автоматизованими алгоритмами аналізу, відкритістю вихідного коду та сумісністю з легкими платформами типу minikube, microk8s чи K3s. Сукупне використання kube-bench, kubesec, kubeaudit, kube-hunter і kube-scan формує ефективний методологічний підхід до комплексного дослідження системи безпеки контейнерної інфраструктури.

Обраний комплексний підхід до дослідження системи безпеки легких контейнерних платформ забезпечує всебічний аналіз через поєднання теоретичних моделей загроз, практичного тестування з використанням індустріальних інструментів та автоматизованого моніторингу. Вибір конкретних технологій базується на їхній поширеності в індустрії, відкритості архітектури для дослідження та повноті покриття різних аспектів безпеки.

Такий методологічний підхід дозволить не лише виявити існуючі вразливості, але й розробити практичні рекомендації щодо побудови безпечної контейнерної інфраструктури, що є критично важливим для сучасних хмарних та мікросервісних архітектур.

2.2 Практична реалізація об’єкта проектування

При створенні програмного застосунку важливо визначити платформу, на якій здійснюватиметься сканування коду, а також обрати відповідні мови програмування та технологічний стек. Вибір має базуватися на функціональних вимогах проєкту, технічних можливостях команди розробників та специфічних потребах майбутньої системи для забезпечення оптимальної продуктивності.

Центр інтернет-безпеки (Center for Internet Security, CIS) функціонує як авторитетна некомерційна організація міжнародного рівня, основною місією якої є створення та впровадження комплексних контрольних показників, стандартів та практичних рекомендацій у сфері кібербезпеки. Організація надає детальні методологічні вказівки, які дозволяють компаніям та установам різного масштабу суттєво вдосконалювати власні системи захисту інформації, оптимізувати застосування спеціалізованих програмних рішень та здійснювати налаштування безпечних конфігурацій для корпоративного програмного забезпечення.

Стратегічна ініціатива CIS зосереджена на формуванні універсальних базових параметрів безпечної конфігурації для інформаційних систем, які широко використовуються в організаціях будь-якого профілю. Розробка відбувається з активним залученням провідних експертів у галузі кібербезпеки та інформаційних технологій, що представляють урядові структури, приватний бізнес-сектор та академічні установи з різних країн світу. Для формування стандартів, контрольних показників CIS та посилених захищених образів систем застосовується консенсусна модель колективного прийняття рішень, що забезпечує максимальну об’єктивність та практичну застосовність розроблених рекомендацій.

Розроблене програмне забезпечення в рамках магістерської роботи базуватиметься на принципах, методиках та практичних рекомендаціях, викладених у документі CIS Kubernetes Benchmark версії 1.5.1. Цей вибір обґрунтовується тим, що CIS Benchmark є визнаним міжнародним стандартом

безпеки, на який орієнтуються DevSecOps-інженери під час проектування та підтримки безпечної інфраструктури Kubernetes по всьому світу.

CIS Benchmark представляє собою комплексний набір базових конфігураційних параметрів та перевірених практик для забезпечення безпечного налаштування інформаційних систем. Кожна рекомендація в цьому стандарті безпосередньо пов’язана з одним або декількома контрольними механізмами CIS, які спеціально створені для того, щоб надати організаціям можливість суттєво підвищити рівень їхнього кіберзахисту та зміцнити загальну інформаційну безпеку. Важливою особливістю елементів керування CIS є їхня повна відповідність численним міжнародним стандартам та нормативним вимогам, серед яких: NIST Cybersecurity Framework (CSF), NIST SP 800-53, комплексна серія стандартів ISO 27000, PCI DSS для безпеки платіжних карток, HIPAA для охорони здоров’я та багато інших галузевих регуляторних документів.

Враховуючи той факт, що зазначений гайдлайн охоплює понад двісті п’ятдесят сторінок деталізованих технічних інструкцій та містить сотні конкретних рекомендацій щодо налаштування безпеки кластерів, їх ручна імплементація стає надзвичайно трудомістким та схильним до людських помилок процесом. Кожна неточність у конфігурації може призвести до серйозних вразливостей безпеки системи.

Саме з метою мінімізації ризиків та підвищення ефективності роботи фахівців було створено спеціалізований застосунок, який автоматизує процес перевірки конфігураційних файлів та контейнерних образів, забезпечуючи при цьому мінімальне втручання інженера та максимальну точність виконання перевірок відповідно до стандартів CIS.

Оскільки безпекові дослідники стикаються з обмеженнями щодо сканування та тестування застосунків, які не є їхньою власною розробкою або для перевірки яких відсутній офіційний дозвіл від власників, виникла потреба у створенні спеціалізованого навчального середовища. Саме тому фахівці з безпеки, які спеціалізуються на вивченні вразливостей екосистеми Kubernetes,

розробили унікальний тренувальний полігон під назвою Kubernetes-Goat. Ця платформа являє собою комплексне навчальне середовище, яке дозволяє в контрольованих умовах відтворити, проаналізувати та відпрацювати захист від усіх найбільш поширених векторів атак на Kubernetes-кластери. Завдяки цьому інструменту спеціалісти можуть безпечно вдосконалювати свої практичні навички без ризику порушення законодавства чи завдання шкоди реальним системам.

Ініціалізація програмного застосунку відбувається надзвичайно просто та виконується шляхом введення однієї службової команди в терміналі. Що відображено на рисунку 2.1. Однак важливо наголосити, що для забезпечення коректного та стабільного функціонування всієї системи необхідно попередньо активувати minikube, оскільки він створює необхідне середовище для розгортання та виконання програми в локальному кластері Kubernetes.

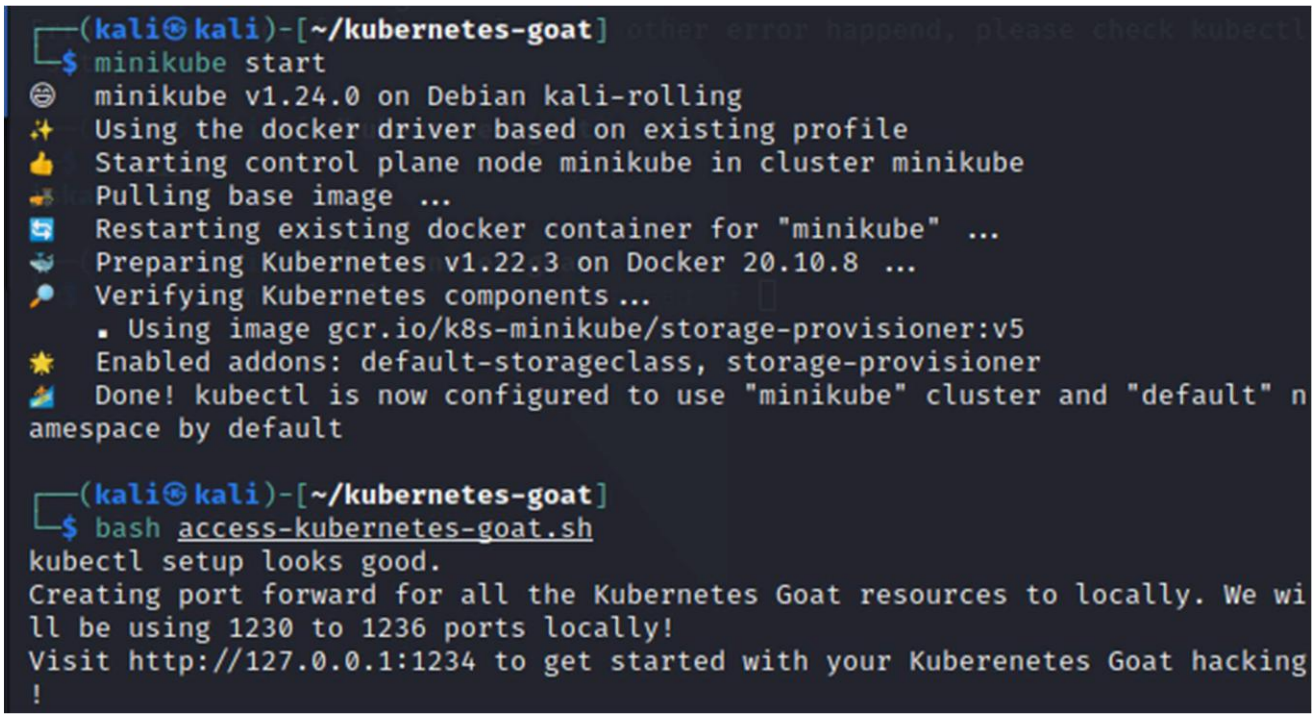


Рисунок 2.1 – Налаштування Kuber-goat

Програмне забезпечення оснащене зручним графічним інтерфейсом користувача, доступним за локальною адресою http://127.0.0.1:1234. У ньому

можна ідентифікувати веб-сторінки (рис. 2.2), які є вразливими до різноманітних типів атак.

