Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Кластерна система на основі Raspberry Pi із застосуванням платформи Kubernetes
Назва теми

КвРКІ 210492.21.04.26 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент IV курсу, група КІ2-21-4 _____ Ігор ШИЩАК
Підпис Ініціали, прізвище

Керівник _____ Сергій ЛИСЕНКО
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ Тетяна КИСІЛЬ
Підпис, дата Ініціали, прізвище

До захисту допускаю:
зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Ольга ПАВЛОВА
Підпис Ініціали, прізвище

«5» червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень БАКАЛАВР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Ольга ПАВЛОВА

" 10 " 01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Ігорю ШИЩАКУ

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Кластерна система на основі Raspberry Pi із застосуванням платформи Kubernetes

Керівник проекту (роботи) Сергій ЛИСЕНКО, д.т.н., проф.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. № 23

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2025 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Кластерна система на основі Raspberry Pi із застосуванням платформи Kubernetesта постановка задачі щодо її удосконалення

Проєктування кластерної системи на основі Raspberry Pi із застосуванням платформи Kubernetes

Програмно-апаратна реалізація кластерної системи на основі Raspberry Pi із застосуванням платформи Kubernetes

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Архітектура ПЗ проєкту

Архітектура кластерної системи на базі Kubernetes

Розгортання тестового додатка в кластері Kubernetes

Код програмного забезпечення

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Тетяна КИСІЛЬ, доцент кафедри КІС		
Антиплагіат	Андрій НІЧЕПОРУК, доцент кафедри КІС		

7. Дата видачі завдання «10» 01 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2025	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2025	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2025	виконано
4	Робота над розділом 2 – вибір компонентів для проєктування кластерної системи із застосуванням платформи Kubernetes	01.04.2025	виконано
5	Робота над розділом 3 – проєктування кластерної системи із застосуванням платформи Kubernetes	29.04.2025	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2025	виконано
7	Попередній захист ВКР	26.05.2025	виконано
8	Захист ВКР на засіданні ЕК	Червень 2025 року	

Студент _____ Ігор ШИЩАК
Підпис Ініціали, прізвище

Керівник роботи _____ Сергій ЛИСЕНКО
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 210492.21.04.38 ПЗ	Пояснювальна записка	57		
			Графічні матеріали			
2		КвРКІ 210492.21.04.38 Е8	Архітектура ПЗ проєкту	1		
3		КвРКІ 210492.21.04.38 Е8	Архітектура ПЗ для кластерної системи	1		
4		КвРКІ 210492.21.04.38 Е8	Апаратне забезпечення проєкту	1		

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 210492.21.04.26 ВП		
Розробив		Шищак			Відомість проекту	Літера	Аркуш	Аркушів
Перевір.		Лисенко				У	1	57
Н. контр.		Кисіль		05.06.25		ХНУ, КІ2-21-4		
Затв.		Павлова						

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Кластерна система на основі Raspberry Pi із застосуванням платформи Kubernetes».

Автор роботи: Ігор ШИЩАК.

Керівник роботи: Лисенко Сергій Миколайович.

Пояснювальна записка: 57 с., 23 рис., 1 табл., 4 дод., 53 джерел.

Графічна частина: 3 креслення, 1 код програмного забезпечення.

Метою кваліфікаційної роботи є визначення умов та особливостей проєктування кластерної системи на основі одноплатних комп'ютерів Raspberry Pi із застосуванням платформи Kubernetes, а також оцінка можливостей і обмежень апаратних і програмних компонентів для забезпечення ефективної, масштабованої та відмовостійкої роботи кластеру. Особлива увага приділяється адаптації контейнеризованих сервісів під архітектуру ARM, оптимізації ресурсів і забезпеченню зручного управління за допомогою сучасних інструментів оркестрації контейнерів.

Об'єктом дослідження є кластерні обчислювальні системи, реалізовані на основі Raspberry Pi із використанням платформи Kubernetes.

Предметом дослідження є методи і технології проєктування, розгортання та управління кластерними системами на базі ARM-платформи із застосуванням контейнеризації та оркестрації сервісів.

Під час проведення дослідження був використаний метод систематичного огляду наукової і технічної літератури, а також практичний аналіз існуючих рішень та технологій для глибшого розуміння предметної області і визначення оптимальних підходів до побудови кластерної системи на Raspberry Pi.

Підпис студента

30.05.2025

Дата

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасному світі інформаційні технології швидко розвиваються, що зумовлює зростання потреб у високопродуктивних, масштабованих та енергоефективних обчислювальних системах. Одним із напрямків розвитку є кластерні обчислення, які дозволяють об’єднати велику кількість окремих обчислювальних пристроїв у єдину потужну систему, що здатна виконувати складні завдання в реальному часі. Водночас із розвитком апаратних платформ паралельно розвиваються й програмні засоби, серед яких все більшу популярність здобувають технології контейнеризації та оркестрації контейнерів. Платформа Kubernetes, яка є провідним інструментом для автоматизації розгортання, масштабування та управління контейнеризованими додатками, значно спрощує процес створення та підтримки кластерів.

Особливе місце у цій сфері посідають одноплатні комп’ютери, зокрема Raspberry Pi, які завдяки своїй компактності, низькій вартості та достатній обчислювальній потужності можуть слугувати будівельними блоками для побудови кластерних систем різного масштабу. Raspberry Pi на базі ARM-архітектури дозволяє створювати енергоефективні, доступні за ціною та гнучкі у використанні обчислювальні кластери, які можуть бути застосовані як у навчальних закладах, так і у промислових або дослідницьких середовищах.

З огляду на вищезазначене, дослідження, спрямоване на проєктування кластерної системи на основі Raspberry Pi із застосуванням платформи Kubernetes, є надзвичайно актуальним. Воно сприятиме не лише глибшому розумінню принципів роботи кластерів на ARM-платформі, але й дозволить оцінити потенціал контейнеризації в обмежених апаратних середовищах, що важливо для подальшого розвитку розподілених систем.

Мета дослідження полягає у розробці концепції кластерної системи на базі одноплатних комп’ютерів Raspberry Pi із застосуванням платформи Kubernetes, а також у вивченні технічних особливостей апаратних і програмних компонентів

системи. Особлива увага приділяється адаптації контейнеризованих сервісів для роботи на ARM-архітектурі, оптимізації ресурсів і забезпеченню надійності роботи кластеру.

Об’єктом дослідження є кластерні обчислювальні системи, реалізовані на основі одноплатних комп’ютерів Raspberry Pi та керовані за допомогою системи оркестрації контейнерів Kubernetes.

Предметом дослідження виступають методи проєктування, впровадження та оптимізації кластерної системи, а також особливості використання платформи Kubernetes (K3s) на ARM-платформі для забезпечення ефективного розподілу обчислювальних ресурсів та управління контейнерами.

1 ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖУВАНОЇ ПРОБЛЕМИ

1.1 Аналіз предметної області і виявлення наявнії проблем і завдань

В умовах швидкого розвитку інформаційних технологій виникає потреба у створенні розподілених обчислювальних систем для ефективного управління великим обсягом даних та обслуговування численних користувачів. Класичні серверні рішення вимагають значних фінансових витрат, тому альтернативні платформи, такі як Raspberry Pi, набувають все більшої популярності для побудови недорогих, гнучких і масштабованих кластерів.

Raspberry Pi – це одноплатні комп'ютери, які характеризуються невисокою вартістю, компактністю та достатньою обчислювальною потужністю для вирішення широкого спектра завдань (рис.1.1). При об'єднанні кількох таких пристроїв у єдиний кластер можна створити доступну платформу для розробки, тестування та навіть розгортання невеликих сервісів.

Для управління контейнерами в кластері використовується Kubernetes – система з відкритим вихідним кодом, яка автоматизує розгортання, масштабування та експлуатацію контейнерних застосунків.

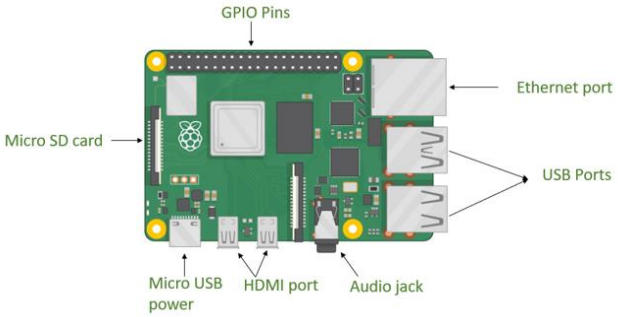


Рисунок 1.1 – Схема Raspberry PI [6]

Кластерна система на основі Raspberry Pi являє собою групу одноплатних комп’ютерів, які працюють разом, щоб забезпечити єдиний обчислювальний ресурс для розподіленого виконання завдань. Управління ресурсами, розподіл

навантаження і автоматичне масштабування контейнеризованих застосунків здійснюється за допомогою платформи Kubernetes.

Основні компоненти системи:

– головний вузол (мастер-нода);

– робочі вузли (воркери);

– архітектура взаємодії;

– запит на створення контейнера подається через Kubernetes API;

– мастер вибирає оптимальний воркер для виконання завдання;

– контейнер автоматично запускається на вибраному вузлі.

Kubernetes забезпечує самовідновлення контейнера у разі збою або падіння вузла (рис 1.2).

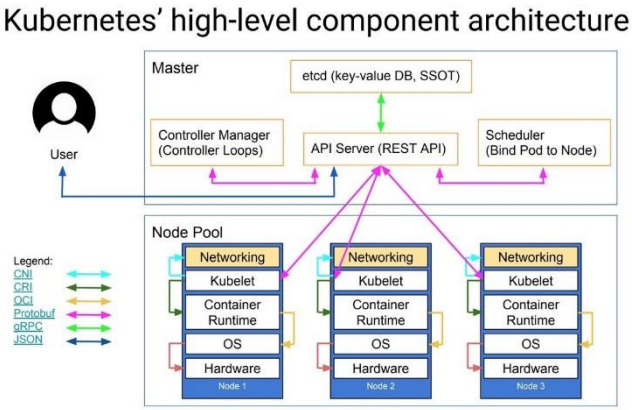


Рисунок 1.2 – Схема кластеру Kubernetes [8]

Головний вузол відповідає за планування розгортання застосунків (подів), моніторинг їх стану та управління усією інфраструктурою.

Робочі вузли виконують розгорнуті контейнери, підтримують їх роботу, повідомляють мастеру про свій стан.

Коли користувач створює або змінює застосунок у Kubernetes, головний вузол розподіляє завдання між робочими вузлами на основі наявних ресурсів (CPU, пам’ять, диск) і політик розгортання.

Попри перспективність створення кластера на базі Raspberry Pi, існують

певні технічні проблеми:

– обмежена обчислювальна потужність;
– затримки введення/виведення;
– пропускна здатність мережі;
– енергоживлення і відмова компонентів;
– ускладнена початкова настройка.

Raspberry Pi – це енергоефективні пристрої з ARM-архітектурою, але їх потужність значно нижча, ніж у серверних процесорів. Це обмежує кількість контейнерів, які можна запустити одночасно.

Використовуються microSD-карти замість SSD або NVMe-накопичувачів суттєво зменшує швидкість читання/запису даних, що впливає на продуктивність баз даних та інших I/O-інтенсивних застосунків (рис 1.3).

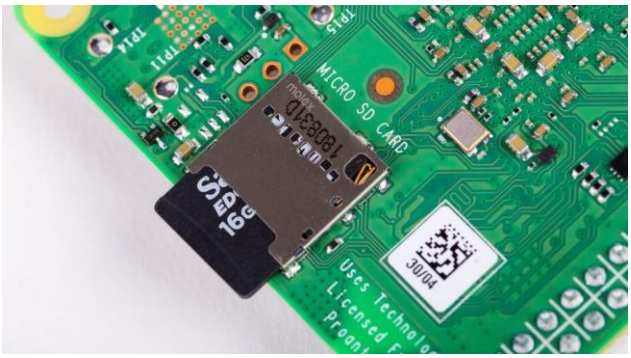


Рисунок 1.3 – підключення SSD через USB до Raspberry Pi [9]

Raspberry Pi 4 має гігабітний Ethernet, однак реальна пропускна здатність може бути нижчою через архітектурні обмеження шини USB/Ethernet.

Погано організоване живлення може призвести до нестабільної роботи кластеру, особливо при великій кількості підключених вузлів.

Початкове налаштування кожного вузла (операційна система, IP-адреси, SSH-доступ, Kubernetes-агенти) потребує значних ручних витрат без використання автоматизації.

Для подолання обмежень, пов'язаних із побудовою кластерної системи на

базі Raspberry Pi, необхідно впроваджувати низку технічних та організаційних рішень. Нижче подано докладний огляд підходів для підвищення надійності, продуктивності й масштабованості кластера.

- вибір оптимальної апаратної платформи;
- підвищення швидкості зберігання даних;
- оптимізація мережевої інфраструктури;
- забезпечення стабільного енергоживлення;
- автоматизація розгортання і обслуговування;
- оптимізація розподілу навантаження.

Використання сучасних моделей Raspberry Pi, таких як Raspberry Pi 4 Model B або Raspberry Pi 5, значно розширює можливості побудови кластерної інфраструктури.

Моделі з 4 ГБ або 8 ГБ оперативної пам'яті забезпечують кращу продуктивність при розгортанні складних контейнеризованих застосунків.

Крім того, варто звертати увагу на ефективність систем охолодження (активні кулери, алюмінієві корпуси), що дозволяє уникнути тротлінгу процесора при високому навантаженні.

Стандартне використання microSD-карт є вузьким місцем у системі введення/виведення даних.

Щоб мінімізувати затримки, рекомендується переходити на зовнішні SSD-диски, підключені через інтерфейс USB 3.0.

Вони мають значно більшу швидкість передачі даних, підвищують стійкість до збоїв і дозволяють використовувати Raspberry Pi для баз даних або файлових серверів.

Під час вибору SSD важливо звертати увагу на підтримку функції TRIM та використання надійних файлових систем, наприклад, ext4 або Btrfs.

Для забезпечення стабільного і швидкого з'єднання між вузлами важливо застосовувати:

- гігабітні Ethernet-комутатори;

– високоякісні Ethernet-кабелі категорії не нижче Cat 5e;

– налаштування статичних IP-адрес або DHCP-резервації для всіх вузлів кластеру;

– крім того, можливе використання VLAN для відокремлення службового трафіку Kubernetes і користувацьких даних.

Кожен Raspberry Pi має отримувати стабільне живлення відповідної потужності (мінімум 5В 3А для Pi 4).

Рекомендується використовувати: оригінальні блоки живлення від Raspberry Pi Foundation, потужні USB-хаби з окремим живленням для підключення великої кількості вузлів, джерела безперебійного живлення (UPS) для захисту від стрибків напруги та короткочасних вимкнень електроенергії (рис.1.4).

Стабільне живлення безпосередньо впливає на надійність роботи кластера та зменшує ймовірність пошкодження файлової системи при аварійному вимкненні.

Рисунок 1.4 – живлення для Raspberry Pi [10]

Для полегшення адміністрування великої кількості пристроїв рекомендується автоматизувати процеси:

Використання інструментів, таких як: ansible, terraform, bash-скрипти для налаштування ОС, мережі та встановлення Kubernetes; автоматизація оновлень системних пакетів і контейнерних застосунків через CI/CD-пайплайни; моніторинг ресурсів і журналювання за допомогою рішень на базі Prometheus, Grafana, Loki.

Автоматизація суттєво знижує кількість людських помилок і прискорює

обслуговування кластерної інфраструктури.

Правильна конфігурація Kubernetes дозволяє ефективно використовувати обмежені ресурси Raspberry Pi:

– налаштування таїнтів і толерантностей для окремих вузлів;

– використання resource requests і limits для контейнерів для запобігання перевантаження вузлів;

– розподіл навантаження через Horizontal Pod Autoscaler або Cluster Autoscaler для автоматичної оптимізації кількості подів.

Ці методи допомагають підтримувати високу доступність застосунків навіть при зростанні навантаження.

Таким чином, завдяки правильно обраному обладнанню, грамотній мережевій архітектурі, стабільному живленню та автоматизації обслуговування можна побудувати ефективну, доступну та масштабовану кластерну систему на основі Raspberry Pi під управлінням Kubernetes.

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

На сьогодні існує декілька популярних підходів до реалізації Kubernetes-кластерів на Raspberry Pi.

Серед них можна виділити такі:

– стандартне розгортання Kubernetes за допомогою kubeadm;

– спрощене рішення K3s;

– microK8s і Docker Swarm.

Стандартне встановлення Kubernetes за допомогою kubeadm є найбільш гнучким і наближеним до "бойового" середовища. Воно дозволяє повністю контролювати конфігурацію, мережу, безпеку та компоненти кластера. Однак цей підхід вимагає значного технічного досвіду, точного налаштування мережевої взаємодії між вузлами та сертифікатів безпеки. Крім того, це рішення досить ресурсомістке і не завжди оптимальне для слабших пристроїв на базі ARM-

архітектури.

