НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут телекомунікаційних систем
Кафедра інформаційних технологій в телекомунікаціях

До захисту допущено:
Завідувач кафедри
_____ Марія СКУЛИШ
«_____»_____2025 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційно-комунікаційні технології» спеціальності 172 Телекомунікації та радіотехніка
на тему: Підхід до автомасштабування кластерів Kubernetes на основі персональних обчислювальних ресурсів користувачів

Виконав: здобувач вищої освіти 4 курсу, групи ТІ-12
Ковальов Андрій Вячеславович _____
Науковий керівник: доцент кафедри ІТТ НН ІТС,
кандидат технічних наук, доцент
Алєксєєв Микола Олександрович _____
Рецензент: Доцент кафедри телекомунікацій НН ІТС,
кандидат технічних наук, доцент
Міночкін Дмитро Анатолійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут телекомунікаційних систем
Кафедра інформаційних технологій в телекомунікаціях

Рівень вищої освіти – перший (бакалаврський)
Освітньо-професійна програма «Інформаційно-комунікаційні технології»
спеціальності 172 Телекомунікації та радіотехніка

ЗАТВЕДЖУЮ
Завідувач кафедри
_____ Марія СКУЛИШ
«_____»_____2025 р

ЗАВДАННЯ
на дипломну роботу студенту
Ковальову Андрію Вячеславовичу

1. Тема дипломної роботи: Підхід до автомасштабування кластерів Kubernetes на основі персональних обчислювальних ресурсів користувачів, науковий керівник роботи доцент кафедри ІТТ НН ІТС, кандидат технічних наук, доцент Алєксєєв Микола Олександрович, затверджені наказом по університету від 26.05.2025р. № 1755-с.

2. Строк подання студентом дипломної роботи: 06.06.2025 р.

3. Об’єкт дослідження: Кластер Kubernetes на основі персональних обчислювальних ресурсів користувачів.

4. Вихідні дані до роботи: Гетерогенні персональні обчислювальні ресурси користувачів, відомості щодо їх технічних характеристик та встановлених операційних систем.

5. Перелік завдань, які потрібно розробити:

1) здійснити аналітичний огляд існуючого інструментарію, який можна ефективно використати для автомасштабування кластерів Kubernetes на основі персональних обчислювальних ресурсів користувачів;

2) розробити підхід щодо автоматичного масштабування кластерів Kubernetes на основі персональних обчислювальних ресурсів користувачів;

3) провести експериментальну перевірку запропонованого підходу, оцінити практичну ефективність рішення.

6. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація -18 слайдів, 31 таблиця, 41 рисунок

7. Дата видачі завдання: 8.09.2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Узгодження організаційних питань з науковим керівником	1.09.24-7.09.24	Виконано
2	Узгодження теми роботи та початок роботи над першим розділом	7.09.24-10.09.24	Виконано
3	Робота над першим розділом	10.09.24-10.11.24	Виконано
4	Робота над другим розділом.	10.11.24 - 15.12.24	Виконано
5	Робота над третім розділом.	15.12.24 - 2.02.25	Виконано
6	Висновки про ефективність рішення	2.02.25 - 28.02.25	Виконано
7	Оформлення дипломної роботи	28.02.25 - 1.03.25	Виконано
8	Підготовка презентації до захисту	1.03.25 - 1.04.25	Виконано

Здобувач вищої освіти _____ Андрій КОВАЛЬОВ

Науковий керівник роботи_____ Микола АЛЄКСЄЄВ

РЕФЕРАТ

Дипломна робота містить 70 сторінок, 12 рисунків та також використовує 14 літературних джерела посилань.