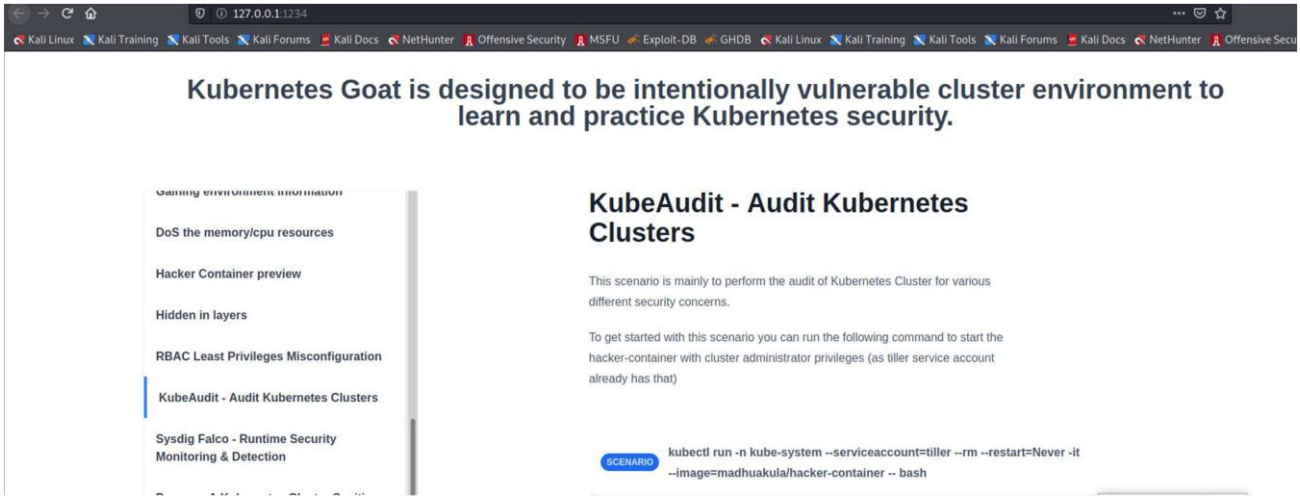


Рисунок 2.2 – Вікно застосунку для тестування на VM

В рамках магістерського дослідження було створено спеціалізований програмний інструмент, призначений для комплексного аналізу безпеки та виявлення вразливостей у контейнеризованих середовищах. Розроблене рішення здійснює перевірку відповідності конфігурацій Docker та Kubernetes вимогам міжнародних стандартів CIS (Center for Internet Security). Застосунок забезпечує користувачів детальною та оперативною оцінкою потенційних ризиків безпеки їхніх Kubernetes-кластерів. Система виконує ретельне сканування всіх контейнерних образів, що функціонують у кластері, охоплюючи як прикладні, так і системні компоненти, а також аналізує конфігураційні файли. Важливою особливістю є те, що інструмент працює безпосередньо з активними образами кластера, без необхідності сканування всіх реєстрів або попередньої інтеграції з системами безперервної інтеграції CI/CD.

Даний інструмент характеризується високим ступенем гнучкості налаштувань, що надає користувачам можливість самостійно визначати ключові параметри сканування. Зокрема, можна конфігурувати цільові простори імен для аналізу, регулювати швидкість виконання процесів, а також встановлювати пріоритетність рівнів уразливостей, які становлять інтерес для дослідження.

Завдяки спеціально розробленому програмному коду на мові Python із інтегрованими бібліотеками, які забезпечують ефективне виконання та візуалізацію результатів у цільовій системі, вдалося досягти максимальної автоматизації робочого процесу. Це мінімізує необхідність активного втручання кінцевого користувача, оскільки для отримання комплексних результатів сканування достатньо лише ініціювати запуск скрипта однією командою у терміналі, що значно спрощує та прискорює процес тестування безпеки. Лістинг 2.1 демонструє частину програмного коду.

Лістинг 2.1 – Частина програмного коду

```
      DEFAULT_CONFIG = {
      {"name": "local", "type": "local"},
      {"name": "kube-bench", "cmd": "kube-bench --json"},
      {"name": "kubeaudit", "cmd": "kubeaudit all --output json"},
      {"name": "kube-hunter", "cmd": "kube-hunter --report json"},
      }
      class KubeBenchScanner:
          """Сканування відповідності CIS Kubernetes Benchmark"""
          def __init__(self, output_dir: str = "./reports"):
              self.output_dir = Path(output_dir)
              self.output_dir.mkdir(exist_ok=True)
          def run_scan(self) -> Dict:
              """Запуск kube-bench сканування"""
              logger.info("Запуск kube-bench сканування...")
      class KubeAuditScanner:
          """Аудит конфігурацій Kubernetes"""
          def __init__(self, output_dir: str = "./reports"):
              self.output_dir = Path(output_dir)
              self.output_dir.mkdir(exist_ok=True)
          def run_scan(self, namespace: str = "all") -> Dict:
              """Запуск kubeaudit сканування"""
              logger.info("Запуск kubeaudit сканування...")
      class KubeHunterScanner:
          """Пентестинг та пошук вразливостей Kubernetes"""
          def __init__(self, output_dir: str = "./reports"):
              self.output_dir = Path(output_dir)
              self.output_dir.mkdir(exist_ok=True)
          def run_scan(self, remote: Optional[str] = None, pod:
bool = False) -> Dict:
              logger.info("Запуск kube-hunter сканування...")
```

кінець лістингу 2.1

Представлений скрипт ініціює процес сканування визначеного програмного застосунку, здійснює пошук всіх файлів, призначених для подальшого аналізу, зберігає знайдені дані в окремому файлі для подальшої обробки, а також відображає отримані результати виконання операції безпосередньо в консолі користувача для зручного моніторингу процесу сканування

Процес аналізу конфігураційних файлів розпочинається з етапу збору конфігураційних даних, які автоматично передаються до спеціалізованого препроцесора. Цей компонент системи виконує важливу функцію розділення вхідного потоку інформації, виокремлюючи кожну окрему конфігурацію для детального індивідуального аналізу відповідних файлів.

Після завершення фази збору даних активується механізм придушення збірника файлів, що слугує сигналом для запуску основного модуля системи – сканера безпеки. Цей ключовий компонент витягує еталонні конфігурації зі спеціалізованого сховища та проводить їх детальне зіставлення з тестовими зразками. На основі комплексного аналізу сканер приймає обґрунтовані рішення щодо коректності кожної перевіреної конфігурації.

Система автоматично ідентифікує та виокремлює виключно потенційно небезпечні налаштування, тоді як валідні конфігурації виключаються з подальшого процесу аналізу для оптимізації продуктивності. На завершальному етапі виявлені проблемні конфігурації класифікуються за рівнем критичності для кожного файлу окремо, отримують спеціальні маркери небезпечності та доповнюються детальною інформацією щодо методів їх усунення та рекомендацій з виправлення.

Для комплексного тестування розробленого програмного забезпечення була проведена ретельна підготовча робота, що включала налаштування та конфігурацію необхідного інструментарію.

Насамперед, було здійснено повноцінне налаштування кластера Kubernetes з відповідною конфігурацією kubeconfig, розміщеною у директорії ~/.kube/config, що забезпечило стабільне з’єднання з цільовим кластером.