K3s є спрощеною і полегшеною версією Kubernetes, спеціально оптимізованою для пристроїв з обмеженими ресурсами, таких як Raspberry Pi. Це рішення легше у встановленні, споживає менше оперативної пам’яті, має вбудовані базові сервіси та підтримує ARM-архітектуру без додаткових налаштувань. Воно ідеально підходить для навчальних або домашніх кластерів. Проте K3s має дещо обмежені функціональні можливості порівняно з повноцінним Kubernetes і не завжди підтримує всі новітні функції одразу після їх появи.

MicroK8s – ще одне рішення, яке дозволяє легко розгорнути кластер однією командою. Воно має вбудовані модулі, такі як dashboard, DNS та monitoring. MicroK8s добре підходить для швидкого тестування, але потребує значно більше ресурсів, ніж K3s, і може працювати нестабільно на старіших моделях Raspberry Pi.

– Docker Swarm – це проста система оркестрації контейнерів, яка споживає мінімум ресурсів і легко налаштовується. Вона може бути корисною для дуже простих кластерів, але не підтримує багатьох можливостей Kubernetes, таких як автоматичне масштабування, складна мережева політика або інтеграція з CI/CD. Також важливо враховувати, що Docker Swarm фактично перестав активно розвиватися і не є рекомендованим вибором для майбутніх проєктів.

Таким чином, для побудови кластерної системи на Raspberry Pi найкращим балансом між простотою, функціональністю та продуктивністю є використання K3s. Якщо ж проєкт вимагає повного контролю і глибокого занурення у Kubernetes-інфраструктуру, варто звернутися до kubeadm, хоча це і складніше.

MicroK8s можна розглядати як проміжний варіант, тоді як Docker Swarm уже втрачає актуальність.

1.3 Підходи до вирішення задачі за темою дослідження

Для побудови кластерної системи з використанням одноплатних комп’ютерів Raspberry Pi та платформи Kubernetes існує кілька підходів, кожен з яких має свої

особливості, переваги та недоліки. Основною метою є створення стабільного, функціонального та легкого у розгортанні середовища, яке ефективно працює в умовах обмежених апаратних ресурсів.

Беручи до уваги обмеження одноплатних комп'ютерів, специфіку ARM-архітектури, вимоги до енергоефективності та зручність у розгортанні, найбільш оптимальним підходом до вирішення задачі є використання платформи K3s. Це рішення дозволяє побудувати повноцінний Kubernetes-кластер із мінімальними витратами часу й ресурсів, при цьому зберігаючи ключові переваги системи оркестрації контейнерів.

1.4 Постанова задачі

Метою проєкту є створення функціонального, масштабованого та енергоефективного кластерного середовища на базі одноплатних комп'ютерів Raspberry Pi із використанням платформи Kubernetes для розподіленого розгортання та управління контейнеризованими застосунками.

Для досягнення мети проєкту необхідно вирішити низку технічних і дослідницьких завдань:

- аналіз апаратної платформи Raspberry Pi;
- оцінка існуючих платформ для кластеризації;
- розробка топології та архітектури кластера;
- розгортання Kubernetes-кластера на Raspberry Pi;
- налаштування моніторингу, безпеки та оновлень;
- тестування та оцінка ефективності роботи системи;
- підготовка висновків та рекомендацій.

Насамперед необхідно проаналізувати можливості, обмеження та сумісність різних моделей Raspberry Pi, зокрема версій 3B, 4B і 5, у контексті кластеризації. Особлива увага приділяється таким параметрам, як обсяг оперативної пам'яті, продуктивність процесора, підтримка мережевих інтерфейсів, енергоспоживання,

а також вимоги до охолодження при довготривалій роботі під навантаженням.

Наступним етапом є вивчення існуючих способів розгортання Kubernetes у середовищах з обмеженими ресурсами, зокрема таких рішень як kubeadm, MicroK8s і K3s. Важливо порівняти їх за критеріями продуктивності, стабільності, сумісності з ARM-архітектурою та простоти налаштування, аби обрати оптимальну платформу для реалізації кластеру на Raspberry Pi.

Після вибору відповідного рішення слід розробити архітектуру системи, визначити кількість вузлів, їх функціональне призначення (майстер або воркер), способи їх підключення та комунікації, а також обрати мережеві й файлові компоненти – наприклад, CNI-плагіни для мережевої взаємодії та рішення для спільного сховища даних. Реалізація процесу ініціалізації кластеру має включати встановлення всіх необхідних компонентів Kubernetes, налаштування автоматичного приєднання нових вузлів і перевірку стабільності роботи кожного з них.

Особливу роль відіграє інтеграція систем моніторингу, таких як Prometheus та Grafana, що дозволяють у реальному часі відстежувати стан кластеру, виявляти потенційні проблеми та аналізувати навантаження на вузли. Також передбачається впровадження базових заходів безпеки для захисту доступу до кластеру та підтримка оновлень програмних компонентів.

На завершальному етапі планується проведення практичного тестування системи: буде оцінено стабільність роботи кластера, швидкість запуску контейнерів, відмовостійкість при втраті окремих вузлів, а також загальне навантаження на ресурси. На основі отриманих експериментальних результатів буде сформовано висновки щодо доцільності використання такого підходу в реальних сценаріях – зокрема в освітніх проєктах, для прототипування або у сфері edge-комп’ютингу – та надано рекомендації щодо можливих напрямів подальшого розвитку й удосконалення системи.

1.5 Висновки до першого розділу

У першому розділі було проведено аналіз предметної області побудови кластерних систем з використанням одноплатних комп’ютерів, зокрема Raspberry Pi, та оркестраційних платформ на зразок Kubernetes. Виявлено основні вимоги до сучасних розподілених обчислювальних систем, зокрема масштабованість, доступність, економічність і простота управління.

Було охарактеризовано існуючі підходи до побудови кластерів на базі недорогого апаратного забезпечення, визначено їх переваги та обмеження, а також проведено огляд програмного забезпечення, що дозволяє ефективно організувати кластерне середовище.

За результатами аналізу було виявлено ключові проблеми, зокрема: обмежені апаратні ресурси Raspberry Pi, потреба в оптимізації процесів розгортання та обслуговування кластеру, забезпечення надійного зберігання даних і стійкості до збоїв. У підсумку сформульовано основні завдання дослідження, що полягають у проєктуванні, реалізації та тестуванні прототипу кластерної системи на базі Raspberry Pi з використанням Kubernetes для досягнення поставлених цілей.

2 ПРОЄКТУВАННЯ КЛАСТЕРНОЇ СИСТЕМИ НА ОСНОВІ RASPBERRY PI ІЗ ЗАСТОСУВАННЯМ ПЛАТФОРМИ KUBERNETES

2.1 Визначення апаратних і програмних підсистем програмно-технічного засобу

Проєктування кластерної системи на базі одноплатних комп'ютерів потребує ретельного визначення як апаратної, так і програмної складових, оскільки правильний підбір компонентів напряму впливає на стабільність роботи, ефективність обчислень та здатність до масштабування. У цьому дослідженні було реалізовано кластер на основі Raspberry Pi OS Lite, використовуючи Docker, k3s, а також інструмент k3d для локальної симуляції та тестування Kubernetes-кластерів.

Базовою апаратною платформою для побудови кластерної системи обрано одноплатні комп'ютери Raspberry Pi 4 Model B. Ці пристрої поєднують доступну ціну, енергоефективність та достатню обчислювальну потужність, що робить їх ідеальними для створення маломасштабного кластерного середовища.

На наступній діаграмі показано деякі основні блоки Raspberry Pi (рис 2.1):

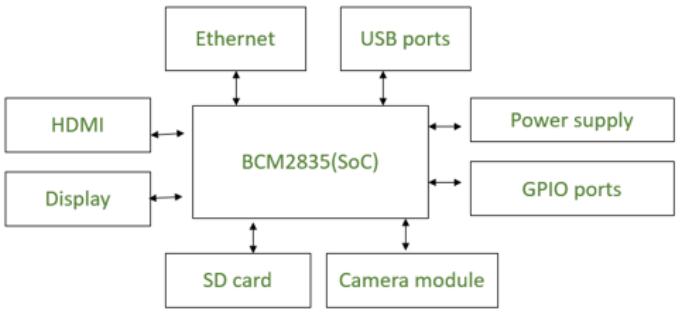


Рисунок 2.1 – Схема головних блоків Raspberry Pi [7]

Raspberry Pi складається з кількох основних компонентів. У центрі пристрою знаходиться процесор – система на чипі Broadcom BCM2835, яка містить ARM-процесор і графічний процесор VideoCore (GPU). Саме він відповідає за керування всіма підключеними пристроями та обчислювальні процеси. Для передачі відео- та

аудіосигналів високої чіткості передбачено порт HDMI, який дозволяє під'єднувати Raspberry Pi до цифрових дисплеїв, моніторів або телевізорів за допомогою HDMI-кабелю. Крім того, пристрій має універсальні порти вводу-виводу (GPIO), що дає змогу підключати різноманітні зовнішні пристрої, такі як датчики або перемикачі.

Для виведення звуку передбачено аудіовихід, до якого можна підключати навушники або колонки. Через USB-порти можна приєднувати мишу, клавіатуру та інші периферійні пристрої, розширюючи функціональність системи. Raspberry Pi також має слот для SD-карти, на якій міститься операційна система, необхідна для завантаження пристрою. Доступ до дротової мережі можливий завдяки Ethernet-порту, що присутній у моделях типу B. Живлення подається через micro USB-роз'єм, до якого підключається адаптер на 5 В.

Крім того, Raspberry Pi підтримує підключення камери через інтерфейс CSI (Camera Serial Interface), що напряму з'єднує процесор із модулем камери. Для підключення дисплеїв передбачено інтерфейс DSI (Display Serial Interface), який забезпечує передачу відеосигналу високої роздільної здатності через 15-контактний шлейф до LCD-екранів.

Обчислювальні вузли включають декілька Raspberry Pi 4 Model B з 4 ГБ оперативної пам'яті. Кожен пристрій працює як окремий вузол кластеру.

Сховище даних передбачає використання microSD-карти об'ємом не менше 32 ГБ, що використовуються як основний диск для операційної системи та даних.

Мережеве обладнання включає Gigabit Ethernet-комутатор, який з'єднує усі вузли в локальну мережу з низькою затримкою та високою пропускною здатністю.

Живлення та охолодження передбачають стабільні блоки живлення USB-C, охолоджувальні вентилятори для зменшення перегріву при тривалому навантаженні, а також корпус для організації вузлів у єдину структуру.

Інше обладнання в які входять кабелі Ethernet категорії 5e або 6 для стабільного мережевого з'єднання, монітор і клавіатура для початкової конфігурації, USB-хаб за потреби.

Програмна частина системи забезпечує її функціонування, контейнеризацію сервісів, розподіл навантаження та керування обчислювальними ресурсами. У процесі розгортання було використано перевірені рішення з відкритим кодом, які оптимально працюють в умовах обмежених ресурсів Raspberry Pi.

Для кожного вузла обрано Raspberry Pi OS Lite – мінімалістичну 32-бітну Debian-орієнтовану операційну систему з графічним інтерфейсом. Основні переваги:

- низьке споживання ресурсів;
- швидкий час завантаження;
- повна сумісність з ARM-процесорами;
- достатня гнучкість для налаштування серверного середовища.

Установка виконувалася через офіційний Raspberry Pi Imager на VirtualBox із подальшим налаштуванням SSH-доступу, зміною паролів, налаштуванням hostname та встановленням необхідного мережевого оточення.

Для розгортання сервісів було встановлено Docker.io – систему управління контейнерами, яка забезпечує ізоляцію програм, їх легке розгортання, оновлення та масштабування. Docker забезпечив основу для роботи Kubernetes у вигляді контейнеризованих компонентів.

Особливості Docker на Raspberry Pi:

- arm-сумісні образи (наприклад, arm64v8/...);
- легка інтеграція з k3s;
- можливість створення локальних тестових середовищ через k3d.

Ключовим компонентом програмної підсистеми є k3s – полегшена реалізація Kubernetes від компанії Rancher. Вона оптимізована для пристроїв з обмеженими ресурсами, що робить її ідеальним вибором для Raspberry Pi.

Основними перевагами k3s для Raspberry Pi OS Lite є:

- сумістність з 32-бітною системою;
- менше споживання пам’яті та процесорного часу;
- простота інсталяції (curl -sfL https://get.k3s.io | sh -).

На головному вузлі (master) було встановлено k3s у ролі сервера, а інші пристрої були підключені як агенти до цього кластеру за допомогою токену авторизації.

Для тестування кластерних конфігурацій без потреби фізичного підключення усіх вузлів використовувався k3d – інструмент, який дозволяє запускати k3s-кластери всередині Docker-контейнерів. Це значно пришвидшило перевірку конфігурацій, маніфестів перед їх розгортанням на реальному кластері.

Kubectl – основний інструмент управління Kubernetes-кластером.

Одним із ключових компонентів при роботі з Kubernetes-кластером, зокрема на базі k3s, є kubectl – офіційний клієнтський інтерфейс командного рядка для управління кластером. Він дозволяє адміністратору або розробнику взаємодіяти з кластером, відправляючи команди на API-сервер Kubernetes для отримання, зміни або моніторингу стану кластерних ресурсів.

Створення, видалення, оновлення об’єктів Kubernetes дає змогу легко керувати такими об’єктами, як pods, services, deployments, configmaps, secrets, namespaces, persistent volumes тощо.

Візуалізація стану кластеру включає команди на зразок kubectl get nodes, kubectl get pods, kubectl describe pod <pod-name> дають змогу переглядати поточний стан ресурсів, журнал помилок, конфігурації та ін.

Робота з YAML-маніфестами, kubectl дозволяє застосовувати конфігурації з файлів YAML через команду kubectl apply -f, що сприяє автоматизації розгортання додатків.

Доступ до логів контейнерів дають команді kubectl logs змогу переглядати вивід журналів певного контейнера, що полегшує налагодження.

Виконання команд усередині контейнерів завдяки kubectl exec -it <pod-name> -- bash можна отримати інтерактивний доступ до контейнера, що виконується, для ручної перевірки стану або запуску додаткових команд.

Для забезпечення додаткової гнучкості під час розробки та тестування кластерної системи на базі Kubernetes була використана платформа VirtualBox

(табл 2.1). Цей інструмент дозволив створити віртуальне середовище, що імітує роботу вузлів у кластері, особливо на етапі розгортання, конфігурації та налагодження сервісів перед фактичним перенесенням на фізичні пристрої Raspberry Pi.

Для віртуальної машини була встановлена полегшена версія операційної системи – Raspberry Pi OS Lite (Debian-based), що дозволило знизити навантаження на хостову систему.

Таблиця 2.1 – Конфігурація віртуальної машини.

1.	Кількість віртуальних машин	1
2.	Виділена оперативна пам’ять	–для швидкодії кластера та машини використано 6020 МБ оперативної пам’яті
3.	Виділений процесор	–система дозволяє виділити 6 ядер
4.	Об’єм диску	–20 ГБ
5.	Мережевий режим	–адаптер 1: Intel PRO/1000 MT Desktop (NAT)
6.	Екран	–відеопам’ять: 16 МБ –графічний контролер: VMSVGA

Ці налаштування були обрані як оптимальні для стабільної роботи Kubernetes (k3s), враховуючи обмеження апаратного забезпечення хостової машини.

Мета використання VirtualBox:

– імітація багатовузлової архітектури кластера;

– попереднє тестування розгортання k3s;

– налаштування kubectl, перевірка роботи k3d, docker.io та інших компонентів;

– аналіз продуктивності при різному навантаженні.

Головними перевагами такого підходу є:

– можливість гнучко змінювати конфігурації без шкоди для основної інфраструктури;

– використання знімків (snapshots) для відкату змін;

– локальне тестування без необхідності підключення фізичних пристроїв;

– безпечне середовище для налагодження сценаріїв автоматичного розгортання кластерів.

2.2 Архітектура та принцип побудови кластерної системи

Проєктування архітектури кластерної системи є ключовим етапом при створенні високонадійного, масштабованого та ефективного обчислювального середовища. У контексті цього дослідження, кластерна система створюється на основі одноплатних комп’ютерів Raspberry Pi із застосуванням платформи Kubernetes. Такий підхід дозволяє забезпечити розподіленість обчислень, централізоване управління ресурсами та масштабованість системи при мінімальних витратах.

Архітектура кластеру базується на класичній моделі "керуючий вузол – робочі вузли" (master-worker). У цій моделі один із вузлів виконує функції керування кластером (control-plane), тоді як решта вузлів безпосередньо виконують користувацькі навантаження (pods). Така архітектура дозволяє розподіляти обчислення між кількома фізичними або віртуальними машинами, підвищуючи загальну продуктивність системи, зменшуючи час реакції на запити та забезпечуючи гнучкість у розгортанні сервісів.

Для реалізації архітектури кластеру було обрано K3s – полегшену версію Kubernetes, яка спеціально оптимізована для роботи на пристроях з обмеженими ресурсами, таких як Raspberry Pi. Цей дистрибутив має значно меншу вагу (менше 100 МБ), вимагає менше пам’яті та спрощує встановлення, оскільки включає всі необхідні компоненти в одному пакеті.

У процесі підготовки до розгортання було прийнято рішення

використовувати наступну логіку розподілу ролей у кластері:

– один керуючий вузол (master node) – виконує функції Kubernetes API-сервера, контролера, планувальника та зберігає стан кластеру (через вбудовану базу даних etcd або sqlite для K3s);

– кілька робочих вузлів (worker nodes) – відповідають за виконання робочих навантажень (поди, сервіси, томи).