Паралельно встановлено kubectl – потужний інструмент командного рядка Kubernetes, який надає широкий спектр можливостей для управління кластером, включаючи виконання перевірок сухого запуску (dry-run) для YAML-файлів, що дозволяє виявити потенційні помилки до фактичного розгортання.

Для проведення комплексного тестування розробленого інструменту забезпечення аудиту безпеки Kubernetes-середовищ було обрано як цільовий об’єкт дослідження вищезгаданий навчальний застосунок Kubernetes-goat. Даний застосунок було успішно розгорнуто у локальному середовищі на спеціально підготовленій віртуальній машині. Попередньо було налаштовано всі необхідні компоненти інфраструктури, зокрема Docker для контейнеризації, minikube для емуляції Kubernetes-кластера, а також встановлено повний набір обов’язкових бібліотек та залежностей для коректного функціонування системи. На рисунку 2.3 продемонстровано розгортання застосунку.

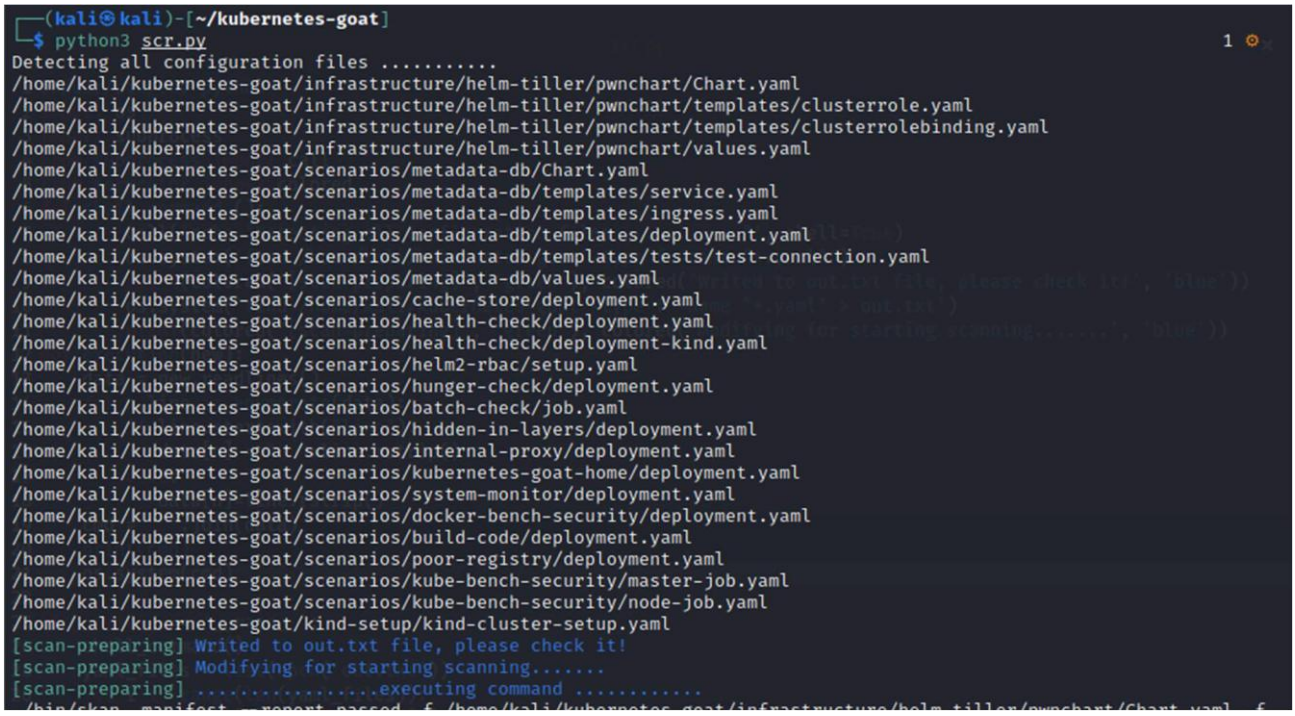


Рисунок 2.3 – Процес розгортання застосунку

Після ретельної перевірки та підтвердження стабільної роботи тестового застосунку було визначено його точне розташування у файловій системі та ініційовано виконання Python-скрипту для проведення автоматизованого

сканування безпеки. Під час виконання процесу аудиту результати сканування виводилися у реальному часі безпосередньо в консоль локальної машини, що дозволило наочно спостерігати за процесом аналізу та виявленими вразливостями. Результати роботи інструменту сканування в консолі проілюстровано на рисунку 2.4.

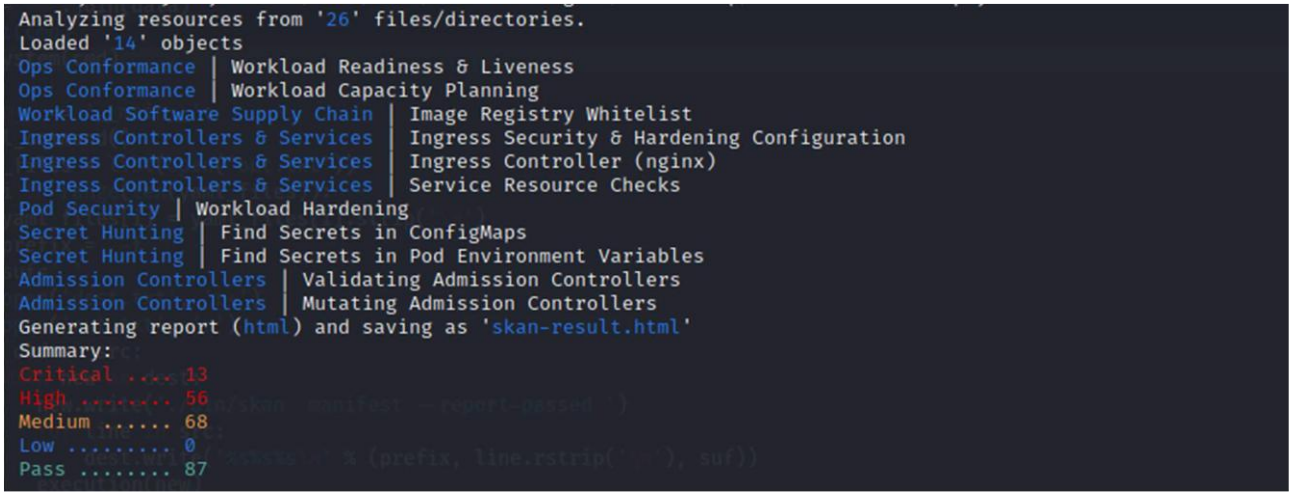


Рисунок 2.4 – Результати роботи інструменту сканування

Окрему увагу приділено валідації конфігураційних файлів: для цього використано Config Lint, що дозволяє перевіряти конфігурації згідно зі спеціальними правилами, визначеними у YAML-форматі. Завершальним етапом підготовки стало розгортання minikube – зручного open-source рішення, яке широко застосовується в корпоративному середовищі для локальних експериментів та налагодження, створюючи повнофункціональний одновузловий Kubernetes-кластер безпосередньо на робочій станції розробника.

Наступним етапом дослідження стало проведення детальної мануальної перевірки результатів, яка мала на меті визначити, які саме з десяти запланованих сценаріїв атак можливо виявити виключно за допомогою автоматизованого інструменту безпеки. Результати цієї ретельної перевірки продемонстрували досить високу ефективність розробленого рішення: вісім із десяти сценаріїв атак, які були детально описані в офіційній документації проєкту Kubernetes-goat, можна успішно попередити шляхом правильного

налаштування відповідних конфігурацій безпеки. Особливо вражає той факт, що тривалість роботи створеного автоматизованого інструменту виявилася надзвичайно короткою – усього двадцять п’ять секунд, тоді як додаткова валідація отриманих результатів та комплексна мануальна верифікація зайняли приблизно дві години робочого часу.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи було детально представлено процес створення спеціалізованого програмного забезпечення, призначеного для проведення пасивного аудиту безпеки інфраструктури Kubernetes. Розробка ґрунтувалася на результатах попередніх всебічних досліджень, які охоплювали моделювання дерев атак та систематичний аналіз потенційних загроз, визначення оптимального методу сканування, а також ретельний вибір технологічного стеку та відповідних інструментів для реалізації.