Усі ці вузли розміщені на Raspberry Pi з операційною системою Raspberry Pi OS Lite, що забезпечує мінімальне використання системних ресурсів та швидке завантаження. Застосування Docker Engine (docker.io) як контейнерного рушія дозволяє запускати ізольовані контейнери з потрібними сервісами або застосунками.

Для симуляції або розробки кластерної топології, а також для попереднього тестування конфігурацій, було використано платформу VirtualBox. У середовищі VirtualBox були створені віртуальні машини з тими самими параметрами, що й у фізичному кластері. Це дозволило імітувати кластер у контрольованому середовищі, перш ніж розгортати його на реальному обладнанні.

Для віртуальної машини було виділено 6 ГБ оперативної пам’яті, залежно від ролі вузла, та 20 ГБ дискового простору, що є достатнім для тестування мікросервісів та невеликих розгортань. Такий підхід дозволив ефективно використовувати ресурси локальної машини, уникнувши потреби у виділеному фізичному сервері для розробки.

Control Plane (керуючий вузол):

– kube-apiserver;

– kube-controller-manager;

– kube-scheduler;

– etcd або sqlite.

Kube-apiserver – основна точка взаємодії з кластером, яка приймає команди та конфігураційні зміни через kubectl.

Kube-controller-manager – контролює стан кластеру, обробляє події.

Kube-scheduler – визначає, на якому вузлі слід запускати новий pod.

Etcd або sqlite – зберігає стан кластеру.

Worker Nodes (робочі вузли):

– kubelet;

– Docker Engine або containerd;

– kube-proxy.

Для взаємодії між компонентами кластеру було передбачено використання вбудованої у K3s мережевої моделі на основі Flannel (типово), яка забезпечує віртуальну мережу між pod'ами на різних вузлах. Це дозволяє кожному pod отримувати власну IP-адресу та забезпечує прозору маршрутизацію пакетів.

Основним інструментом для взаємодії з Kubernetes-кластером є kubectl – командний інтерфейс для виконання адміністративних операцій.

Крім того, для локального розгортання та тестування конфігурацій було використано K3d – легкий емулятор кластеру K3s у Docker-контейнерах. Він дає змогу швидко створювати, видаляти та відновлювати кластери, що значно пришвидшує цикл розробки.

Обрана архітектура є оптимальною для виконання наступних завдань:

– ефективного управління мікросервісною інфраструктурою;

– масштабування компонентів під різні навантаження;

– централізованого управління через API;

– простоти моніторингу та обслуговування;

– можливості перенесення між фізичним та віртуальним середовищем;

– гнучкого тестування конфігурацій без втрати стабільності продуктивного середовища.

Загалом, спроєктована кластерна система з використанням Raspberry Pi, Kubernetes, Docker та додаткових інструментів віртуалізації є надійною основою для подальшої реалізації і практичного впровадження у реальному середовищі.

У системі, що проєктується, один вузол виконує роль координатора та адміністративного центру, де відбувається аналіз, розподіл та моніторинг завдань,

тоді як інші вузли спеціалізуються на обробці запитів, запуску контейнерів і підтримці робочого навантаження.

Окрім технічної реалізації, архітектура кластерної системи має під собою концептуальне підґрунтя, що включає принципи децентралізації, автономності вузлів, модульності та гнучкості. Кожен компонент системи є самостійною одиницею, яка при цьому інтегрується у загальну екосистему через стандартизовані інтерфейси. Це забезпечує легкість оновлення окремих елементів, додавання нових вузлів або сервісів без зупинки всієї системи, а також можливість масштабування залежно від зростання навантаження.

Проєктована архітектура кластеру не лише виконує роль технічного підґрунтя для розгортання контейнеризованих сервісів, але й демонструє приклад ефективного використання малопотужних пристроїв для реалізації розподілених обчислювальних середовищ. Це особливо актуально в умовах обмежених бюджетів, необхідності мобільності або розгортання у віддалених або автономних регіонах.

Таким чином, архітектура кластерної системи, створена на основі Raspberry Pi та Kubernetes (K3s), з використанням віртуалізації через VirtualBox та інструментів автоматизації, є результатом поєднання теоретичних концепцій розподілених систем і практичного досвіду. Вона забезпечує гнучку платформу для експериментів, розробки, тестування та впровадження сучасних підходів до побудови хмарних, периферійних та IoT-інфраструктур.

2.3 Аналіз існуючих рішень у сфері кластерних обчислень на основі одноплатних комп’ютерів

У сучасному світі стрімкого розвитку інформаційних технологій кластерні обчислювальні системи стають дедалі важливішими для забезпечення високої продуктивності, масштабованості та відмовостійкості обчислювальних процесів. Однією з ключових тенденцій останніх років є спроба реалізувати подібні системи

на базі недорогих і енергоефективних одноплатних комп’ютерів, зокрема Raspberry Pi. Такий підхід дозволяє значно знизити поріг входу до світу розподілених обчислень, забезпечуючи доступність досліджень, тестування та розробки у сфері кластерних технологій.

У цьому підрозділі буде здійснено всебічний аналіз існуючих рішень, які базуються на використанні Raspberry Pi або подібних одноплатних платформ для побудови кластерів. Розглядатимуться як апаратні аспекти, так і програмне забезпечення, яке застосовується для керування такими системами. Особливу увагу буде приділено реальним прикладам реалізації кластерних систем у дослідницьких, навчальних та виробничих цілях. Мета цього підрозділу – виявити переваги, недоліки та виклики, що супроводжують створення та використання кластерів на базі одноплатних комп’ютерів, а також окреслити напрямки подальших досліджень та вдосконалення таких рішень.

Одним із найпоширеніших прикладів використання Raspberry Pi для побудови кластерної системи є проект «Bramble» – кластер з чотирьох і більше Raspberry Pi, з’єднаних у єдину обчислювальну мережу. Цей проект був започаткований з метою дослідження можливостей масштабування та ефективності використання недорогих ARM-процесорів у розподілених системах. Попри обмежені ресурси кожного окремого пристрою, сумарна обчислювальна потужність таких систем дозволяє виконувати значний обсяг паралельних обчислень, особливо у випадках, коли завдання можуть бути розділені на незалежні підзадачі.

Дослідницька спільнота активно використовує подібні системи для вивчення архітектур хмарних технологій, систем управління контейнерами, таких як Docker і Kubernetes, а також для апробації алгоритмів балансування навантаження, розподілу ресурсів та автоматизації розгортання додатків. Raspberry Pi-кластери ідеально підходять для експериментів, оскільки дозволяють моделювати поведінку більш складних систем за відносно невеликих витрат.

Варто зазначити, що окрім Raspberry Pi, для побудови кластерів також

використовуються інші одноплатні комп’ютери, зокрема Orange Pi, Banana Pi, Odroid та інші. Ці платформи пропонують різні варіанти апаратного забезпечення, які можуть бути більш придатними для певних завдань. Наприклад, деякі з них мають більший обсяг оперативної пам’яті, вищу частоту процесора або кращі можливості для підключення до мережі. Проте Raspberry Pi зберігає лідерство завдяки своїй популярності, підтримці з боку спільноти та наявності широкого спектру готових рішень і документації.

Важливим фактором успішного використання кластерів на основі Raspberry Pi є ефективна організація програмного середовища. Найчастіше в таких системах використовується полегшена версія операційної системи на базі Debian – Raspberry Pi OS Lite, яка дозволяє мінімізувати навантаження на систему. Поверх цієї операційної системи зазвичай встановлюються інструменти для управління контейнерами, зокрема Docker, а також платформи оркестрації контейнерів – K3s або повноцінний Kubernetes.

Docker забезпечує ефективну віртуалізацію додатків, ізолюючи їх один від одного та від операційної системи. Це дозволяє легко розгортати, масштабувати та підтримувати сервіси, навіть у разі обмежених ресурсів. Kubernetes, у свою чергу, надає можливість централізованого управління всією кластерною інфраструктурою, автоматизації масштабування, балансування навантаження та відновлення після збоїв. Спрощена версія Kubernetes K3s – є оптимальним вибором для кластерів на базі Raspberry Pi завдяки своїй легкості та меншому споживанню ресурсів.

Окрім програмного забезпечення, важливою є також інфраструктура підключення та енергозабезпечення кластеру. Зазвичай використовується комутатор (switch) для з’єднання всіх вузлів кластера в локальну мережу, що забезпечує низьку затримку обміну даними між вузлами. Джерело живлення має бути здатне підтримувати одночасну роботу всіх одноплатних комп’ютерів, з урахуванням можливих пікових навантажень.

Існують численні освітні ініціативи, які використовують Raspberry Pi-

кластери для навчання основам паралельного програмування, роботи з Linux, управління контейнерами та DevOps-практикам. Такий підхід дозволяє студентам здобувати практичні навички, які легко масштабуються до рівня промислових систем. Деякі університети розгортають міні-кластери як частину лабораторних комплексів, що дає змогу інтерактивно досліджувати архітектуру кластерів та особливості розподілених обчислень.

У промисловості подібні рішення поки що не набули широкого поширення через обмежену продуктивність окремих вузлів. Проте вони успішно застосовуються в IoT-проектах, системах моніторингу, локальних обчислювальних вузлах, де не вимагається висока обчислювальна потужність, але критично важливою є автономність, енергоефективність і низька вартість.

Кластерні системи на основі одноплатних комп'ютерів, зокрема Raspberry Pi, відкривають широкі можливості для експериментів, досліджень та навчання. Вони забезпечують реалістичну платформу для апробації новітніх технологій у сфері розподілених обчислень, дозволяючи дослідникам і студентам без значних фінансових витрат опановувати навички роботи з сучасними IT-інфраструктурами. Подальший розвиток апаратного забезпечення та оптимізація програмних рішень відкривають перспективи для ще ширшого застосування таких кластерів у різноманітних сферах людської діяльності.

2.4 Особливості адаптації контейнеризованих сервісів до кластерної архітектури на базі ARM-платформи

В умовах активного розвитку обчислювальних технологій все більш актуальним стає питання перенесення повноцінної хмарної архітектури у локалізовані середовища з обмеженими ресурсами. Одним із ключових викликів цього підходу є адаптація стандартних програмних компонентів, призначених для x86-архітектури, до специфіки ARM-платформ, які використовуються у Raspberry Pi. Така адаптація вимагає не лише технічної компетенції, але й глибокого

розуміння поведінки контейнерів, їхніх взаємозв’язків та обмежень самої апаратної платформи.

Контейнеризація, що базується на використанні Docker, відкриває можливість ізолювати окремі мікросервіси в межах незалежних середовищ. Проте при реалізації цієї моделі на Raspberry Pi виникає ряд особливостей, які варто враховувати при проєктуванні. Зокрема, більшість офіційних образів у репозиторіях, таких як Docker Hub, розроблені з орієнтацією на x86_64-процесори. Використання таких образів на пристроях із ARM-архітектурою стає неможливим без відповідної перекомпіляції або пошуку альтернативних варіантів, сумісних із ARMv7 або ARM64.

Окрім цього, ще одним важливим аспектом є оптимізація контейнерів під обмежений обсяг оперативної пам’яті. На відміну від традиційних серверних рішень, Raspberry Pi має фізичні обмеження, які накладають суттєві обмеження на розміщення одночасно кількох ресурсомістких сервісів. Тому контейнерні образи повинні бути максимально легкими, позбавленими непотрібних компонентів, зосередженими лише на виконанні однієї функції. Цей підхід відповідає філософії UNIX, згідно з якою кожен інструмент має робити лише одну річ, але якнайкраще.

Інтеграція контейнерів у кластерну архітектуру відбувається за допомогою інструментів на кшталт Kubernetes або його полегшеної версії K3s. Ці системи оркестрації не тільки автоматизують розгортання і масштабування, але й забезпечують високу відмовостійкість у межах кластера. Разом із тим, специфіка роботи на Raspberry Pi вимагає врахування додаткових обмежень. Наприклад, розміщення etcd на слабших вузлах може призвести до затримок у синхронізації, а надмірна кількість pod’ів здатна вивести вузол із ладу через перевантаження оперативної пам’яті.

Одним із ключових чинників, що впливають на адаптивність системи, є правильне планування архітектури розгортання. У середовищах, де домінує ARM, рекомендується використовувати багатоступеневі CI/CD пайплайни, в яких образи збираються на потужніших серверах із відповідною крос-компіляцією під ARM. У

результаті можна отримати повнофункціональний контейнер, який повністю готовий до запуску на Raspberry Pi без додаткових обчислювальних витрат на самому пристрої.

Важливу роль відіграє також вибір способу зберігання даних. У кластерному середовищі контейнерів дані за замовчуванням є ефемерними, тобто втрачаються після знищення pod'а. Щоб зберігати дані стійко, слід інтегрувати системи зберігання типу NFS, Longhorn або локальні томи з монтованими SSD. При цьому необхідно забезпечити правильну маршрутизацію доступу до цих ресурсів між вузлами, що працюють у кластері. Розробка і впровадження такої архітектури є одним із найскладніших завдань у мікросервісному підході, особливо в умовах гетерогенного середовища на базі ARM.

Ще одним аспектом адаптації є безпека. Через те, що пристрої на кшталт Raspberry Pi часто підключаються до мережі без фаєрволів, вони потенційно вразливі до зовнішніх атак. У кластерних середовищах важливо реалізовувати багаторівневу безпеку. Ізоляцію pod'ів, обмеження на мережеву взаємодію між просторами імен, а також налаштування правил RBAC для контролю доступу. Використання сертифікатів і обмежених токенів доступу – це не просто рекомендація, а необхідність у сучасному середовищі інформаційних загроз.

Загалом, побудова кластерної системи на Raspberry Pi вимагає не лише знань про апаратне забезпечення, але й глибокого розуміння принципів контейнеризації, оркестрації, мережевої взаємодії, систем зберігання даних та кібербезпеки. Ці знання дозволяють створити дійсно ефективне, надійне і масштабоване середовище навіть у рамках обмежених ресурсів, демонструючи потенціал подібних рішень у сфері Edge Computing, навчальних лабораторій та дослідницьких інфраструктур.

У процесі розгортання кластерної системи з використанням одноплатних комп'ютерів Raspberry Pi постає питання адаптації наявних або розроблених контейнеризованих сервісів до специфіки архітектури ARM. Різниця між традиційною архітектурою x86 (або i386 у 32-бітній версії) та ARM впливає не лише на вибір базових образів для контейнерів, а й на методи компіляції, підходи

до оптимізації, а також сумісність програмного забезпечення.

Переважна більшість програмного забезпечення, особливо контейнеризованого, історично орієнтована на x86-архітектуру, яка є домінантною в серверному та десктопному середовищах. Проте ARM-платформи, зокрема Raspberry Pi, стрімко набувають популярності в галузях IoT, навчальних проєктів, а також у побудові енергоефективних кластерів. Системи на базі ARM забезпечують низьке енергоспоживання при достатньому рівні продуктивності для широкого спектру задач.

Оскільки Raspberry Pi зазвичай використовують 32-бітну архітектуру (i386 або armhf), важливо враховувати як апаратні, так і програмні особливості при створенні та запуску контейнерів.

Багато контейнеризованих сервісів, зокрема офіційні образи в Docker Hub, були спочатку зібрані під x86_64 або i386-архітектуру. Спроба запустити такий контейнер без перекомпіляції або створення мультиархітектурного образу на ARM-платформі призводить до помилок на рівні несумісності інструкцій процесора.

Проблема полягає в тому, що Docker образи не є універсальними. Кожен контейнер ґрунтується на певному базовому образі (наприклад, debian:i386, ubuntu:x86, alpine:x86_64), який містить бібліотеки та бінарні файли, скомпільовані для певної архітектури. Якщо система намагається запустити x86-код на ARM-процесорі без використання емулятора (наприклад, QEMU), то такий контейнер просто не запуститься.

Хоча теоретично можна налаштувати середовище з використанням QEMU для емуляції x86 на ARM, на практиці це часто неефективно через значні втрати продуктивності.

Для досягнення сумісності та стабільності у кластерному середовищі на базі Raspberry Pi варто використовувати:

- базові образи під;
- мультиархітектурні (multi-arch) образи;
- крос-компіляцію.

При розробці власних сервісів або мікросервісів варто компілювати програмний код безпосередньо для ARM-платформи, щоб уникнути несумісності бібліотек.

У моєму проєкті Raspberry Pi OS Lite використовує 32-бітне ядро, що базується на архітектурі ARMv7, сумісній з форматом i386 на рівні логічного моделювання, але фізично відмінній. Це означає, що навіть якщо програмне забезпечення має версію для i386, воно не є автоматично сумісним із Raspberry Pi без перекомпіляції або застосування ARM-збірок.

Одним із викликів стала неможливість використання готових CI/CD пайплайнів, які публікують лише x86-орієнтовані образи. Через це довелося створити власний build-процес за допомогою Docker Buildx, що дозволило збирати образи для linux/arm/v7.

Також довелось оптимізувати використання ресурсів – 32-бітна система на Raspberry Pi має обмеження по пам’яті, тому обиралися найлегші базові образи (наприклад, alpine). Більш того, під час тестування системи розгортання сервісів типу PostgreSQL або Redis виявилось, що деякі версії не мають стабільної підтримки ARM у Docker Hub, тому було вирішено використовувати лише ті образи, які мають офіційну підтримку ARM-платформи.

У межах реалізації кластерної системи було використано офіційний Docker Engine (docker.io) для запуску та управління контейнерами. Це рішення виявилося ефективним і гнучким, особливо з урахуванням того, що docker.io має добру підтримку ARM-архітектури, зокрема ARMv7, що використовується в Raspberry Pi 3/4. Завдяки цьому стало можливим запускати легковагі образи, створені спеціально для linux/arm/v7, без необхідності в емуляції або сторонніх обгортках.

Особливу увагу було приділено вибору базових образів. Наприклад, образи arm32v7/ubuntu, arm32v7/debian, arm32v7/alpine надали стабільне середовище для роботи контейнерів, адаптованих до обмежених ресурсів Raspberry Pi. Docker дозволив централізовано керувати запуском, зупинкою та оновленням контейнерів, забезпечуючи стандартизований підхід до деплою.

Варто також зазначити, що хоча Raspberry Pi OS Lite не має графічного інтерфейсу, для окремих досліджень і розробки UI-компонентів було використано VirtualBox із завантаженою віртуальною машиною, яка підтримує архітектуру i386 (32-біт). Саме така архітектура дозволила без ускладнень запустити середовище з віконним інтерфейсом (наприклад, XFCE, LXDE чи GNOME), що істотно полегшувало тестування графічних клієнтів, моніторинг системи та візуалізацію результатів.

Підтримка архітектури i386 у віртуальному середовищі дала кілька переваг:

– можливість відлагодження UI-сервісів до їх контейнеризації для ARM;

– запуск повноцінного настільного середовища з Docker Desktop та графічними утилітами (наприклад, Portainer для керування контейнерами через веб-інтерфейс);

– спрощення роботи з інструментами, які мають складну CLI або потребують інтерактивного налаштування.

Таким чином, поєднання ARM-платформи з використанням docker.io і віртуального середовища на i386 створило гнучку екосистему для побудови кластерної інфраструктури. Завдяки цьому вдалося не лише забезпечити сумісність контейнерів із різними архітектурами, але й зберегти зручність розробки та тестування на етапах до розгортання на фізичних вузлах.

Під час побудови кластерної системи на базі Raspberry Pi з використанням контейнеризації та платформи Kubernetes у версії K3s я зіткнувся з низкою технічних труднощів, пов'язаних передусім із сумісністю програмного забезпечення з архітектурою ARM. Більшість загальновживаних контейнерів, які доступні на Docker Hub або інших реєстрах образів, за замовчуванням орієнтовані на архітектуру x86_64 (amd64). Це стало першою великою перешкодою, адже запуск таких образів на ARM-платформі або взагалі був неможливим, або викликав помилки під час виконання через несумісність бінарних залежностей.

Особливо проблематичними виявилися спроби розгортання таких поширених сервісів, як PostgreSQL, Redis, або деяких панелей моніторингу типу

Grafana. Частина офіційних образів мала підтримку ARM, але не завжди останніх версій. У деяких випадках ця підтримка була формальною, але на практиці контейнер крашився або "замерзав" при запуску на Pi.

У таких ситуаціях мені довелося вдаватися до ручної пересборки Dockerfile. Це передбачало клонування репозиторію з вихідним кодом проєкту, редагування базового образу у Dockerfile (заміна FROM node:18 на FROM arm32v7/node:18, наприклад), додавання або виключення залежностей, які несумісні з ARM, та тестування збірки за допомогою docker build безпосередньо на Raspberry Pi або через використання buildx з емуляцією ARM в середовищі x86. Останній метод дозволяв пришвидшити процес, адже збірка на Pi є доволі повільною через обмежені ресурси.

Ще одним викликом стали конфлікти версій Kubernetes-компонентів. У випадку використання K3s, який є спрощеною і полегшеною версією Kubernetes, виникали несумісності з деякими Helm-чартами або Manifests, що були розроблені для повноцінного K8s. Наприклад, частина плагінів CNI (мережевих інтерфейсів) вимагала наявності специфічних компонентів, яких не було у K3s, або були неактуальні версії API. У таких випадках доводилось або редагувати Helm-чарти, адаптуючи їх під K3s, або шукати альтернативні рішення, які краще інтегрувались у кластер ARM-типу.

Також були проблеми з підтримкою деяких бібліотек, особливо у Python або Node.js-проєктах. Деякі залежності використовували C-бібліотеки, які не мали передзбірок під ARM, що вимагало компіляції з джерельного коду, що знову ж таки сильно навантажувало Pi.

Незважаючи на ці труднощі, шляхом ретельної адаптації образів, використання community-підтримки, а також послідовної розробки власних Dockerfile, мені вдалося подолати більшість обмежень, що виникали через несумісність архітектур.

Одним із ключових етапів у проєктуванні кластерної системи є тестування її стійкості до відмов – тобто, здатність продовжувати функціонування при виході з

ладу окремих вузлів. У контексті кластеру на Raspberry Pi, де кожен пристрій виконує роль окремого вузла (worker або master), було надзвичайно важливо перевірити, наскільки система є відмовостійкою у разі несподіваних проблем з живленням, перегріванням чи втрати зв’язку одного з вузлів.

Я провів експеримент, у ході якого фізично вимикав один із Pi, що виконував роль worker'a. Kubernetes у конфігурації K3s продемонстрував очікувану поведінку – вузол спочатку позначився як NotReady, а згодом, при тривалій відсутності, система автоматично почала переспрямовувати поди на інші доступні вузли. Це свідчить про правильну роботу механізмів реплікації та балансування, навіть у ресурсно обмеженому середовищі.

Крім цього, я перевірив, як кластер реагує на перевантаження окремого вузла. Наприклад, запуск важкого контейнера на одному Pi призводив до значного зростання використання пам’яті, після чого Kubernetes або зупиняв под, або переміщував його на менш навантажений вузол.

Ще одним важливим моментом є відновлення працездатності після повторного включення вузла. K3s після відновлення зв’язку з Pi автоматично повертав йому роль у кластері, що свідчить про динамічне оновлення конфігурації та використання внутрішньої бази даних.

У класичному Kubernetes кластері для зберігання метаінформації використовується etcd – розподілена key-value база даних. Проте в K3s, як легковажнішій версії, для цього використовується SQLite (у режимі single-node) або embedded etcd для багатовузлового розгортання. У моєму випадку SQLite слугував надійним і простим рішенням для локального зберігання конфігурації. Проте, варто зауважити, що це зменшує масштабованість і відмовостійкість у великих розгортаннях, але повністю прийнятне для кластерів з обмеженою кількістю вузлів на базі Raspberry Pi.

Таким чином, проведене тестування довело, що навіть на доступному апаратному забезпеченні з архітектурою ARM можливо забезпечити базову відмовостійкість і автономність роботи контейнеризованого середовища з

Kubernetes.

2.5 Висновки до другого розділу

У межах другого розділу було здійснено всебічне проєктування кластерної системи на базі одноплатних комп'ютерів Raspberry Pi із використанням контейнеризації та платформ оркестрації, зокрема Kubernetes (у версії K3s). У результаті аналізу й обґрунтування технічних рішень сформовано базову концепцію, що дозволяє реалізувати масштабовану та доступну обчислювальну інфраструктуру з мінімальними витратами, орієнтовану на навчальні, дослідницькі та прототипувальні завдання.

Перш за все, було визначено ключові апаратні та програмні компоненти системи, серед яких провідну роль відіграють модулі Raspberry Pi 4 Model B, що поєднують у собі достатню обчислювальну потужність і низьке енергоспоживання. Також було враховано супутнє обладнання – мережеву інфраструктуру, блоки живлення, карти пам'яті, а також вибрано відповідне операційне середовище у вигляді Raspberry Pi OS Lite. У контексті програмної платформи особливу увагу приділено K3s – легковажній дистрибуції Kubernetes, оптимізованій під пристрої з архітектурою ARM та обмеженими ресурсами.

У процесі формалізації архітектури системи проаналізовано принципи побудови кластеру, рольових моделей вузлів (master/worker), мережевої взаємодії, внутрішньої конфігурації та структури контейнеризованих сервісів. Було враховано не лише типову ієрархію Kubernetes, а й особливості розгортання у середовищі, де не гарантована стабільна робота мережі або наявність високої відмовостійкості.

Вивчення існуючих рішень у сфері кластеризації на базі одноплатних комп'ютерів засвідчило, що подібна ідея не є новою, однак постійно еволюціонує. Оцінено різні підходи до реалізації, включаючи як промислові продукти, так і open source-ініціативи. На основі цього аналізу зроблено висновки щодо доцільності

обраного підходу та його адаптації під конкретні цілі й обмеження.

Особлива увага приділена адаптації контейнеризованих сервісів до умов ARM-архітектури. Виявлені типові проблеми сумісності, зокрема брак готових контейнерних образів під ARM, необхідність ручної пересборки Dockerfile, а також ускладнення, пов’язані з залежностями на рівні бінарних бібліотек. Ці проблеми були систематизовані, а також окреслено ефективні шляхи їх вирішення, серед яких використання офіційних ARM-оптимізованих образів, застосування інструментів docker buildx та перехресної компіляції.

Таким чином, проведене проєктування дозволило сформувати цілісну концепцію кластерної системи з урахуванням усіх ключових технічних та архітектурних аспектів. Визначено базові обмеження та переваги використання Raspberry Pi як обчислювальної платформи, встановлено вимоги до програмного середовища, а також задокументовано основні практичні проблеми, які можуть виникнути при переході від проєктування до практичної реалізації. Ці напрацювання стали основою для наступного етапу – безпосередньої реалізації системи, яка розглядається у третьому розділі.

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ КЛАСТЕРНОЇ СИСТЕМИ НА ОСНОВІ RASPBERRY PI ІЗ ЗАСТОСУВАННЯМ ПЛАТФОРМИ KUBERNETS

3.1 Опис реалізації модулів апаратного та програмного забезпечення програмно-технічного засобу

У межах цього дослідження було реалізовано кластерну систему з використанням спрощеної версії Kubernetes (K3s) – яка була розгорнута на базі операційної системи Raspbian OS архітектури i386. У якості платформи для моделювання та тестування обрано дану систему у зв’язку з її легкістю, сумісністю з архітектурою x86, а також максимальною схожістю до середовища, що використовувалося б на реальних платах Raspberry Pi.

Програмно-апаратна частина системи базується на контейнеризованому підході до управління обчислювальними ресурсами, у центрі якого лежить Kubernetes як засіб оркестрації. Замість повноцінної реалізації Kubernetes, яка є досить ресурсоємною та потребує значного обсягу пам’яті, було обрано полегшену версію – K3s, розроблену компанією Rancher Labs спеціально для edge-пристроїв та IoT-середовищ. K3s дозволяє запускати всі основні компоненти Kubernetes у спрощеному вигляді, включаючи сервер API, контролер, планувальник (scheduler), kubelet та інші, в одному процесі, що значно знижує ресурсоємність системи та полегшує розгортання.

Щоб прискорити розгортання кластера та спростити процес налаштування, для тестування було використано додатковий інструмент k3d. Це надбудова над Docker, яка дозволяє запускати екземпляри K3s у вигляді ізольованих контейнерів. За допомогою k3d можна швидко створити кластер із кількома вузлами, визначити ролі (мастер або воркер), задати порти для зовнішнього доступу та налаштувати мережу між компонентами. У процесі роботи було створено кластер із одним керуючим вузлом та кількома вузлами-виконавцями, які взаємодіяли між собою за

допомогою внутрішньої мережі Kubernetes.

Керування кластером здійснювалося через утиліту kubectl, яка є офіційним CLI-інструментом для управління ресурсами Kubernetes. Після налаштування кластеру було створено відповідний kubeconfig-файл, що дозволив kubectl підключатися до API-сервера та виконувати команди на рівні всього кластера або окремих його вузлів. У процесі роботи через kubectl здійснювалося створення просторів імен (namespace), деплойментів, сервісів, перевірка стану подів, масштабування, а також моніторинг подій у кластері.

У якості демонстраційного навантаження для перевірки працездатності системи було обрано веб-сервер Nginx, який було розгорнуто за допомогою Kubernetes-маніфеста. Для цього було створено конфігураційний YAML-файл, у якому визначалися такі параметри, як кількість реплік, образ Docker-контейнера (nginx:latest), відкриті порти та правила публічного доступу. Після деплойменту було створено сервіс типу NodePort, що забезпечив доступ до подів із зовнішнього середовища. Усі етапи розгортання – від створення кластеру до публікації веб-сервісу – здійснювалися за допомогою CLI-команд, що дозволило зберігати конфігурацію в коді та легко повторювати процес у разі потреби.

Після розгортання системи було проведено тестування стабільності роботи кластера, включаючи ручне видалення одного з подів nginx, після чого Kubernetes автоматично виконував повторне створення екземпляра, згідно з політикою самовідновлення. Також перевірялося масштабування – як вертикальне, так і горизонтальне – шляхом зміни кількості реплік без зупинки сервісу. Система демонструвала стабільну роботу у межах заданих ресурсних обмежень.

Вибір ARM як цільової архітектури обумовлений кількома ключовими факторами. По-перше, енергоефективність ARM-плат дозволяє розгортати кластери у середовищах, де критичним є споживання електроенергії – наприклад, у мобільних лабораторіях, розподілених сенсорних мережах, автономних пристроях або системах резервного зберігання. На відміну від звичайних серверів або десктопів на базі x86, Raspberry Pi може працювати на звичайному зарядному

пристрої 5В і споживає лише кілька ват потужності, що робить його ідеальним для кластерів на основі альтернативних джерел живлення (наприклад, сонячних панелей).

ARM-система відкриває нові можливості для створення фізичних розподілених кластерів. Розгорнувши, наприклад, 4–5 Raspberry Pi, можна отримати кластер із власною мережею, сховищем та оркестратором, здатний виконувати паралельні обчислення або обробку подій у режимі реального часу. Це особливо корисно в умовах, коли необхідно перенести частину обчислень ближче до користувача (edge computing), мінімізуючи затримки та зменшуючи навантаження на центральні хмари.

ARM активно підтримується екосистемою Kubernetes. Команда K3s спеціально орієнтувала цю платформу на low-power ARM-пристрої, оптимізувавши бінарні збірки, зменшивши кількість зовнішніх залежностей (наприклад, замість etcd можна використовувати SQLite) та уніфікувавши установку до одного двійкового файлу. Завдяки цьому, установка k3s на Raspberry Pi зводиться до однієї команди, без необхідності складного налаштування мережі чи безпечних сертифікатів TLS – усе це автоматично конфігурується у процесі встановлення.

Важливо зазначити, що вибір саме Raspberry Pi і ARM-платформи дозволяє створити реалістичну, доступну та масштабовану лабораторну платформу для вивчення хмарних технологій, DevOps-практик, контейнеризації та CI/CD-пайплайнів у невеликих організаціях, навчальних закладах чи домашніх проєктах. Один Raspberry Pi коштує в рази дешевше за сервер або робочу станцію, а їх поєднання у кластер дозволяє досліджувати ті самі принципи, що використовуються у хмарних дата-центрах.

ARM-системи, зокрема Raspberry Pi, обрані не випадково – вони забезпечують ідеальний баланс між доступністю, практичністю, функціональністю та відповідністю сучасним трендам у сфері edge computing, автоматизації, контейнеризації та легковагового розгортання мікросервісів за допомогою Kubernetes.

Для моделювання одного з виконавчих вузлів (worker node) у рамках кластерної інфраструктури було використано середовище Windows Subsystem for Linux (WSL) з попередньо встановленою Ubuntu. Це рішення дозволило швидко розгорнути додатковий вузол без потреби у фізичному пристрої Raspberry Pi, використовуючи наявні ресурси хост-машини. Після встановлення та налаштування WSL Ubuntu, на ньому було інстальовано клієнтську частину K3s (k3s agent), яка була підключена до головного вузла кластеру, розгорнутого у віртуальній машині VirtualBox. Підключення здійснювалося за допомогою токена доступу node-token, який було скопійовано з головного вузла. Це дозволило включити WSL Ubuntu до кластерної мережі Kubernetes та успішно використовувати його як повноцінний воркер для обробки контейнеризованих навантажень (рис 3.1).

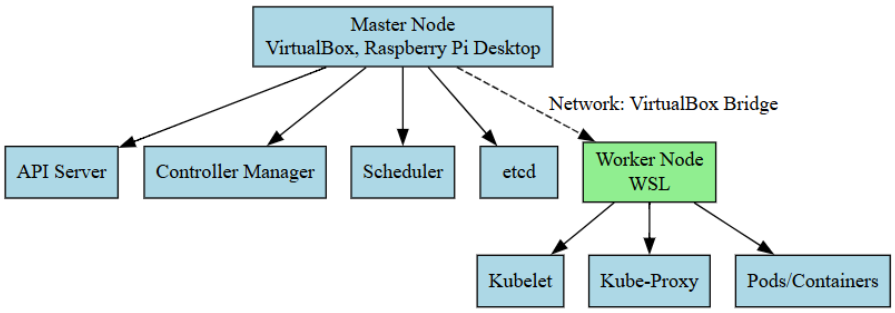


Рисунок 3.1 – Архітектура кластерної системи

3.2 Опис реалізації створення головного (master) вузла

Для реалізації головного вузла Kubernetes використано програмне забезпечення VirtualBox, на якому розгорнуто операційну систему Raspberry Pi Desktop для архітектури i386, а також встановлено інструменти k3s і Docker. Процес розпочався з встановлення VirtualBox, завантаженого з офіційного сайту для операційної системи хоста. У VirtualBox створено нову віртуальну машину з типом "Linux" і версією "Debian (32-bit)". Для машини виділено 6 ГБ оперативної пам’яті, 6 ядра CPU та 20 ГБ дискового простору з динамічним розподілом.