Для апробації створеного рішення в якості тестового середовища було обрано Kubernetes-goat – загальнодоступну безкоштовну платформу, спеціально розроблену для навчання та тестування безпеки. Як основну мову програмування було визначено Python завдяки його універсальності, багатому екосистемі бібліотек та ідеальній придатності для вирішення поставлених завдань.

У процесі тестування розробленого інструменту було успішно ідентифіковано численні вразливості та потенційні загрози в конфігураціях системи. Після ретельної валідації та всебічного аналізу результатів встановлено, що застосунок забезпечує покриття восьми з десяти запланованих сценаріїв атак, що підтверджує його ефективність та повне задоволення встановлених функціональних вимог.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ МОДУЛЮ БЕЗПЕКИ ІНФОРМАЦІЇ

3.1 Методика проведення дослідження

Методика дослідження спрямована на комплексне вивчення стану безпеки легких Kubernetes-кластерів та визначення ефективності сучасних інструментів аналізу безпеки, до яких належать kube-hunter, kube-bench та kubeaudit. Для досягнення поставленої мети розроблено цілісну експериментальну процедуру, яка включає підготовку середовища, проведення базових вимірювань, моделювання вразливостей, повторний аудит, порівняння результатів та формування висновків щодо підвищення рівня захищеності контейнерної інфраструктури. Методика побудована відповідно до принципів об’єктивності, відтворюваності та системності, що забезпечує достовірність і практичну цінність отриманих результатів.

Робоча гіпотеза дослідження полягає у припущенні, що комплексне застосування спеціалізованих інструментів kube-hunter, kube-bench та kubeaudit в рамках єдиної оркестрованої системи забезпечує суттєво вищий рівень виявлення вразливостей та повноти аудиту безпеки Kubernetes-кластерів порівняно з використанням цих інструментів окремо, завдяки взаємному доповненню їх функціональних можливостей та покриттю різних векторів атак і аспектів безпеки.

Серед загальнонаукових методів ми звернули увагу на:

– аналіз і синтез застосовуються для розкладання складної системи безпеки Kubernetes на окремі компоненти з подальшим їх вивченням та інтеграцією у цілісне розуміння механізмів захисту;

– індукція та дедукція використовуються для формування загальних закономірностей на основі результатів окремих експериментів та виведення конкретних висновків із загальних принципів безпеки;

– абстрагування дозволяє виокремити найсуттєвіші характеристики вразливостей та механізмів захисту, відволікаючись від другорядних деталей реалізації;

– моделювання застосовується для створення спрощених, але адекватних моделей загроз безпеки та сценаріїв атак на контейнерну інфраструктуру;

– системний підхід забезпечує розгляд Kubernetes-кластера як складної системи з множиною взаємопов’язаних компонентів, що впливають на загальний рівень безпеки.

Спеціальні методи проведення дослідження інформаційної безпеки передбачають використання:

– методу аналізу вразливостей, що передбачає систематичне виявлення слабких місць у конфігурації та архітектурі системи з використанням автоматизованих сканерів та ручного аналізу;

– методу пентестингу, що реалізується через імітацію дій потенційного зловмисника для виявлення експлуатованих вразливостей у контрольованих умовах;

– методу аудиту відповідності, який полягає у верифікації конфігурацій системи відносно встановлених стандартів безпеки, зокрема CIS Kubernetes Benchmark;

– методу статичного аналізу конфігурацій, що передбачає перевірку файлів маніфестів, політик RBAC та інших конфігураційних параметрів без виконання коду;

– методу динамічного тестування, що включає активне сканування працюючого кластера з виявленням вразливостей у реальному часі.

У процесі дослідження безпеки легковагових контейнерних оркестраційних платформ важливим є застосування інструментів, що забезпечують комплексну, багатовимірну та стандартизовану оцінку стану захищеності Kubernetes-кластера. Вибір інструментів базується на таких принципах: відповідність сучасним рекомендаціям безпеки Kubernetes (CNCF, CIS, NIST); можливість автоматизованого виявлення вразливостей; сумісність із

lightweight Kubernetes-платформами (k3s, MicroK8s, kind); покриття різних площин безпеки (мережевий рівень, конфігураційний рівень, відповідність стандартам).

На основі аналізу світових практик і вимог до безпеки Kubernetes обрано три ключові інструменти дослідження: kube-bench, kubeaudit та kube-hunter. Разом вони забезпечують повне охоплення критичних компонентів безпеки – інфраструктурної, конфігураційної та мережевої.

Kube-bench характеризує open-source інструмент, розроблений компанією Aqua Security для автоматизованої перевірки відповідності Kubernetes встановленим вимогам CIS Kubernetes Benchmark. Стандарт CIS є загальноприйнятим у сфері кібербезпеки та широко використовується для оцінки рівня безпеки хмарних та контейнерних платформ. Інструмент виконує аудит конфігурації таких ключових компонентів Kubernetes: kube-apiserver (автентифікація, авторизація, доступність інтерфейсів); kube-controller-manager; kube-scheduler; etcd (шифрування, політики доступу); kubelet (параметри безпеки вузлів); контрольні параметри вузлів (OS-level).

Використання kube-bench дозволяє:

– оцінити базову відповідність кластеру міжнародним стандартам безпеки;

– ідентифікувати системні конфігураційні недоліки;

– порівняти безпеку перед та після впровадження контрзаходів;

– отримати структуровані звіти у форматах JSON, YAML, TXT, що спрощує науковий аналіз.

Таким чином, kube-bench є інструментом еталонного рівня, який дозволяє встановити «вимірювальну шкалу» для подальших експериментів.

Kubeaudit – це утиліта від Shopify, що виконує автоматизований аудит ресурсів Kubernetes з метою виявлення небезпечних конфігурацій на рівні: подів, контейнерів, deployment’ів, securityContext, RBAC-політик. На відміну від kube-bench, який орієнтований на системні компоненти, kubeaudit зосереджується на конфігураціях робочих навантажень. Інструмент перевіряє понад 20 типів

небезпечних налаштувань, зокрема: запуск контейнера від root (runAsUser=0); можливість привілейованого виконання (privileged: true); відсутність readOnlyRootFilesystem; надмірні Linux capabilities (CAP_SYS_ADMIN, CAP_NET_RAW); відсутність обмежень ресурсів (resources.limits); відсутність Pod Security Standards; використання застарілих API версій; небезпечні параметри мережевої політики; надмірні RBAC-привілеї.

Під час проведення дослідження kubeaudit дає можливість:

– аналізувати саме ті параметри, які найбільш часто стають причиною експлойтів;

– моделювати та виявляти привілейовані ескалації;

– проводити автоматизований аудит масштабованих кластерів;

– використовувати інструмент як частину CI/CD-конвеєру (DevSecOps-підхід).

У рамках дослідження kubeaudit забезпечує повний огляд помилок у розробці та конфігуруванні контейнерних маніфестів, що є критично важливим для легких Kubernetes-платформ, де безпека часто залежить саме від конфігурацій.

Kube-hunter в роботі використовується як інструмент активного та пасивного тестування безпеки Kubernetes, також створений Aqua Security. На відміну від попередніх інструментів, kube-hunter орієнтований не на конфігурації, а на мережеву площину атак, перевіряючи доступність компонентів кластера та можливість експлуатації мережевих вразливостей. До основних типів сканування відносять:

– пасивне сканування, де відбувається збір інформації про відкриті порти, API-сервер, kubelet, dashboard тощо.

– активне тестування, що забезпечує імітацію атак із каталогом known exploits (API Server доступність без автентифікації, небезпечні метрики kubelet, відкриті insecure endpoints, доступ до сервісів у мережевому просторі кластеру, можливість SSRF або перехоплення токенів).

Kube-hunter виконує роль інструмента «червоного командування» (Red Teaming), забезпечуючи: тестування кластера з позиції потенційного зловмисника; перевірку мережевої сегментації; виявлення неправильно відкритих сервісів; оцінку ризику експлуатації компонентів.