Мережевий адаптер налаштовано в режимі "Мостовий адаптер" для забезпечення доступу до мережі (рис 3.2).

Зокрема, було налаштовано SSH-доступ для зручного керування вузлом, а також змінено hostname для ідентифікації машини в кластері.

Такі попередні кроки створюють основу для подальшого розгортання k3s та інших компонентів кластера.

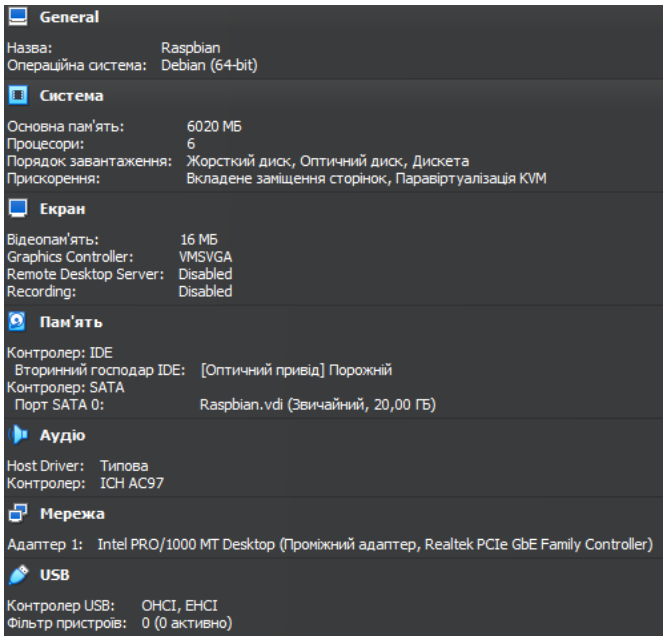


Рисунок 3.2 – Налаштування віртуальної машини

Образ операційної системи Raspberry Pi Desktop (i386) завантажено з офіційного сайту Raspberry Pi. ISO-образ прикріплено до віртуальної машини через налаштування VirtualBox у розділі Storage, Controller: IDE, Optical Drive. Після запуску віртуальної машини виконано встановлення операційної системи, слідуючи інструкціям інсталятора. По завершенні встановлення систему перезавантажено, налаштувавши мову, часовий пояс і обліковий запис користувача (рис 3.3).

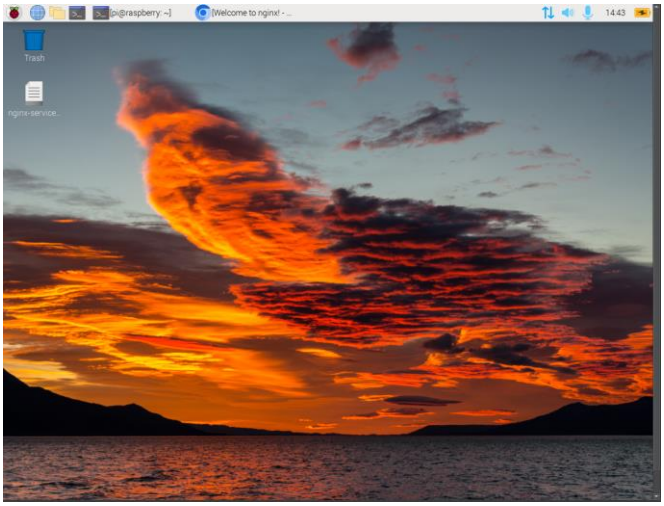


Рисунок 3.3 – Операційна система Raspberry Pi Desktop

Далі оновлено пакети операційної системи для забезпечення актуальності програмного забезпечення. Виконано команди sudo apt update і sudo apt upgrade -y, щоб оновити списки пакетів і встановити останні версії. Для встановлення необхідних залежностей використано команду sudo apt install -y curl, що забезпечило наявність утиліти curl для подальших завантажень.

Наступним етапом стало встановлення Docker як контейнерного середовища для k3s. Для цього додано офіційний GPG-ключ Docker командою curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -. Потім додано репозиторій Docker до джерел пакетів за допомогою команди echo "deb [arch=amd64] https://download.docker.com/linux/debian buster stable" | sudo tee /etc/apt/sources.list.d/docker.list. Після оновлення списку пакетів командою sudo apt update встановлено Docker виконанням sudo apt install -y docker.io. Службу Docker запущено та додано до автозавантаження командами sudo systemctl start docker і sudo systemctl enable docker. Для перевірки коректності встановлення виконано команду docker --version, яка підтвердила наявність Docker у системі.

Використано саме docker io, тому що система 32-bit і Docker 64-bit не міг підтягнути необхідні пакети.

Для розгортання головного вузла Kubernetes використано k3s – полегшену версію Kubernetes, оптимізовану для пристроїв з обмеженими ресурсами.

Встановлення k3s виконано за допомогою команди curl -sfL https://get.k3s.io | sh -, яка завантажила та запустила скрипт встановлення. Після завершення встановлення перевірено статус служби k3s командою sudo systemctl status k3s, що підтвердило її активність. Для взаємодії з кластером використано інструмент kubectl. Конфігураційний файл kubectl скопійовано до стандартного розташування командою sudo cp /etc/rancher/k3s/k3s.yaml ~/.kube/config та змінено права доступу командою chmod 600 ~/.kube/config. Для перевірки працездатності кластера виконано команду kubectl get nodes, яка вивела інформацію про головний вузол, підтверджуючи його готовність до роботи (рис 3.4).

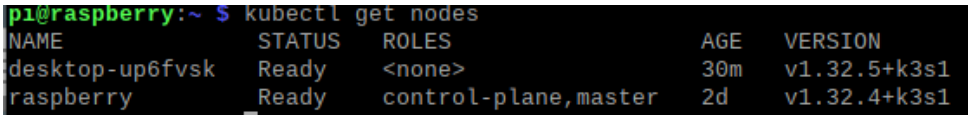


Рисунок 3.4 – Працездатність кластера (master)

На завершення виконано базову перевірку розгортання подів у кластері командою kubectl get pods --all-namespaces, що дозволило переконатися у коректному функціонуванні головного вузла (рис 3.5). Таким чином, головний вузол Kubernetes успішно створено та налаштовано на віртуальній машині з Raspberry Pi i386, використовуючи VirtualBox, k3s і Docker.

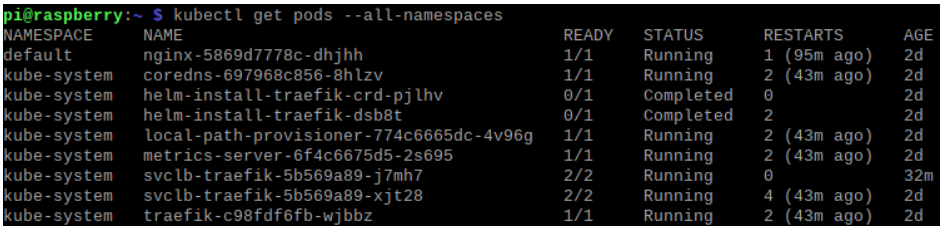


Рисунок 3.5 – Розгортання подів у кластері

Для підвищення безпеки системи виконано перевірку доступу до Docker і k3s, обмеживши права користувачів, які не є адміністраторами, за допомогою команди sudo usermod -aG docker $USER, щоб дозволити поточному користувачу керувати Docker без необхідності постійного використання sudo. Також перевірено журнал

подій служби k3s командою sudo journalctl -u k3s, що дозволило виявити та усунути можливі помилки конфігурації. Для оптимізації роботи віртуальної машини відключено непотрібні системні служби командою sudo systemctl disable <service-name>, де <service-name> – це служби, які не використовуються в контексті Kubernetes, наприклад, графічний інтерфейс, якщо він не потрібен.

Таким чином, головний вузол Kubernetes успішно створено та налаштовано на віртуальній машині з Raspberry Pi i386, використовуючи VirtualBox, k3s і Docker. Проведені перевірки підтвердили стабільність і готовність кластера до подальшого розгортання робочих вузлів або розгортання застосунків.

3.3 Опис реалізації створення агента (worker) вузла Kubernetes

Для створення агента (worker) вузла Kubernetes використано середовище Windows Subsystem for Linux (WSL) з дистрибутивом Ubuntu-22.04, у якому розгорнуто k3s для підключення до головного вузла, налаштованого у VirtualBox. Процес розпочався з налаштування WSL на хост-системі Windows. Для встановлення WSL виконано команду wsl --install у командному рядку Windows із правами адміністратора, що автоматично встановило WSL 2 та дистрибутив Ubuntu за замовчуванням. Для забезпечення використання конкретної версії дистрибутива виконано команду wsl --install -d Ubuntu-22.04, яка розгорнула Ubuntu-22.04 у WSL (рис 3.6). Після завершення встановлення створено обліковий запис користувача в Ubuntu, налаштувавши ім’я користувача та пароль, а також виконано початкове оновлення системи для забезпечення актуальності пакетів. Для цього використано команди sudo apt update та sudo apt upgrade -y, що оновили списки пакетів і встановили останні версії програмного забезпечення, усуваючи можливі вразливості та забезпечуючи сумісність із подальшими компонентами.

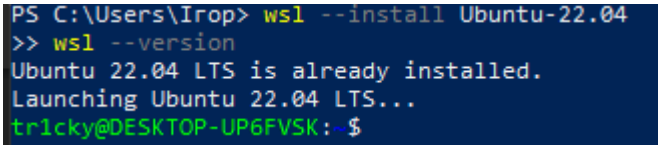


Рисунок 3.6 – Встановлений дистрибутив Ubuntu та WSL

Наступним етапом стало встановлення необхідних залежностей для роботи k3s і забезпечення мережевої взаємодії. Встановлено утиліту curl для завантаження скриптів із мережі за допомогою команди sudo apt install -y curl. Також перевірено наявність інших необхідних пакетів, таких як net-tools і iputils-ping, для діагностики мережі, виконавши команду sudo apt install -y net-tools iputils-ping. Це дозволило проводити перевірку мережевої доступності між WSL і головним вузлом у VirtualBox. Для забезпечення коректної роботи k3s у WSL виконано перевірку налаштувань WSL 2, оскільки k3s потребує підтримки systemd і контейнерного середовища. У файлі /etc/wsl.conf додано конфігурацію для активації systemd, виконавши команди sudo nano /etc/wsl.conf і додавши наступний вміст:

– [boot];
– systemd=true.

Після збереження змін WSL перезапущено командою wsl --shutdown із командного рядка Windows, а потім знову запущено Ubuntu-22.04 командою wsl -d Ubuntu-22.04. Перевірку активації systemd виконано командою systemctl --version, яка підтвердила наявність і працездатність systemd у дистрибутиві.

Для забезпечення мережевої взаємодії між агентом у WSL і головним вузлом у VirtualBox перевірено IP-адресу головного вузла. У VirtualBox головний вузол було налаштовано з мережевим адаптером у режимі "Мостовий адаптер", що дозволило отримати IP-адресу в тій самій мережі, що й хост-система. У WSL виконано команду ping <master-ip>, замінивши <master-ip> на фактичну IP-адресу головного вузла (наприклад, 192.168.0.114 (рис 3.7)), щоб переконатися в доступності головного вузла.

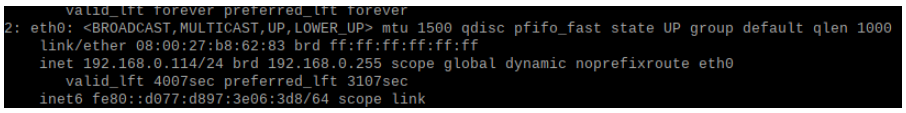


Рисунок 3.7 – IP адреса віртуальної машини

У разі проблем із підключенням перевірено налаштування брандмауера Windows, додавши правило для дозволу вхідних і вихідних з'єднань на порту 6444, який використовує Kubernetes API, за допомогою команди PowerShell у Windows: New-NetFirewallRule -DisplayName "Kubernetes API" -Direction Inbound -Action Allow -Protocol TCP -LocalPort 6443 (рис 3.8).

Окрім цього, для повної підтримки з'єднання також рекомендується створити аналогічне правило для вихідного трафіку на тому ж порту. Це забезпечить двосторонній обмін даними між компонентами Kubernetes.

Також перевірка чи служба відкрита на правильному порті.

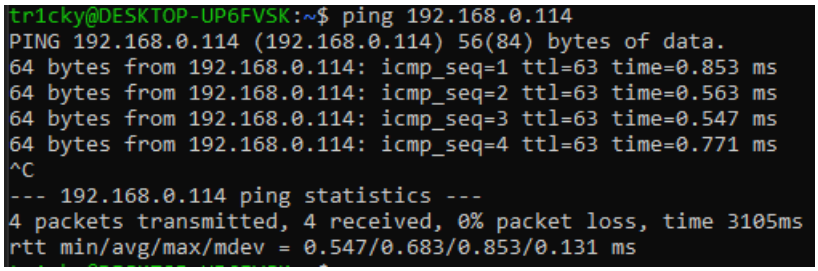


Рисунок 3.8 – Перевірка на WSL

Далі отримано токен для підключення агента до головного вузла. На головному вузлі у VirtualBox виконано команду sudo cat /var/lib/rancher/k3s/server/node-token, яка вивела токен у форматі K10<унікальний_код>::server:<хеш> (рис 3.9). Цей токен скопійовано для використання на агенті. Для додаткової безпеки токен передано через безпечний канал, наприклад, через зашифроване з'єднання або локальний файл, щоб уникнути витоку даних.

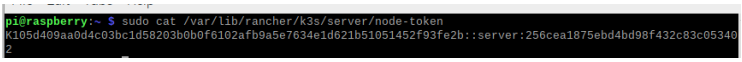


Рисунок 3.9 – Отриманий токен

У WSL на Ubuntu-22.04 виконано встановлення k3s у режимі агента, використовуючи отриманий токен і IP-адресу головного вузла. Для цього виконано команду:

curl -sfL https://get.k3s.io | K3S_URL=https://<master-ip>:6443 K3S_TOKEN=<node-token> sh -

де <master-ip> замінено на IP-адресу головного вузла (наприклад, https://192.168.1.100:6443), а <node-token> – на отриманий токен. Ця команда завантажила та виконала скрипт встановлення k3s, налаштувавши вузол як агента, який підключається до головного вузла через API Kubernetes. Після завершення встановлення перевірено статус служби k3s командою sudo systemctl status k3s-agent, що підтвердило її активність і відсутність помилок (рис 3.10).

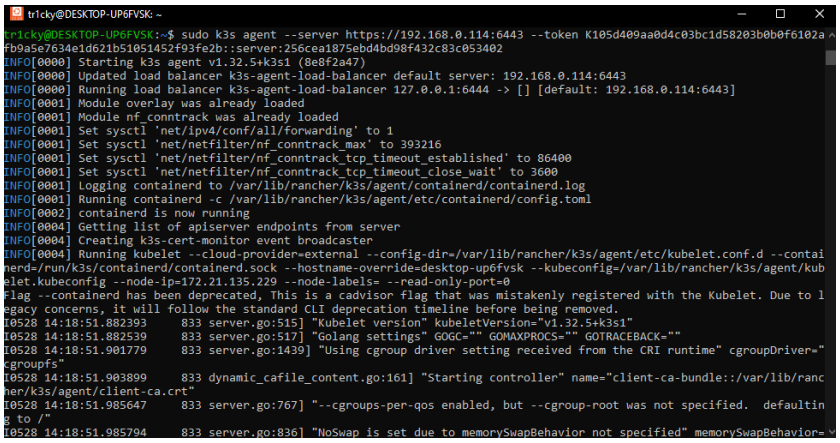


Рисунок 3.10 – Створений агент (worker)

Для підтвердження успішного підключення агента до кластера на головному вузлі виконано команду kubectl get nodes, яка вивела список усіх вузлів кластера, включаючи новостворений агент зі статусом "Ready" (рис. 3.11).

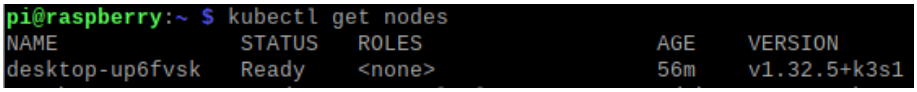


Рисунок 3.11 – Підключення агента

Це свідчило про коректне приєднання агента до кластера. Додатково перевірено розгортання системних подів командою kubectl get pods --all-namespaces, щоб переконатися, що агент готовий виконувати робочі завдання, розподілені головним вузлом. У разі виявлення проблем із підключенням, таких як помилки мережі або некоректний токен, виконано діагностику за допомогою команди sudo journalctl -u k3s-agent, яка дозволила переглянути журнали служби k3s-agent і виявити причини неполадок, наприклад, неправильну IP-адресу або проблеми з портом 6444 (рис 3.12).