Для lightweight Kubernetes це особливо важливо через спрощені конфігурації; збільшену кількість мережевих endpoint'ів; потенційні помилки у налаштуваннях мережевих плагінів (Flannel, Cilium, Calico). Таким чином, kube-hunter доповнює інші інструменти, перевіряючи захист perimeter-security та внутрішніх мережевих шляхів. Комбінація kube-bench, kubeaudit та kube-hunter забезпечує комплексну оцінку безпеки Kubernetes-кластера за трьома параметрами, що продемонстровано в таблиці 3.1

Таблиця 3.1 – Комплексна оцінка безпеки Kubernetes-кластера

Інструмент	Рівень безпеки	Що перевіряє	Наукова цінність
kube-bench	інфраструктурний	відповідність CIS Benchmark, системні налаштування	дає стандартизовані метрики, створює основу для порівняльного аналізу
kubeaudit	конфігураційний	параметри контейнерів, Pod Security, RBAC, capabilities	виявляє реальні помилки конфігурації робочих навантажень
kube-hunter	мережевий	відкриті порти, мережеві вразливості, доступність API	моделює поведінку зловмисника, тест мережевої площини атак

Таким чином, ці інструменти не дублюють, а доповнюють один одного, створюючи трикомпонентну модель аудиту: стандарти (kube-bench); конфігурації (kubeaudit); мережеві загрози (kube-hunter). Це забезпечує повне охоплення безпеки lightweight Kubernetes-кластеру та дозволяє провести достовірне, багатоаспектне дослідження. Вибір інструментів зумовлений їх широким застосуванням у DevSecOps-практиках, сумісністю з легку ваговими контейнерними платформами та можливістю надання структурованих звітів Таким чином, застосований набір інструментів є оптимальним для вирішення завдань дослідження та отримання об'єктивних результатів щодо безпеки Kubernetes-орієнтованих систем.

3.2 Обробка та аналіз отриманих результатів

У рамках проведеного дослідження було здійснено комплексний аудит безпеки оркестраційних рішень для легких контейнеризаційних середовищ, зокрема таких платформ, як microK8s, K3s та Minikube. Ці системи є спрощеними та оптимізованими реалізаціями Kubernetes, що розроблені для більш швидкого розгортання кластерів, зниження вимог до ресурсів та спрощення адміністративних операцій. Вони широко використовуються у тестових, навчальних та edge-сценаріях, що робить питання їхньої безпеки особливо актуальним.

Під час дослідження ми застосували набір інструментів для аналізу безпеки, зібраних із відкритих репозиторіїв на платформі GitHub. Одним із ключових завдань стало оцінити їхню практичну ефективність, надійність, ступінь довіри зі сторони спільноти та відповідність сучасним вимогам інформаційної безпеки. Не менш важливою була перевірка можливості адаптації та застосування обраних інструментів у межах нашого тестового стенду.

Додатковим критерієм оцінки слугувала популярність відповідних репозиторіїв, адже вона часто відображає активність їхнього розвитку та рівень довіри користувачів. У контексті GitHub кількість форків зазвичай свідчить про інтерес розробників до подальшої модифікації коду, тоді як кількість зірок є індикатором підтримки та визнання корисності проєкту спільнотою.

Під час проведення огляду літератури ми зосередили увагу на виявленні проблем безпеки, які залишаються актуальними навіть у контексті найпоширеніших найкращих практик та безкоштовних інструментів для сканування безпеки. Аналіз наукових праць, технічної документації та звітів з кібербезпеки дав нам змогу простежити обмеження, притаманні цим підходам, а також визначити потенційні прогалини, що можуть вплинути на ефективність захисту систем. Літературний огляд показав, що навіть широко рекомендовані практики безпеки та популярні безкоштовні сканери нерідко мають низку

вразливостей: від обмежених можливостей виявлення складних загроз до затримок у впровадженні оновлень.

З огляду на стрімкий розвиток технологій у сфері кіберзахисту, навіть ті методики, які сьогодні позиціонуються як ефективні та сучасні, можуть швидко втратити актуальність під впливом нових підходів, алгоритмів та архітектур. Саме тому важливо систематично переглядати інформаційні джерела та перевіряти, наскільки запропоновані практики залишаються дієвими в сучасних умовах. Для цього ми застосували метод фільтрації отриманих матеріалів, вилучивши застарілі або такі, що втратили релевантність. Це дало змогу сформувати більш точне та актуальне уявлення про стан безпеки, на яке можна спиратися у подальших дослідженнях.

Для оцінки практичної придатності інструментів проведено бенчмаркінг, який передбачав аналіз таких показників:

– CIS Compliance (відповідність стандарту CIS Benchmark, визначена за допомогою kube-bench);

– кількість критичних, середніх та низькорівневих вразливостей (визначена за допомогою kubeaudit та kube-hunter);

– рівень RBAC безпеки (оцінка повноти та мінімальності привілеїв);

– мережеві ризики (відкриті порти, анонімний доступ до API, небезпечні kubelet endpoints);

– тривалість сканування та детальність звітів.

Узагальнені показники безпеки продемонстровано в таблиці 3.2.

Таблиця 3.2 – Узагальнені показники безпеки

Інструмент	CIS Compliance	критичні вразливості	середні вразливості	низькорівневі вразливості	рівень RBAC безпеки	мережеві ризики	час сканування (хв)
kube-bench	0.78	3	5	2	-	-	12
kubeaudit	-	4	6	3	0.65	-	8
kube-hunter	-	2	4	1	-	0.8	10

На рисунках 3.1, 3.2 та 3.3 наведено результати сканування, що ілюструють ефективність інтегрованого підходу та різноманіття виявлених вразливостей.

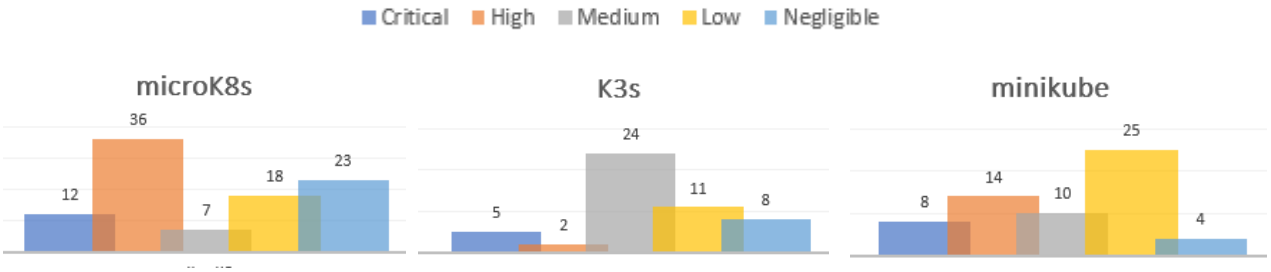


Рисунок 3.1 – Результати пошуку вразливостей kube-hunter

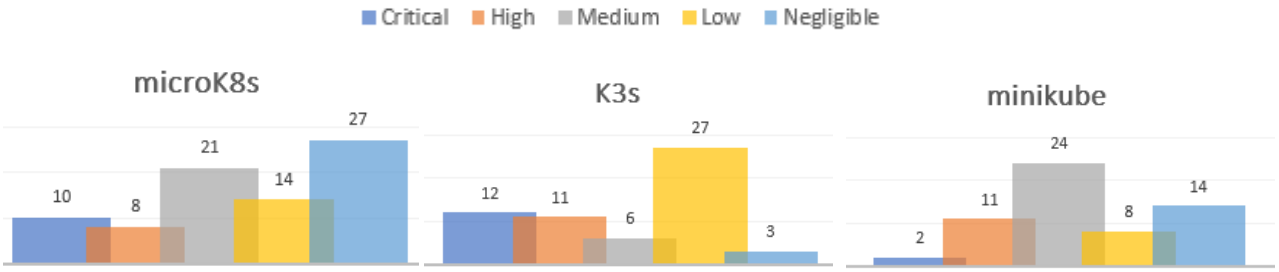


Рисунок 3.2 – Результати оцінки безпеки kube-bench

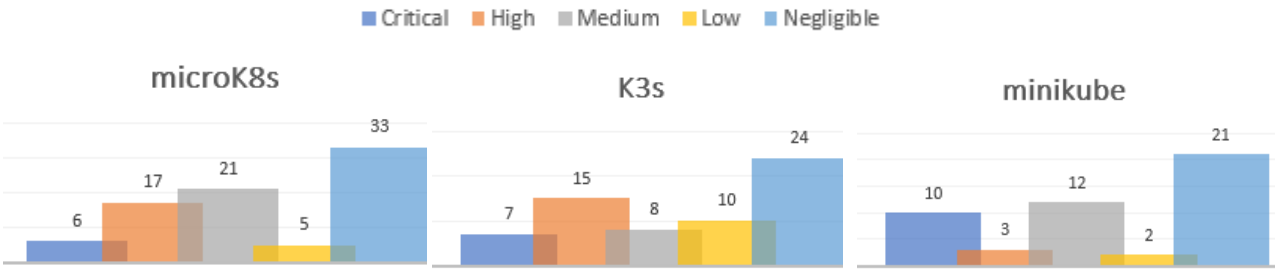


Рисунок 3.3 – Результати аудиту kubeaudit

Результати показали, що kube-bench дозволяє оцінити відповідність конфігураційним стандартам, kubeaudit ефективно виявляє конфігураційні та політичні порушення, а kube-hunter забезпечує виявлення мережевих та експлуатаційних загроз. Використання цих інструментів у комплексі забезпечує багаторівневий аудит безпеки, дозволяє систематично класифікувати вразливості за категоріями та формувати кількісні показники безпеки, придатні

для подальшого порівняльного аналізу та інтеграції у тестовий набір мікросервісів.

Такий інтегрований підхід забезпечує багаторівневий аудит безпеки, дозволяє систематично класифікувати вразливості за категоріями (конфігураційні, мережеві, інфраструктурні) та формувати кількісні показники безпеки кластеру. На рисунках наведено результати сканування, що ілюструють ефективність комплексного використання обраних інструментів.

Забезпечення кібербезпеки платформи Kubernetes у сучасному корпоративному середовищі здійснюється через комплексне застосування двох фундаментальних практичних стратегій. Перша стратегія ґрунтується на систематичному конфігуруванні параметрів безпеки (security hardening) та послідовній імплементації загальновизнаних рекомендованих практик (security best practices) на всіх критичних компонентах інфраструктури Kubernetes, включаючи головні вузли (master nodes), робочі вузли (worker nodes), мережеві політики та системи управління доступом. Друга стратегія передбачає ретельну інтеграцію спеціалізованих сторонніх інструментів та рішень безпеки для забезпечення безперервного моніторингу, всебічного контролю та ефективного управління загальним рівнем захищеності кластерної інфраструктури.

Впровадження технології оркестрування контейнерів суттєво сприяє практичному розвитку концепції незмінної інфраструктури (immutable infrastructure), де контейнеризовані застосунки не піддаються оновленню чи модифікації, а повністю замінюються новими версіями образів. Такий підхід радикально зменшує потенційний ризик виникнення та накопичення критичних вразливостей, які могли б тривалий час зберігатися та поширюватися у продуктивному робочому середовищі. Для успішного розгортання незмінної інфраструктури необхідні надійні та стабільні CI/CD пайплайни, розвинута система керування версіями коду та конфігурацій, а також автоматизовані механізми верифікації. Забезпечення консистентної побудови, безпечної доставки та валідації контейнерних образів виступає визначальним фактором

для підтримки високого рівня безпеки протягом усього життєвого циклу застосунку.

До ключових системних недоліків існуючих підходів варто віднести обмежену глибину та ефективність сканування окремих застосунків, їхню надмірну ресурсну інтенсивність взаємодії з інфраструктурою, що створює значні перешкоди для проведення комплексного тестування масштабних розподілених програмних продуктів. Водночас застосування автоматизованих аналізаторів безпеки середовища Kubernetes вимагає постійного кваліфікованого контролю та втручання з боку DevSecOps-фахівців для запобігання потенційним негативним наслідкам. Зменшення загальної кількості можливих сценаріїв у моделі загроз значною мірою залежить від фундаментального удосконалення процесу проектування програмного забезпечення, що включає статичну перевірку конфігурацій та параметрів захисту.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи продемонстровано обробку отриманих результатів та методику проведення дослідження. Дослідження аналізує загрози контейнерного розгортання, охоплюючи вразливості платформ оркестрування, середовищ виконання та операційних систем. Спеціалізований інструментарій забезпечує систематичну оцінку безпеки контейнерних середовищ.

Запропонований підхід уможливлює послідовну перевірку захищеності контейнерних систем. Регулярні аудити безпеки дозволяють організаціям проактивно виявляти вразливості та протидіяти потенційним атакам. Інтеграція захисних механізмів упродовж усього життєвого циклу контейнерів у поєднанні з інноваційними технологіями формує комплексну стратегію, що підвищує спроможність компаній своєчасно виявляти загрози та ефективно реагувати на них, мінімізуючи ризики безпеки контейнерної інфраструктури.

ВИСНОВКИ

У даній кваліфікаційній роботі магістра було запропоновано реалізацію та дослідження системи безпеки за допомогою оркестрації легких контейнерних платформ. Для виконання поставлених завдань кваліфікаційної роботи було здійснено огляд і аналіз предметної області проблеми, результати існуючих теоретичних та експериментальних досліджень; огляд і аналіз методів та засобів безпеки оркестрації контейнерних платформ для вирішення проблеми дослідження; обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання; реалізовано експериментальне дослідження модулю безпеки інформації.

В кваліфікаційній роботі було виконано основні завдання дослідження, а саме:

– на основі проведеного аналізу виявлено, що сучасні підходи до забезпечення безпеки в середовищах Kubernetes базуються на багаторівневій архітектурі захисту. Повнофункціональний Kubernetes пропонує вбудовані механізми RBAC (Role-Based Access Control), мережеві політики (Network Policies), контроль доступу на рівні Pod Security Standards та інтеграцію з зовнішніми системами управління секретами. Водночас легкі дистрибутиви, такі як K3s та Minikube, демонструють спрощену архітектуру безпеки зі зменшеним набором компонентів, що одночасно знижує поверхню атаки, але й обмежує можливості тонкого налаштування політик безпеки;

– систематична ідентифікація загроз виявила п’ять критичних категорій ризиків для легких систем оркестрації контейнерів. Несанкціонований доступ до API-сервера визначено як найбільш критичну загрозу, оскільки компрометація цього компонента надає зловмиснику повний контроль над кластером. Аналіз показав, що в легких дистрибутивах API-сервер часто розгортається з мінімальними налаштуваннями безпеки, використовуючи HTTP замість HTTPS в окремих конфігураціях або слабкі механізми автентифікації;

– розроблена архітектура системи безпеки базується на принципах defense-in-depth та zero-trust, адаптованих для специфіки легких контейнерних платформ. Архітектура включає три ключові модулі, які працюють інтегровано для забезпечення комплексного захисту кластера. Загальна архітектура побудована на мікросервісних принципах з чітким розділенням відповідальності між модулями, що дозволяє незалежно розвивати та масштабувати окремі компоненти. Модулі взаємодіють через стандартизовані API та event-driven механізми, забезпечуючи слабку зв'язаність та високу гнучкість системи. Архітектура також передбачає механізми самодіагностики та автоматичного відновлення для підвищення надійності системи безпеки.

– практичне тестування прототипу системи безпеки проводилося на трьох репрезентативних легких платформах: microK8s, K3s та minikube. Тестове середовище включало віртуальні машини з обмеженими ресурсами (2 CPU, 4GB RAM), що відповідає типовим умовам розгортання легких платформ.

Дослідження показало, що основні відмінності між повноцінним Kubernetes та його легкими версіями полягають у спрощеній конфігурації компонентів площини управління, зменшеній кількості залежностей та оптимізованому використанні ресурсів. K3s замінює etcd на SQLite за замовчуванням, об'єднує компоненти в єдиний бінарний файл та спрощує процес розгортання, що робить його привабливим для edge-обчислень та IoT-сценаріїв. Minikube, орієнтований на локальну розробку, надає гнучкість у виборі runtime та драйверів, але потребує додаткової уваги до налаштувань безпеки в продуктивному середовищі.