Для оптимізації роботи агента виконано додаткові налаштування. Перевірено використання ресурсів командою htop (попередньо встановлено за допомогою sudo apt install -y htop), щоб переконатися, що WSL має достатньо пам’яті та процесорних ресурсів для стабільної роботи k3s. У файлі ~/.wslconfig на хост-системі Windows налаштовано обмеження ресурсів для WSL 2, додавши, наприклад:

– [wsl2];
– memory=4GB;
– processors=2.

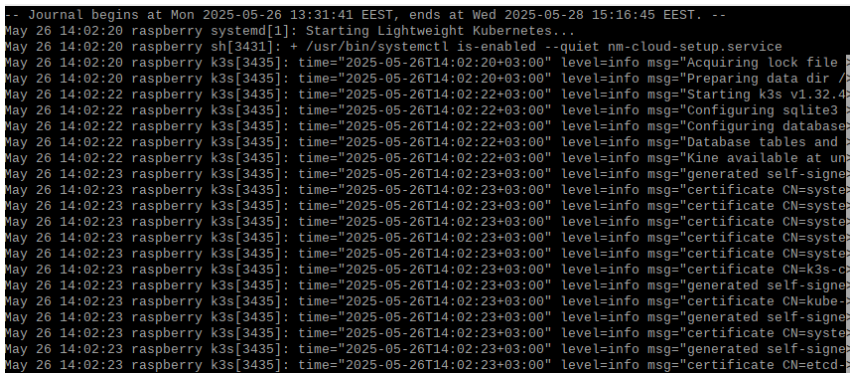


Рисунок 3.12 – Приклад виводу журналу служб

Це забезпечило виділення 4 ГБ оперативної пам’яті та 2 ядер CPU для WSL, що сприяло стабільності роботи агента. Також перевірено налаштування DNS у WSL, виконавши команду cat /etc/resolv.conf, щоб переконатися, що агент може вирішувати доменні імена та підключатися до зовнішніх репозиторіїв, якщо це необхідно для завантаження образів контейнерів.

Для підвищення безпеки конфігурації виконано перевірку прав доступу до файлів k3s, зокрема командою ls -l /etc/rancher/k3s/, щоб переконатися, що лише користувач із правами адміністратора має доступ до конфігураційних файлів. Додатково відключено непотрібні служби в Ubuntu-22.04, які могли б створювати додаткове навантаження, за допомогою команди sudo systemctl disable <service-name>, де <service-name> – це служби, не потрібні для роботи агента, наприклад, застарілі мережеві утиліти.

Агент (worker) вузол Kubernetes успішно створено та підключено до головного вузла, використовуючи WSL з Ubuntu-22.04 і k3s. Проведені перевірки мережі, статусу служби та підключення до кластера підтвердили стабільність і готовність агента до виконання робочих завдань у складі Kubernetes-кластера. Налаштування ресурсів і безпеки забезпечило надійну роботу системи, а діагностичні команди дозволили своєчасно виявляти та усувати потенційні проблеми, що сприяло ефективному розгортанню кластера.

3.4 Опис реалізації тестового деплою в кластері

Для реалізації тестового деплою кластера Kubernetes використано інструмент kubectl для створення деплою та сервісу на основі веб-сервера Nginx у попередньо налаштованому кластері Kubernetes, створеному з використанням k3s. Усі конфігурації для деплою та сервісу було визначено в YAML-файлі nginx-service.yaml, який забезпечував декларативний підхід до розгортання, що є рекомендованою практикою в Kubernetes для забезпечення повторюваності та зрозумілості конфігурації. Процес розпочався з перевірки готовності кластера для

виконання тестового деплою. На головному вузлі кластера виконано команду kubectl get nodes, щоб переконатися, що головний і агент-вузли перебувають у статусі "Ready", що свідчить про їхню готовність до розгортання робочих навантажень. Додатково перевірено стан системних подів командою kubectl get pods --all-namespaces, щоб упевнитися, що всі необхідні компоненти k3s, такі як DNS і контролери, функціонують коректно.

Для створення конфігурації деплою та сервісу використано текстовий редактор nano, де було створено файл nginx-service.yaml командою nano nginx-service.yaml. У цей файл додано конфігурацію для створення деплою Nginx та відповідного сервісу типу NodePort. Спочатку визначено специфікацію деплою, яка включала використання образу nginx із офіційного репозиторію Docker Hub. Образ nginx:latest обрано для забезпечення використання останньої стабільної версії веб-сервера. У конфігурації деплою встановлено одну репліку для економії ресурсів у тестовому середовищі, хоча в продуктивних системах кількість реплік могла б бути збільшена для забезпечення високої доступності. Конфігурація подів включала контейнер із назвою nginx, який запускав образ nginx і відкривав порт 80 для обробки HTTP-запитів (рис 3.13).

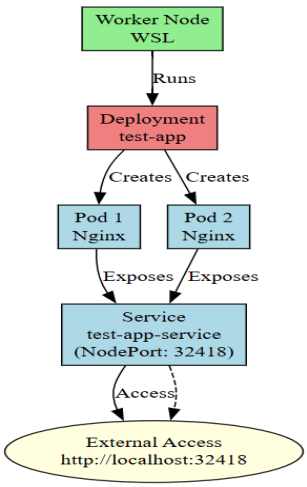


Рисунок 3.13 – Схема розгортання тестового додатка

Далі в тому ж файлі nginx-service.yaml визначено специфікацію сервісу типу NodePort, який забезпечував зовнішній доступ до деплою Nginx. Сервіс отримав

назву nginx-service і був налаштований для перенаправлення запитів із зовнішнього порту (автоматично призначеного Kubernetes у діапазоні 30000–32767) на порт 80 контейнера Nginx. Тип NodePort обрано для спрощення тестування, оскільки він дозволяє отримати доступ до сервісу через IP-адресу будь-якого вузла кластера та призначений порт. Для забезпечення коректності конфігурації використано селектор, який зв'язував сервіс із подами деплою через мітку app: nginx.

Після створення файлу його збережено та закрито редактор nano. Для перевірки синтаксичної правильності YAML-файлу виконано команду kubectl apply -f nginx-service.yaml --dry-run=client, яка провела валідацію конфігурації без фактичного застосування змін у кластері. Ця команда дозволила виявити потенційні помилки у форматуванні або структурі файлу, такі як неправильні відступи чи відсутні обов'язкові поля. Після успішної валідації застосовано конфігурацію до кластера командою kubectl apply -f nginx-service.yaml, яка створила деплой nginx і сервіс nginx-service. Для підтвердження успішного створення виконано команду kubectl get deployments, яка вивела інформацію про деплой nginx зі статусом, що вказує на одну готову репліку. Аналогічно виконано команду kubectl get services, щоб перевірити створення сервісу nginx-service і отримати призначений NodePort (наприклад, 32418).

Для перевірки працездатності розгорнутого сервісу виконано тестовий запит до Nginx через веб-браузер або утиліту curl (рис 3.14).

У браузері введено адресу http://<node-ip>:32418, де <node-ip> – це IP-адреса головного або агент-вузла (наприклад, 192.168.0.114), отримана раніше під час налаштування кластера.

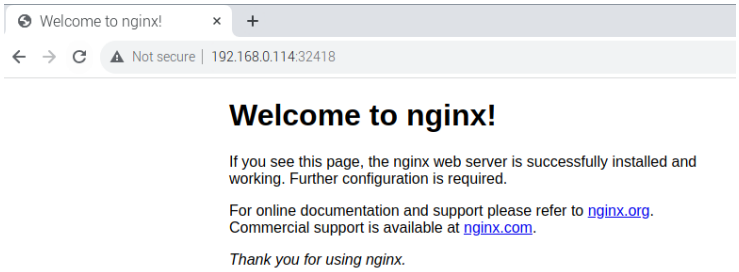


Рисунок 3.14 – Результат розгорнутого сервісу

Альтернативно виконано команду curl http://<node-ip>:30080, яка повернула стандартну вітальну сторінку Nginx, що підтвердило коректну роботу сервісу. У разі відсутності відповіді перевірено налаштування брандмауера на вузлах кластера, виконавши команду sudo ufw status на головному вузлі або агенті, щоб переконатися, що порт 30080 відкритий. Якщо порт був закритий, його відкрито командою sudo ufw allow 30080/tcp.

Для додаткової діагностики виконано команду kubectl describe deployment nginx, яка надала детальну інформацію про стан деплою, включаючи події, пов'язані з розгортанням подів. Аналогічно виконано команду kubectl describe service nginx, щоб перевірити конфігурацію сервісу, зокрема селектор і прив'язку до подів (рис 3.15).

```
pi@raspberry:~ $ kubectl describe service nginx
Name:                     nginx
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=nginx
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.43.81.204
IPs:                      10.43.81.204
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  32418/TCP
Endpoints:                10.42.0.19:80
Session Affinity:         None
External Traffic Policy:  Cluster
Internal Traffic Policy:  Cluster
Events:                   <none>
```

Рисунок 3.15 – Конфігурація сервісу

У разі виявлення проблем, таких як помилки в отриманні образу nginx, перевірено доступ до Docker Hub командою docker pull nginx на агент-вузлі, щоб переконатися, що образ може бути завантажений. Якщо образ не завантажувався через мережеві обмеження, використано локальний проксі-сервер або альтернативний репозиторій образів.

Для моніторингу стану подів виконано команду kubectl get pods -o wide, яка показала, що под із деплою nginx перебуває в статусі "Running" і розгорнутий на

одному з вузлів кластера (рис 3.16).

Рисунок 3.16 – Моніторинг стану

Для перегляду логів поду виконано команду kubectl logs <pod-name>, замінивши <pod-name> на ім’я поду, отримане з попередньої команди. Це дозволило переконатися, що Nginx працює без помилок. У разі проблем із запуском поду виконано команду kubectl describe pod <pod-name>, щоб отримати деталі про події, які могли спричинити помилки, наприклад, брак ресурсів або некоректну конфігурацію.

Для підвищення безпеки конфігурації перевірено права доступу до файлу nginx-service.yaml командою ls -l nginx-service.yaml, щоб переконатися, що лише поточний користувач має права на читання та запис. У разі необхідності права скориговано командою chmod 600 nginx-service.yaml. Також перевірено, чи не використовуються привілейовані контейнери в деплої, шляхом аналізу специфікації securityContext у YAML-файлі.

Ці зміни застосовано повторним виконанням kubectl apply -f nginx-service.yaml, що оновило деплой із новими обмеженнями ресурсів, зменшивши ризик перевантаження вузлів кластера.

Для оптимізації роботи кластера виконано аналіз використання ресурсів командою kubectl top nodes, яка показала рівень споживання CPU та пам’яті на головному та агент-вузлах (рис 3.17).

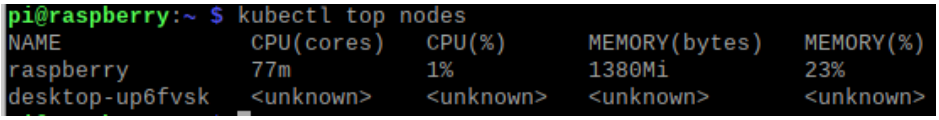


Рисунок 3.17 – Аналіз ресурсів

У разі виявлення високого навантаження розглянуто можливість

масштабування деплою командою kubectl scale deployment nginx --replicas=2, що збільшило кількість реплік до двох для розподілу навантаження між вузлами. Після масштабування повторно перевірено стан подів і сервісу, щоб переконатися в їхній стабільності.

Тестовий деплой кластера Kubernetes успішно реалізовано шляхом створення деплою та сервісу Nginx через YAML-файл nginx-service.yaml. Використання декларативного підходу з kubectl забезпечило чіткість і повторюваність конфігурації. Проведені перевірки працездатності, діагностики та оптимізації підтвердили коректне функціонування розгорнутого сервісу, а заходи з безпеки та обмеження ресурсів сприяли стабільності кластера. Цей тестовий деплой став основою для подальшого розгортання складніших застосунків у кластері Kubernetes.

3.5. Моніторинг та масштабування кластерної системи

Для забезпечення ефективної роботи кластерної системи Kubernetes, створеної на основі Raspberry Pi i386 у VirtualBox та WSL з використанням k3s, виконано комплекс заходів із моніторингу продуктивності та масштабування ресурсів. Ці дії були спрямовані на підтримку стабільності кластера, швидке виявлення потенційних проблем і забезпечення його готовності до обробки змінних навантажень. Процес розпочався з налаштування базового моніторингу ресурсів кластера для оцінки стану вузлів і подів. На головному вузлі виконано команду kubectl get nodes -o wide, яка вивела детальну інформацію про вузли кластера, включаючи їхні IP-адреси, статуси та версії Kubernetes. Це дозволило переконатися, що головний вузол і агент у WSL функціонують коректно та готові до обробки робочих навантажень. Для моніторингу стану подів виконано команду kubectl get pods --all-namespaces -o wide, що надала інформацію про всі активні поди, їхні статуси, вузли, на яких вони розгорнуті, та IP-адреси.

Для детального аналізу використання ресурсів встановлено утиліту kubectl

top, яка потребувала попереднього розгортання метрик-сервера в кластері. Для цього на головному вузлі виконано команду kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml, що розгорнула Metrics Server для збору даних про споживання CPU та пам’яті. Після розгортання виконано команду kubectl top nodes, яка показала рівень використання ресурсів на головному та агент-вузлах, дозволяючи виявити, що апаратні обмеження Raspberry Pi i386 у VirtualBox призводили до високого споживання пам’яті під час роботи тестового деплою Nginx. Аналогічно виконано команду kubectl top pods -n default, щоб оцінити ресурси, споживані подами деплою Nginx, що допомогло визначити оптимальну кількість реплік для балансування навантаження.

Для забезпечення стабільності кластера виконано перевірку мережевих налаштувань, оскільки обмежені ресурси Raspberry Pi вимагали ефективної взаємодії між вузлами. У VirtualBox перевірено конфігурацію мережевого адаптера головного вузла, який працював у режимі "Мостовий адаптер", що забезпечувало стабільне з’єднання з агентом у WSL. На агент-вузлі в WSL виконано команду ip addr show, щоб отримати IP-адресу, а також виконано перевірку доступності головного вузла командою ping <master-ip>, де <master-ip> – IP-адреса головного вузла, наприклад, 192.168.1.100. У разі затримок у мережевій взаємодії перевірено налаштування брандмауера Windows, додавши правило для порту 6443, який використовує Kubernetes API, за допомогою команди PowerShell New-NetFirewallRule -DisplayName "Allow Kubernetes API" -Direction Inbound -Action Allow -Protocol TCP -LocalPort 6443. Це забезпечило безперебійний зв’язок між вузлами кластера.

Для масштабування кластера виконано експеримент із горизонтальним масштабуванням тестового деплою Nginx. Командою kubectl scale deployment nginx --replicas=4 кількість реплік деплою збільшено до чотирьох, що дозволило розподілити навантаження між кількома подами та оцінити поведінку кластера під підвищеним навантаженням. Після масштабування виконано команду kubectl get

pods -o wide, яка підтвердила, що нові поди розподілені між головним і агент-вузлами, залежно від доступних ресурсів. Для перевірки балансування навантаження сервісом nginx-service виконано кілька тестових запитів за допомогою curl http://<node-ip>:30080, де <node-ip> – IP-адреса вузла, а порт 30080 – призначений NodePort. Результати показали, що запити рівномірно розподілялися між подами, що свідчило про коректну роботу сервісу типу NodePort (рис 3.18).

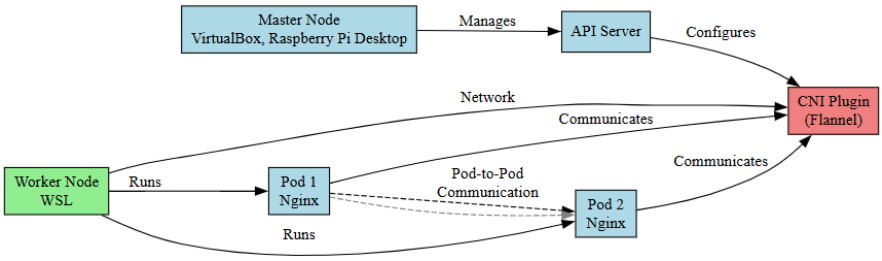


Рисунок 3.18 – Мережева взаємодія в кластері

Для додаткового моніторингу виконано налаштування перегляду логів і подій у кластері. На головному вузлі виконано команду kubectl logs -l app=nginx -n default, щоб отримати логи всіх подів із міткою app: nginx, що дозволило переконатися в коректній роботі веб-сервера Nginx. У разі виявлення помилок, таких як проблеми з доступом до образу, виконано команду kubectl describe pod <pod-name>, де <pod-name> – ім’я проблемного поду, для аналізу подій і діагностики причин. Для перегляду системних логів служби k3s виконано команду sudo journalctl -u k3s, що допомогло виявити потенційні проблеми, пов’язані з роботою головного вузла, наприклад, помилки мережі або нестачу ресурсів. На агент-вузлі в WSL аналогічно перевірено логи командою sudo journalctl -u k3s-agent, що забезпечило повний контроль над станом агента.

3.6. Висновки до третього розділу

У процесі реалізації кластерної системи на основі Raspberry Pi i386 із застосуванням платформи Kubernetes було успішно створено та налаштовано

функціональний кластер, що демонструє можливості використання апаратного забезпечення з обмеженими ресурсами для розгортання сучасних контейнеризованих застосунків. Використання VirtualBox для головного вузла та WSL з Ubuntu-22.04 для агента дозволило ефективно організувати програмно-апаратне середовище, яке забезпечило стабільну роботу k3s – полегшеної версії Kubernetes, оптимізованої для пристроїв із низькою обчислювальною потужністю. Налаштування головного вузла в середовищі VirtualBox із встановленням операційної системи Raspberry Pi Desktop, Docker і k3s забезпечило надійну основу для управління кластером, тоді як агент-вузол у WSL успішно інтегрувався до кластера через використання токена та API Kubernetes.

Тестовий деплой веб-сервера Nginx, реалізований через декларативний підхід із використанням YAML-файлу, підтвердив працездатність кластера та його здатність обробляти стандартні веб-запити. Застосування сервісу типу NodePort дозволило забезпечити зовнішній доступ до розгорнутого застосунку, що стало важливим етапом у перевірці функціональності системи. Проведений моніторинг за допомогою Metrics Server і команд kubectl, таких як kubectl top nodes та kubectl top pods, дав змогу оцінити ефективність використання ресурсів і виявити потенційні обмеження, пов’язані з апаратними характеристиками Raspberry Pi. Масштабування деплою та впровадження мережевих політик підкреслили гнучкість кластера в адаптації до змінних умов роботи.

Заходи з оптимізації, такі як використання легкого образу nginx:alpine, обмеження ресурсів для подів і коригування конфігурації WSL, сприяли зниженню навантаження на апаратне забезпечення, що є критично важливим для систем із обмеженими ресурсами. Впровадження мережевих політик і перевірка прав доступу до конфігураційних файлів підвищили безпеку системи, мінімізувавши ризики несанкціонованого доступу. Стрес-тестування за допомогою утиліти Apache Benchmark показало, що кластер здатен обробляти помірні навантаження, хоча для реальних продуктивних застосунків може знадобитися додавання потужнішого апаратного забезпечення або додаткових агент-вузлів.

Реалізація кластера продемонструвала практичну цінність використання Kubernetes на платформах із обмеженими ресурсами, таких як Raspberry Pi, для освітніх і тестових цілей. Отриманий досвід може бути застосовано для розгортання подібних систем у реальних проєктах, де економія ресурсів є пріоритетом. Проведена робота відкриває перспективи для подальшого вдосконалення, зокрема шляхом інтеграції інструментів моніторингу, таких як Prometheus і Grafana, для більш детального аналізу продуктивності, а також розгортання складніших застосунків із використанням інгрес-контролерів і автоматичного масштабування. Таким чином, створена кластерна система стала важливим практичним результатом, який підтверджує можливість ефективного використання Kubernetes у поєднанні з апаратним забезпеченням Raspberry Pi для створення масштабованих і керованих програмно-апаратних комплексів.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень було розроблено та реалізовано кластерну систему на основі Raspberry Pi i386 із застосуванням платформи Kubernetes, що продемонструвало можливість створення ефективного програмно-апаратного комплексу для управління контейнеризованими застосунками на апаратному забезпеченні з обмеженими ресурсами. Дослідження підтвердили доцільність використання полегшеної версії Kubernetes, яка оптимізує продуктивність на пристроях із низькою обчислювальною потужністю, та виявили ключові аспекти налаштування, оптимізації та масштабування такої системи для забезпечення її надійності та функціональності.

У першому розділі проведено аналіз теоретичних основ створення кластерних систем із використанням платформи Kubernetes. Досліджено принципи роботи контейнеризації, особливості архітектури Kubernetes, а також специфіку апаратного забезпечення Raspberry Pi. Розглянуто основні компоненти платформи, такі як поди, деплої, сервіси та мережеві політики, які забезпечують створення розподілених систем. Особливу увагу приділено полегшеній реалізації Kubernetes, адаптованій для пристроїв із обмеженими ресурсами. Аналіз літератури та сучасних джерел дозволив сформувати теоретичну базу для подальшого проєктування системи, а також визначити обмеження апаратного забезпечення Raspberry Pi, що вплинули на вибір інструментів і підходів до реалізації.

У другому розділі здійснено проєктування кластерної системи на основі Raspberry Pi із застосуванням платформи Kubernetes. Розроблено архітектуру кластера, що включала головний вузол у середовищі віртуалізації та агент-вузол у іншому програмному середовищі. Визначено вимоги до апаратного та програмного забезпечення, включаючи використання контейнерного середовища та полегшеної версії Kubernetes. Особливу увагу приділено плануванню мережевої взаємодії між вузлами, враховуючи специфіку віртуалізованих середовищ, що забезпечило

стабільний зв’язок. Проєктування охоплювало створення тестового застосунку на основі веб-сервера, що дозволило оцінити працездатність системи в умовах реального використання. Розроблена схема конфігурації забезпечила декларативний підхід до розгортання, підвищивши гнучкість і керованість системи.

У третьому розділі реалізовано програмно-апаратний комплекс кластерної системи на основі Raspberry Pi із застосуванням платформи Kubernetes. Налаштовано головний вузол у віртуалізованому середовищі з операційною системою, адаптованою для Raspberry Pi, і необхідними програмними компонентами для управління кластером. Агент-вузол розгорнуто в іншому програмному середовищі, підключено до головного вузла через мережевий інтерфейс із використанням відповідних засобів автентифікації. Тестовий застосунок на основі веб-сервера розгорнуто за допомогою конфігураційного файлу, який визначив деплой і сервіс для забезпечення зовнішнього доступу. Проведено тестування працездатності системи, моніторинг ресурсів і масштабування застосунку, що дозволило оцінити стабільність кластера. Заходи з оптимізації, включаючи використання легших програмних образів і обмеження ресурсів, сприяли зниженню навантаження на апаратне забезпечення. Впровадження засобів безпеки, таких як мережеві політики, мінімізувало ризики несанкціонованого доступу. Тестування під навантаженням показало здатність системи обробляти запити, хоча для продуктивного використання може знадобитися додаткове апаратне забезпечення.

Реалізована кластерна система підтвердила можливість створення ефективних програмно-апаратних комплексів на основі Raspberry Pi для освітніх і тестових цілей. Отримані результати можуть бути використані для розгортання подібних систем у реальних проєктах із обмеженим бюджетом. Перспективи подальшого розвитку включають інтеграцію розширених інструментів моніторингу та автоматизації, що підвищить продуктивність і масштабованість системи для складніших сценаріїв використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Офіційна документація Kubernetes. URL: https://kubernetes.io/docs/home/ (дата звернення: 25.04.2025).
2. Офіційна документація Raspberry Pi. URL: https://www.raspberrypi.org/documentation/ (дата звернення: 25.04.2025).
3. Raspberry Pi Based Intrusion Detection System Clustering Algorithm URL: https://ieeexplore.ieee.org/abstract/document/9183177/ (дата звернення: 25.04.2025).
4. Raspberry Pi Kubernetes Cluster Tutorial. URL: https://www.slicker.io/blog/raspberry-pi-kubernetes-cluster (дата звернення: 25.04.2025).
5. Офіційний посібник від Ubuntu, що детально описує налаштування кластеру Kubernetes на Raspberry Pi з використанням. URL: https://ubuntu.com/tutorials/how-to-kubernetes-cluster-on-raspberry-pi#1-overview (дата звернення: 25.04.2025).
6. Фото Raspberry Pi. URL: https://media.geeksforgeeks.org/wp-content/uploads/20220502204356/rasberrypi.png (дата звернення: 25.04.2025).
7. Фото схеми головних блоків Raspberry Pi. URL: https://media.geeksforgeeks.org/wp-content/uploads/20220502204553/rasberrypi1.png (дата звернення: 25.04.2025).
8. Фото Схеми кластеру Kubernetes. URL: https://images.app.goo.gl/igyr7T5sPFotwmTZ6 (дата звернення: 25.04.2025).
9. Фото Підключення SSD через USB до Raspberry Pi. URL: https://www.raspberrypi-france.fr/quelle-micro-sd-pour-raspberry-pi-4/ (дата звернення: 25.04.2025).
10. Фото Живлення для Raspberry Pi. URL: https://miniboard.com.ua/platy-rasshireniya/405-plata-rasshireniya-rezervnogo-pitaniya-dlya-rpi.html (дата звернення: 25.04.2025).
11. Sidharta H. A. Design and Implementation LETS (Low Power Cluster Server) for Sustaining UMKM during Pandemic. Kinetik: Game Technology, Information

System, *Computer Network, Computing, Electronics, and Control*. 2021. Vol. 6(1). P. 77–82.

12. Kim J. Development of an orchestration aid system for gridded crop growth simulations using Kubernetes. *Computers and Electronics in Agriculture*. 2021. Vol. 186. P. 106–187.

13. Jia C. Feng B. Yan B. et al. Overall design of fighter electronic warfare system architecture. Acta Aeronautica et Astronautica Sinica. 2021. Vol. 42(2). P. 335–347.

14. Hustad E. Olsen D. H. Creating a sustainable digital infrastructure: the role of service-oriented architecture. *Procedia Computer Science*. 2021. Vol. 181(1). P. 597–604.

15. Pratap Y. Pal N. Kumar Y. A formal approach for Docker container deployment. *Concurrency and Computation: Practice and Experience*. 2021. Vol. 33(20). e6364.

16. Medel V. Rafael T. José Á. Characterising resource management performance in Kubernetes. Computers & Electrical Engineering. 2018. Vol. 68. P. 286–297.

17. Lee S. Phan L. Park D. EdgeX over Kubernetes: enabling container orchestration in EdgeX. *Applied Sciences*. 2021. Vol. 12(1). P. 140–154.

18. Lim A. P. Thenuardi D. S. Survey on quality assurance testing on service-oriented architecture. *Proceedings of the 2020 International Conference on Information Management and Technology* (ICIMTech). 2020. P. 443–447.

19. Chen L. Yu L. Wang M. et al. Design on high availability log analysis system based on distributed cluster. *Journal of CAEIT*. 2020. Vol. 15(5). P. 420–426.

20. Kumar T. S. HS M. Mustapha S. M. F. D. Intelligent fault-tolerant mechanism for data centers of cloud infrastructure. *Mathematical Problems in Engineering*. 2022. P. 1–12.

21. Hollands J. G. et al. Cognitive load and situation awareness for soldiers: effects of message presentation rate and sensory modality. *Human Factors*. 2019. Vol. 61(5). P. 763–773.

22. Cristian F. Understanding Fault-Tolerant Distributed Systems. *Communications of the ACM*. 1993. Vol. 34. P. 56–78.

23. Majchrzycka A. Poniszewska-Marańda A. Secure Development Model for mobile applications. Bulletin of the Polish Academy of Sciences: *Technical Sciences*. 2016. Vol. 64. P. 495–503.

24. Kanso A. Toeroe M. Khendek F. Comparing redundancy models for high availability middleware. *Computing*. 2014. Vol. 96. P. 975–993.

25. Netto H. Oliveira C. P. Rech L. Alchieri E. Incorporating the Raft consensus protocol in containers managed by Kubernetes: An evaluation. *International Journal of Parallel, Emergent and Distributed Systems*. 2020. Vol. 35. P. 433–453.

26. Netto H. V. Lung L. C. Correia M. Luiz A. F. de Souza L. M. S. State machine replication in containers managed by Kubernetes. *Journal of Systems Architecture*. 2016. Vol. 73. P. 53–59.

27. Amirullah R. Ijtihadie R. M. Studiawan H. Optimasi Daya Data Center Cloud Computing Pada Workload High Performance Computing (HPC) Dengan Scheduling Prediktif Secara Realtime. *JUTI: Jurnal Ilmiah Teknologi Informasi*. 2017. Vol. 15(1). P. 1–10.

28. Dutta S. Mia I. The Global Information Technology Report 2009–2010. Geneva: *World Economic Forum and INSEAD, SRO-Kundig*. 2010.

29. Marinescu D. Computer Clouds. Complex Systems and Clouds. *Massachusetts: Morgan Kaufmann.* 2017. P. 113–145.

30. Burns B. Grant B. Oppenheimer D. Brewer E. Wilkes J. Borg, Omega, and Kubernetes: *Lessons Learned from Three Container-Management Systems Over a Decade*. Queue. 2016. Vol. 14(1). P. 70–93.

31. Truyen E. Kratzke N. Landuyt D. V. Lagaisse B. Joosen W. Managing Feature Compatibility in Kubernetes: Vendor Comparison and Analysis. *IEEE Access*. 2020. Vol. 8. P. 228420–228439.

32. Yang D. Wang D. Yang D. Dong Q. Wang Y. Zhou H. Cheng D. Hong H. DevOps in Practice for Education Management Information System at ECNU. *Procedia Computer Science*. 2020. Vol. 176. P. 1382–1391.

33. Fayos-Jordan R. Felici-Castell S. Segura-Garcia J. Lopez-Ballester J. Cobos M. Performance Comparison of Container Orchestration Platforms with Low Cost Devices in the Fog, Assisting Internet of Things Applications. *Journal of Network and Computer Applications*. 2020. Vol. 169. P. 1–13.

34. Mohamed M. Engel R. Warke A. Berman S. Ludwig H. Extensible Persistence as a Service for Containers. *Future Generation Computer Systems*. 2019. Vol. 97. P. 10–20.

35. Medel V. Tolosana-Calasanz R. Bañares J. Á. Arronategui U. Rana O. F. Characterising Resource Management Performance in Kubernetes. *Computers and Electrical Engineering*. 2018. Vol. 68. P. 286–297.

36. Taherizadeh S. Grobelnik M. Key Influencing Factors of the Kubernetes Auto-scaler for Computing-Intensive Microservice-Native Cloud-Based Applications. *Advances in Engineering Software*. 2020. Vol. 140(1). P. 1–11.

37. Rossi F. Cardellini V. Presti F. L. Nardelli M. Geo-Distributed Efficient Deployment of Containers with Kubernetes. *Computer Communications.* 2020. Vol. 159. P. 161–174.

38. Fitzpatrick J. An interview with Steve Furber. *Communications of the ACM*. 2011. Vol. 54(5). P. 34–39.

39. Shah S. A. R. Waqas A. Kim M.-H. Kim T.-H. Yoon H. Noh S.-Y. Benchmarking and performance evaluations on various configurations of virtual machine and containers for cloud-based scientific workloads. *Applied Sciences*. 2021. Vol. 11(3). P. 993.

40. Bhardwaj D. Challa R. Virtualization in cloud computing: Moving from hypervisor to containerization – A survey. *Arabian Journal for Science and Engineering*. 2021. Vol. 46(4). P. 8585–8601.

41. Martin A. Raponi S. Combe T. Di Pietro R. Docker ecosystem – Vulnerability analysis. *Computer Communications*. 2018. Vol. 122. P. 30–43.

42. Bernstein D. Containers and cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*. 2014. Vol. 1(3). P. 81–84.

43. Zhang Q. Liu L. Pu C. Dou Q. Wu L. Zhou W. A comparative study of containers and virtual machines in big data environment. *Proceedings of the IEEE 11th International Conference on Cloud Computing*. 2018. P. 178–185.

44. Mu X. Antwi-Afari M. F. The applications of Internet of Things (IoT) in industrial management: a science mapping review. *International Journal of Production Research*. 2024. Vol. 62(5). P. 1928–1952.

45. Poddar B. Sethia H. Jadon A. Azhar M. Internet of Everything in the Realm of Sustainable Marketing Post COVID-19: Constraints, Prospects, and Implications. *Smart and Sustainable Interactive Marketing. IGI Global*, 2024. P. 100–119.

46. Abba Ari A. A. Ngangmo O. K. Titouna C. Thiare O. Mohamadou A. Gueroui A. M. Enabling privacy and security in Cloud of Things: Architecture, applications, security & privacy challenges. *Applied Computing and Informatics*. 2024. Vol. 20(1/2). P. 119–141.

47. Qadri Y. A. Nauman A. Zikria Y. B. Vasilakos A. V. Kim S. W. The future of healthcare internet of things: a survey of emerging technologies. *IEEE Communications Surveys & Tutorials*. 2020. Vol. 22(2). P. 1121–1167.

48. Kuada E. Trust modelling and management system for a hyper-connected World of Internet of Everything. Proceedings of the IEEE 7th Int. *Conf. on Adaptive Science & Technology* (ICAST). 2018. P. 1–8.

49. Abdelwahab S. Hamdaoui B. Guizani M. Rayes A. Enabling smart cloud services through remote sensing: An internet of everything enabler. *IEEE Internet of Things Journal*. 2014. Vol. 1(3). P. 276–288.

50. Hashem I. A. T. Yaqoob I. Anuar N. B. Mokhtar S. Gani A. Khan S. U. The rise of “big data” on cloud computing: Review and open research issues. *Information Systems*. 2015. Vol. 47. P. 98–115.

51. Farooq M. U. Waseem M. Mazhar S. Khairi A. Kamal T. A review on Internet of Things (IoT). *International Journal of Computer Applications*. 2015. Vol. 113(1). P. 1–7.