Критичним висновком є те, що незважаючи на спрощення архітектури, легкі платформи зберігають фундаментальні механізми безпеки Kubernetes, включаючи автентифікацію через сертифікати, токени та webhooks, авторизацію через RBAC, та можливості аудиту. Проте їх конфігурація за замовчуванням часто орієнтована на зручність використання, а не максимальну безпеку, що вимагає додаткового затвердіння (hardening) для продуктивного використання.

Застосування оркестрування контейнерів каталізує еволюцію до парадигми незмінної інфраструктури (immutable infrastructure), де контейнери функціонують за принципом повної заміни замість часткового оновлення. Ця методологія радикально знижує ймовірність накопичення критичних вразливостей, які потенційно могли б персистувати в продуктивному середовищі через застарілі конфігурації чи незастосовані патчі безпеки.

Успішна реалізація незмінної інфраструктури вимагає створення надійних CI/CD пайплайнів із вбудованими механізмами автоматизації та ефективної системи версіонування артефактів. Гарантування консистентності процесів побудови, тестування та доставки образів контейнерів стає критично важливим чинником для підтримки високого рівня безпеки. Автоматизоване сканування вразливостей на етапі побудови, підписування образів та використання приватних реєстрів додатково посилюють захисні механізми. Така інтегрована стратегія дозволяє організаціям мінімізувати поверхню атаки, скоротити час реагування на інциденти безпеки та забезпечити передбачуваність поведінки інфраструктури у виробничому середовищі.

Дослідження аналізує загрози контейнерного розгортання, охоплюючи вразливості платформ оркестрування, середовищ виконання та операційних систем. Спеціалізований інструментарій забезпечує систематичну оцінку безпеки контейнерних середовищ.

Запропонований підхід уможливлює послідовну перевірку захищеності контейнерних систем. Регулярні аудити безпеки дозволяють організаціям проактивно виявляти вразливості та протидіяти потенційним атакам. Інтеграція захисних механізмів упродовж усього життєвого циклу контейнерів у поєднанні з інноваційними технологіями формує комплексну стратегію, що підвищує спроможність компаній своєчасно виявляти загрози та ефективно реагувати на них, мінімізуючи ризики безпеки контейнерної інфраструктури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Андрущак І., Кошелюк В., Ясашний Д. Аудит безпеки оркестрації легких контейнерних платформ. 3 Міжнародна науково-практична конференція «Modern Trends in the Development of Economy, Technology and Industry» (9-11 квітня 2025 р.), Торонто, Канада. International Scientific Unity. 2025. С. 169-174.

2. Андрущак І., Кошелюк В., Ясашний Д. Про ефективність використання легких контейнерних платформ. XV Міжнародна науково-практична конференція «Scientific research: integration of science and practice for effective development» (15-18 квітня 2025 р.), Флоренція, Італія. International Science Group. 2025. С. 47-52.

3. Андрущак І., Кошелюк В., Ясашний Д. Підвищення безпеки контейнерів за допомогою розгортання honeypot. International Science Journal of Engineering. ISJEA, 4(3). 2025. Pp. 15-26.

4. Мікросервісна архітектура для початківців. Орхан Гасімов. URL: https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/ (дата звернення: 18.09.2025).

5. Мікросервісна архітектура. Ivan Zmerzlyi URL: https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d/ (дата звернення: 20.09.2025).

6. Контейнеризація та ШІ. URL: https://itedu.center/ua/blog/review/konteineryzatsiia_ta_shi/?srsltid=AfmBOooNpNAUWQ4V-kuBsvecEy5vPkHD7VlDrG92c-BFKs1RFJxCQxCj/ (дата звернення: 20.09.2025).

7. A Comprehensive Guide for Web3 Security: From Technology, Economic and Legal Aspects (Future of Business and Finance). Huang K. et al. URL: https://www.springerprofessional.de/en/a-comprehensive-guide-for-web3-security/26568874 (accessed: 11.10.2025)

8. Kubernetes Performance Measurements and Roadmap. URL: https://kubernetes.io/blog/2022/09/kubernetes-performance-measurements-and/ (accessed: 11.10.2025).

9. Налаштування кластера Kubernetes. URL: https://robotdreams.cc/uk/blog/409-rozgortannya-zastosunku-na-ubuntu-nalashtovuyemo-klaster-kubernetes (дата звернення: 20.10.2025).

10. Advantages and Best Practices for Docker to Kubernetes Migration. URL: https://itoutposts.com/blog/advantages-and-best-practices-for-docker-to-kuber netes-migration/ (accessed: 22.10.2025).

11. Vasylyshyn S. et al. Information technologies for the synthesis of rule databases of an intelligent lighting control system. Journal of Theoretical and Applied Information Technologythis link is disabled, 2022. 100(5), Pp. 1340–1353

12. Kube-bench Security Benchmark. URL: https://docs.k0sproject.io/stable/cis_ benchmark/ (accessed: 10.11.2025)

13. Security risk analysis for Kubernetes resources. URL: https://kubesec.io/ (accessed: 10.11.2025)

14. K8s Security. KubeAudit. URL: https://github.com/Shopify/kubeaudit (accessed: 10.11.2025)

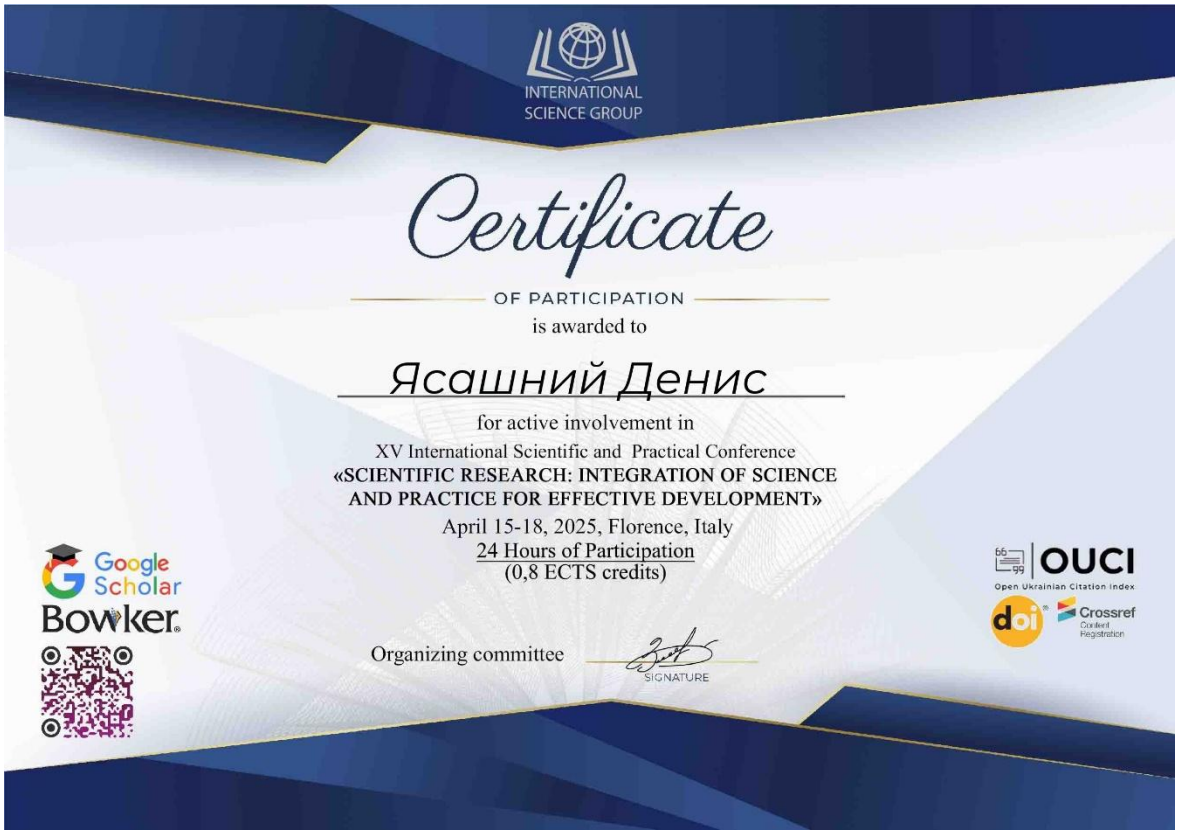
15. Kube-hunter documentation. URL: https://aquasecurity.github.io/kube-hunter/ (accessed: 10.11.2025)