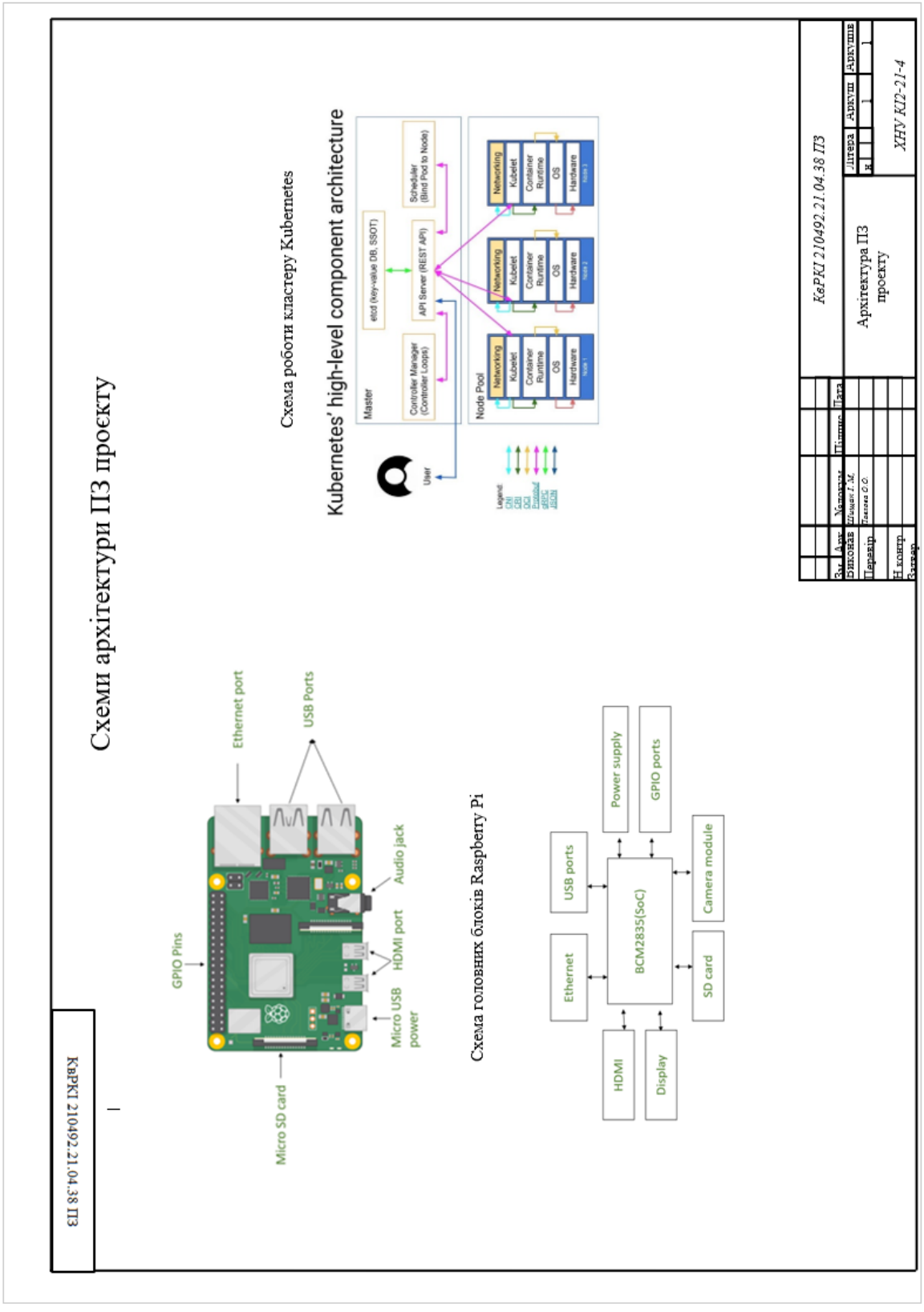
52. Kabir S. Internet of Things and safety assurance of cooperative cyber-physical systems: Opportunities and challenges. *IEEE Internet of Things Magazine*. 2021. Vol. 4(2). P. 74–78.

53. Sivanathan A. Gharakheili H. H. Sivaraman V. Managing IoT cyber-security using programmable telemetry and machine learning. *IEEE Transactions on Network and Service Management*. 2020. Vol. 17(1). P. 60–74.

Додаток А

(обов’язковий)

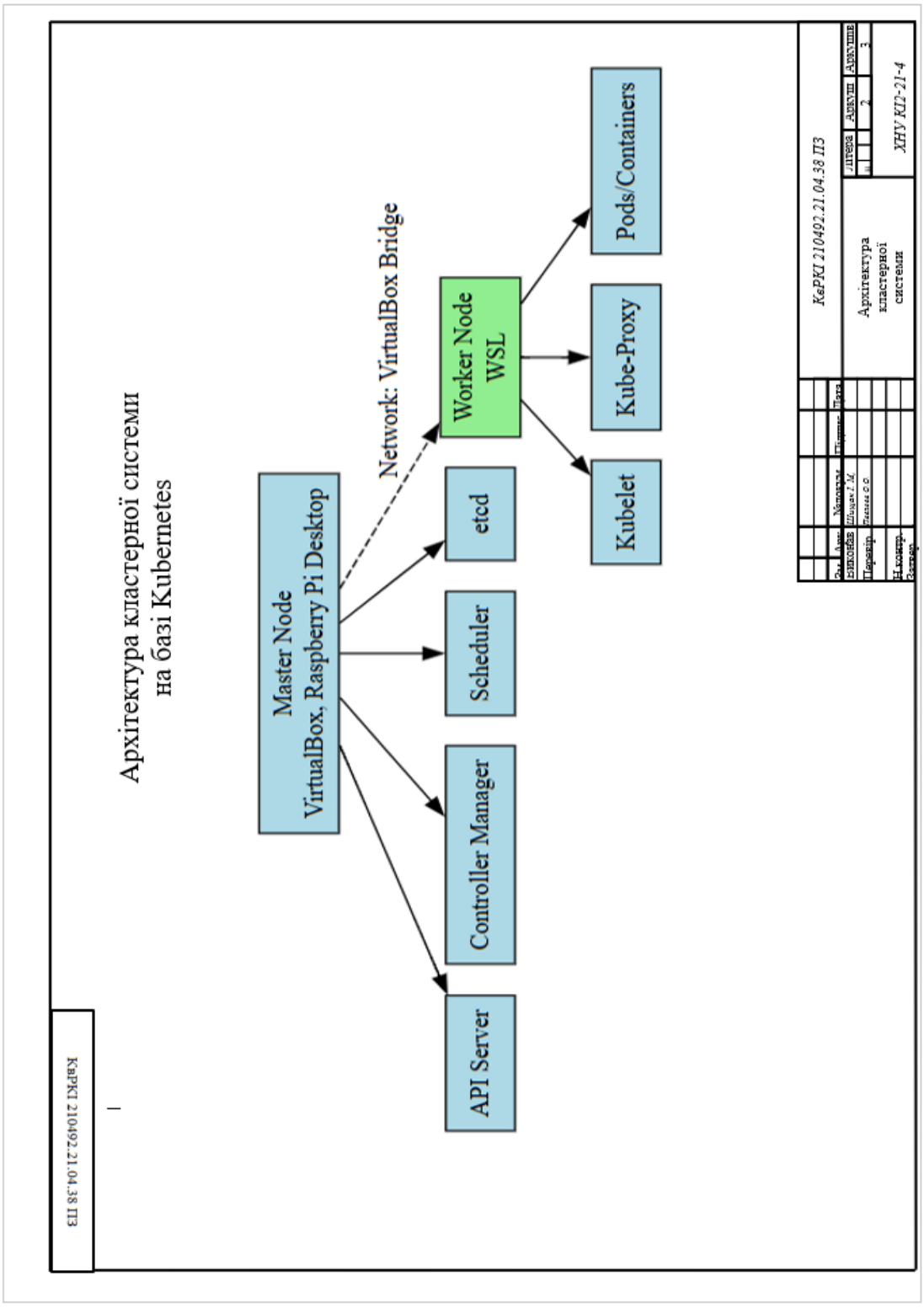
КОПІЯ КРЕСЛЕННЯ «АРХІТЕКТУРА ПЗ ПРОЄКТУ»



Додаток Б

(обов’язковий)

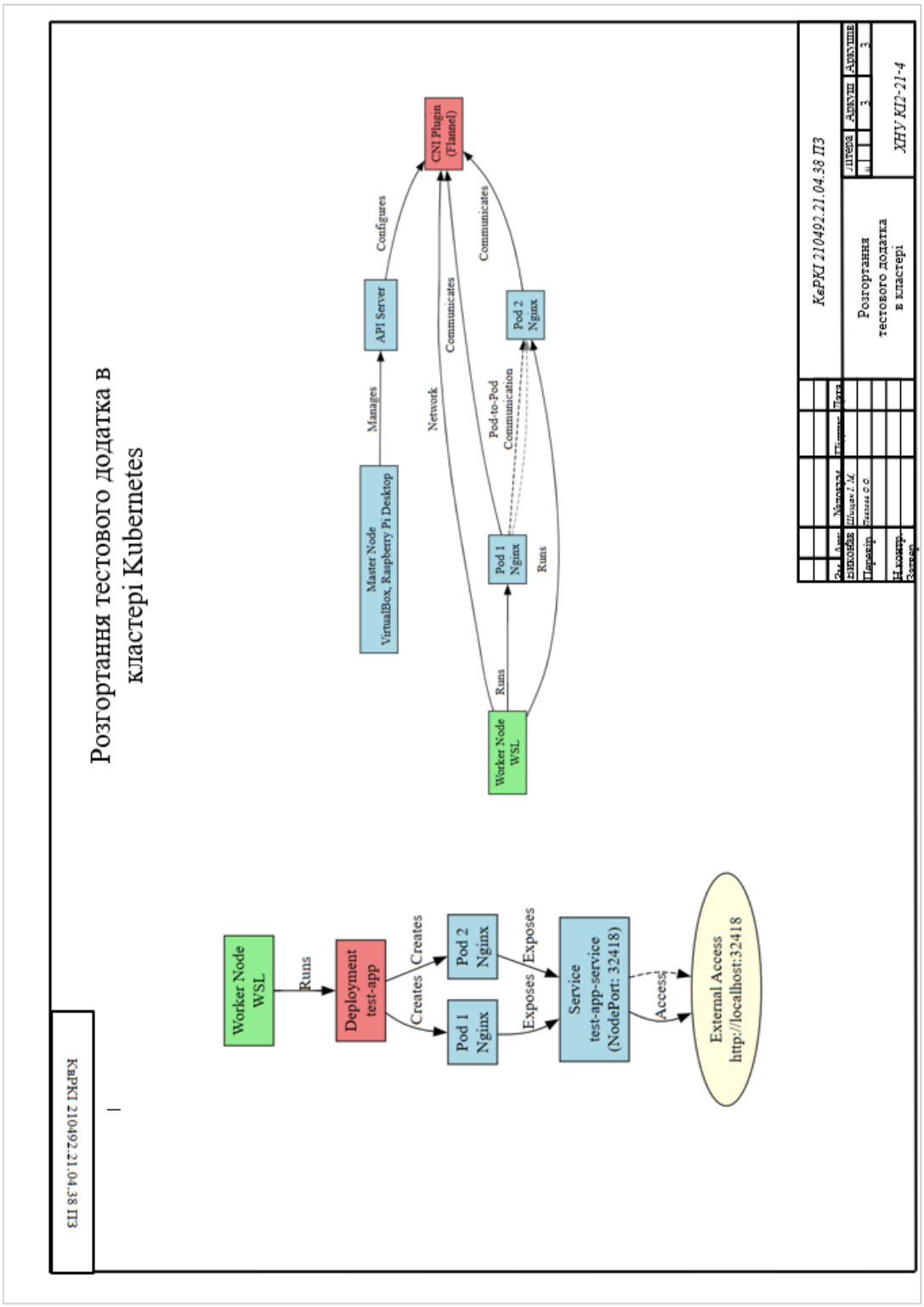
КОПІЯ КРЕСЛЕННЯ «АРХІТЕКТУРА КЛАСТЕРНОЇ СИСТЕМИ НА БАЗІ KUBERNETES»



Додаток В

(обов’язковий)

КОПІЯ КРЕСЛЕННЯ «РОЗГОРТАННЯ ТЕСТОВОГО ДОДАТКА В КЛАСТЕРІ KUBERNETES»



Додаток Г

(обов’язковий)

КОД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

```
apiVersion: V1
kind: Service
metadata:
        name: nginx
spec:
        selector:
        app: nginx
        ports:
        -protocol: TCP
        Port: 80
        targetPort: 80
type: NodePort



resources:
        limits:
        cpu: “500m”
        memory: “512Mi”
requests:
        cpu: “200m”
        memory: “256Mi”=
```

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 12.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 17%**

	Document		Sum coincidence on the DB	
ID: 242692 Title: БКР Кластерна система на основі Raspberry Pi із застосуванням платформи Kubernetes Added in a DB: 2025-05-31 Authors: Ігор ШИЩАК Heads: Сергій ЛИСЕНКО Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	96503	669	11957 (12%)	96 (14%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes
240782	Title: Звіт з ПДП Кластерна система на основі Raspberry Pi із застосуванням платформи Kubernetes Added in a DB: 2025-05-02 Authors: І.М. Шищак Heads: Лисенко С.М. Consultants: Opponents:	11205 (12.0%)	89 (13.0%)

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Ігор ШИЩАК

Співавтор:

Назва: ШИЩАК_Кластерна система на основі Raspberry Pi із застосуванням платформи Kubernetes

Експерт:

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:6.1%

Коефіцієнт подібності 2:3.1%

Мікропробіли: 14

Заміна букв: 1

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-05-31 07:56:36.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2025-05-31 Доцент Андрій Нічепорук

Дата експерт

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Шищак Ігор Михайлович

Тема: Кластерна система на основі Raspberry Pi із застосуванням платформи Kubernetes

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___57___

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є визначення умов та особливостей проєктування кластерної системи на основі одноплатних комп'ютерів Raspberry Pi із застосуванням платформи Kubernetes.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі було проведено аналіз предметної області побудови кластерних систем з використанням одноплатних комп'ютерів, зокрема Raspberry Pi, та оркестраційних платформ на зразок Kubernetes. Виявлено основні вимоги до сучасних розподілених обчислювальних систем, зокрема масштабованість, доступність, економічність і простота управління. В другому розділі було здійснено всебічне проєктування кластерної системи на базі одноплатних комп'ютерів Raspberry Pi із використанням контейнеризації та платформ оркестрації, зокрема Kubernetes (у версії K3s). У результаті аналізу й обґрунтування технічних рішень сформовано базову концепцію, що дозволяє реалізувати масштабовану та доступну обчислювальну інфраструктуру з мінімальними витратами, орієнтовану на навчальні, дослідницькі та прототипувальні завдання. У процесі формалізації архітектури системи проаналізовано принципи побудови кластеру, рольових моделей вузлів (master/worker), мережевої взаємодії, внутрішньої конфігурації та структури контейнеризованих сервісів. Було враховано не лише типову ієрархію Kubernetes, а й особливості розгортання у середовищі, де не

гарантована стабільна робота мережі або наявність високої відмовостійкості. В третьому розділі реалізація кластерної системи на основі Raspberry Pi i386 із застосуванням платформи Kubernetes було успішно створено та налаштовано функціональний кластер, що демонструє можливості використання апаратного забезпечення з обмеженими ресурсами для розгортання сучасних контейнеризованих застосунків. Використання VirtualBox для головного вузла та WSL з Ubuntu-22.04 для агента дозволило ефективно організувати програмно-апаратне середовище, яке забезпечило стабільну роботу k3s – полегшеної версії Kubernetes, оптимізованої для пристроїв із низькою обчислювальною потужністю. Налаштування головного вузла в середовищі VirtualBox із встановленням операційної системи Raspberry Pi Desktop, Docker і k3s забезпечило надійну основу для управління кластером, тоді як агент-вузол у WSL успішно інтегрувався до кластера через використання токена та API Kubernetes

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: -

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на високому науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: відмінно (9.75)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Радельчук Галина Іванівна, канд. техн. наук, доцент кафедри АКІТтаР ХНЕУ

"05" 06 2025 р. _____(підпис)

Завідувачу кафедри КІІС
д-р. філософії, доц. Ользі ПАВЛОВІЙ

Ігоря ШИЩАКА
ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ2-21-4

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Strike-Plagiarism та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

05.06 2025 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ
КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Кластерна система на основі Raspberry Pi із застосуванням платформи Kubernetes

Автор: Ігор ШИЩАК

Спеціальність: 123– Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Сергій ЛИСЕНКО, д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 10-40 джерелами на один фрагмент речення;
4) в якості запозичень в окремих місцях системою зафіксовано послідовності чотирьохрозряних двійкових кодів, які є вхідними даними до великої кількості задач і не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
5) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 6.10% і адресується до 47 першоджерела; та системою Anti-Plagiarism складає 12%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Сергій ЛИСЕНКО

Гарант ОП _____ Андрій Нічепорук

Завідувач кафедри КІІС _____ Ольга ПАВЛОВА