16. Kube-scan. Documentation. URL: https://kube-scan.io/docs/ (accessed: 10.11.2025)

ДОДАТКИ

ДОДАТОК А

Апробація результатів дослідження



ДОДАТОК Б

Автоматичний запуск kube-bench (перевірка CIS Benchmark)

```
#!/bin/bash
DATE=$(date +"%Y-%m-%d_%H-%M")
REPORT_DIR="/opt/security"
REPORT_FILE="$REPORT_DIR/kube-bench-report-$DATE.json"

mkdir -p $REPORT_DIR

echo "[+] Running kube-bench security scan..."

docker run --rm --pid=host \
  -v /etc:/etc \
  -v /var:/var \
  -v $(which kubectl):/usr/local/bin/kubectl \
  aquasec/kube-bench:latest --json > $REPORT_FILE

echo "[✓] CIS Benchmark report saved to: $REPORT_FILE"
```

Активне сканування атак за допомогою kube-hunter

```
#!/bin/bash
DATE=$(date +"%Y-%m-%d_%H-%M")
REPORT_DIR="/opt/security"
REPORT_FILE="$REPORT_DIR/kube-hunter-report-$DATE.txt"

mkdir -p $REPORT_DIR
echo "[+] Running kube-hunter penetration test..."

docker run --rm aquasec/kube-hunter --remote $(hostname -I |
awk '{print $1}') > $REPORT_FILE

echo "[✓] Attack surface scan completed!"
echo "[✓] Report saved to: $REPORT_FILE"
```

Автоматичний аналіз конфігурацій kubeaudit

```
#!/bin/bash
DATE=$(date +"%Y-%m-%d_%H-%M")
REPORT_DIR="/opt/security"
REPORT_FILE="$REPORT_DIR/kubeaudit-report-$DATE.json"

mkdir -p $REPORT_DIR

echo "[+] Running kubeaudit configuration audit..."

kubeaudit all --format json > $REPORT_FILE

echo "[✓] Kubernetes configuration audit completed!"
echo "[✓] Report saved to: $REPORT_FILE"
```

```
import argparse
import asyncio
import json
import os
import shlex
import subprocess
import sys
from datetime import datetime
from typing import Dict, Any, List, Optional

try:
    import yaml
except Exception:
    yaml = None

async def run_command(cmd: str, env: Optional[Dict[str, str]]
= None, timeout: Optional[int] = None) -> Dict[str, Any]:
    """Запустити команду асинхронно та повернути stdout,
stderr, returncode та час виконання."""
    start = datetime.utcnow()
    # розбираємо команду у список для створення процесу
    parts = shlex.split(cmd)
    try:
        proc = await asyncio.create_subprocess_exec(
            *parts,
            stdout=asyncio.subprocess.PIPE,
            stderr=asyncio.subprocess.PIPE,
            env=env,
        )
        try:
            stdout, stderr = await
asyncio.wait_for(proc.communicate(), timeout=timeout)
        except asyncio.TimeoutError:
            proc.kill()
            await proc.wait()
            return {
                "cmd": cmd,
                "timeout": True,
                "stdout": "",
                "stderr": "process timed out",
                "returncode": -1,
                "duration_seconds": (datetime.utcnow() -
start).total_seconds(),
            }
        end = datetime.utcnow()
        return {
            "cmd": cmd,
            "timeout": False,
            "stdout": stdout.decode(errors="ignore"),
            "stderr": stderr.decode(errors="ignore"),
```

```
                "returncode": proc.returncode,
                "duration_seconds": (end - start).total_seconds(),
            }
        except FileNotFoundError:
            return {
                "cmd": cmd,
                "error": "executable not found",
                "stdout": "",
                "stderr": "",
                "returncode": -1,
                "duration_seconds": 0,
            }
        except Exception as e:
            return {
                "cmd": cmd,
                "error": str(e),
                "stdout": "",
                "stderr": "",
                "returncode": -1,
                "duration_seconds": (datetime.utcnow() -
start).total_seconds(),
            }

    async def run_tool_for_target(tool: Dict[str, Any], target:
Dict[str, Any], timeout: Optional[int] = 300) -> Dict[str, Any]:
        cmd = tool.get("cmd")
        env = os.environ.copy()

        try:
            cmd = cmd.format(**target)
        except Exception:
            pass

        result = await run_command(cmd, env=env, timeout=timeout)
        parsed = None
        if result.get("stdout"):
            text = result["stdout"].strip()
            if text.startswith("{") or text.startswith("["):
                try:
                    parsed = json.loads(text)
                except Exception:
                    parsed = None
        result_summary = {
            "tool": tool.get("name"),
            "target": target.get("name"),
            "cmd": cmd,
            "raw": result,
            "parsed_json": parsed,
        }
        return result_summary
```

```
    async def orchestrate(config: Dict[str, Any], concurrency: int
= 3, timeout: Optional[int] = 300) -> Dict[str, Any]:
        targets = config.get("targets", [])
        tools = config.get("tools", [])

        sem = asyncio.Semaphore(concurrency)
        tasks = []

        async def sem_run(tool, target):
            async with sem:
                return await run_tool_for_target(tool, target,
timeout=timeout)

        for target in targets:
            for tool in tools:
                tasks.append(asyncio.create_task(sem_run(tool,
target)))

        results = await asyncio.gather(*tasks)

        aggregated = {
            "generated_at": datetime.utcnow().isoformat() + "Z",
            "summary": {},
            "results": results,
        }

        summary = {}
        for r in results:
            tname = r.get("tool")
            s = summary.setdefault(tname, {"total": 0, "failed":
0, "targets": []})
            s["total"] += 1
            raw = r.get("raw", {})
            rc = raw.get("returncode")
            failed = rc is None or rc != 0
            if failed:
                s["failed"] += 1
            s["targets"].append({
                "target": r.get("target"),
                "returncode": rc,
                "duration": raw.get("duration_seconds"),
                "error": raw.get("error") or raw.get("stderr"),
            })
        aggregated["summary"] = summary
        return aggregated

    def load_config(path: Optional[str]) -> Dict[str, Any]:
        if not path:
            return DEFAULT_CONFIG
        if not os.path.exists(path):
            print(f"Config file {path} not found, using default."
, file=sys.stderr)
            return DEFAULT_CONFIG
```

```
        if yaml is None:
            return DEFAULT_CONFIG
        with open(path, "r", encoding="utf-8") as f:
            cfg = yaml.safe_load(f)
        return cfg

    def save_json(data: Dict[str, Any], path: str) -> None:
        with open(path, "w", encoding="utf-8") as f:
            json.dump(data, f, indent=2, ensure_ascii=False)

    def parse_args():
        p = argparse.ArgumentParser(description="Security
Orchestrator for lightweight container platforms")
        p.add_argument("--config", help="Path to YAML config",
default=None)
        p.add_argument("--out", help="Output JSON file path",
default="orchestrator_report.json")
        p.add_argument("--html", help="Output HTML report path
(optional)", default=None)
        p.add_argument("--concurrency", help="How many scans to
run in parallel", type=int, default=3)
        p.add_argument("--timeout", help="Per-command timeout
seconds", type=int, default=300)
        return p.parse_args()

    def main():
        args = parse_args()
        cfg = load_config(args.config)
        print("Loaded config.")

        try:
            aggregated = asyncio.run(orchestrate(cfg,
concurrency=args.concurrency, timeout=args.timeout))
        except KeyboardInterrupt:
            print("Interrupted by user", file=sys.stderr)
            sys.exit(1)

        save_json(aggregated, args.out)
        print(f"Saved JSON report to {args.out}")
        if args.html:
            generate_html_report(aggregated, args.html)
            print(f"Saved HTML report to {args.html}")

    if __name__ == "__main__":
        main()
```