Актуальність теми:

У сучасних інформаційних системах можливість швидко адаптувати інфраструктуру до змін навантаження є одним із показників її ефективності, оскільки дозволяє виконувати ресурсномісткі задачі з одного боку коли це необхідно, а з іншого - економити при відсутності такої необхідності, а технологія Kubenetes, яка є одним з основних інструментаріїв таких інформаційних систем, має вбудовані механізми для автоматичного масштабування обчислювальних вузлів в залежності від навантаження. Однак, ці механізми орієнтовані або на хмарні середовища, де вони інтегруються з можливостями хмарних сервісів щодо автоматичного масштабування віртуальних машин, або на стаціонарні об’єднання фізичних обчислювальних пристроїв у кластери, де автомасштабування відбувається за рахунок реплікування програмних компонентів поміж обчислювальними вузлами кластерів, а не зміни кількості самих вузлів. Використання у якості таких робочих елементів кластеру персональних комп’ютерів, а також інших обчислювальних пристроїв, підключених до корпоративної мережі, які під час своєї роботи не використовують свої можливості повністю, надає потенційні можливості для вирішення більш ресурсномістких завдань в існуючих інформаційних системах на основі Kubernetes і одночасно дозволяють підвищити ефективність використання корпоративних обчислювальних ресурсів, але при цьому актуальною проблемою постає необхідність у механізмах автоматичного масштабування, які були б ефективними та економічно доцільними в умовах такого гетерогенного середовища з динамічною архітектурою. Таким чином, дана робота, яка присвячена створенню ефективного рішення для автоматизації масштабування кластерів Kubernetes у локальних гетерогенних середовищах з динамічною архітектурою на основі корпоративних обчислювальних ресурсів є актуальною.

Метою роботи є підвищення обчислювальної спроможності розподілених інформаційних систем на основі кластерів Kubernetes за рахунок розробки підходу до їх автоматичного масштабування із залученням незадіяних корпоративних обчислювальних ресурсів.

Об'єктом дослідження є горизонтальне масштабування кластерів Kubernetes у середовищах з гетерогенними невідчужуваними фізичними обчислювальними ресурсами та динамічною архітектурою.

Предметом дослідження є методи та технології реалізації автоматичного масштабування кластерів Kubernetes у фізичних середовищах.

Методами дослідження є аналіз та експертне оцінювання наявних підходів до масштабування Kubernetes, проектування та експериментальне моделювання.

Отримані результати. Проведено порівняльний аналіз існуючих рішень на базі хмарних платформ (AWS, GCP, Azure), статичного масштабування, технологій OpenStack та KubeVirt та ін. На основі отриманих результатів були обрані технології Cluster API та Metal3, розроблений власний підхід з їх використанням, проведено експериментальне дослідження працездатності розробленого рішення щодо забезпечення автомасштабування та ефективного використання обчислювальних ресурсів, що підтвердило доцільність використання розробленого підходу у гетерогенних інформаційних середовищах з динамічною архітектурою.

Результати роботи були апробовані на 19-й Міжнародній науково-технічній конференції "Перспективи телекомунікацій" а також прийняті до публікації у матеріалах конференції IEEE International Black Sea Conference on Communications and Networking 2025.

Ключові слова: KUBERNETES, АВТОМАСШТАБУВАННЯ, CLUSTER API, METAL3, CLUSTER AUTOSCALER, BARE METAL, ГОРИЗОНТАЛЬНЕ МАСШТАБУВАННЯ, ФІЗИЧНІ РЕСУРСИ.

ABSTRACT

The thesis contains 71 pages, 12 figures and also uses 14 references.

Relevance of the topic:

In modern information systems, the ability to quickly adapt infrastructure to changes in load is one of the indicators of its effectiveness, since it allows performing resource-intensive tasks on the one hand when it is necessary, and on the other hand, saving money when there is no such need, and Kubenetes technology, which is one of the main tools of such information systems, has built-in mechanisms for automatic scaling of computing nodes depending on the load. However, these mechanisms are focused either on cloud environments, where they integrate with the capabilities of cloud services to automatically scale virtual machines, or on stationary combinations of physical computing devices into clusters, where auto-scaling occurs due to replication of software components among computing nodes clusters, rather than changing the number of nodes themselves. The use of personal computers as such working elements of a cluster, as well as other computing devices connected to a corporate network that do not fully utilize their capabilities during their work, provides potential opportunities for solving more resource-intensive tasks in existing Kubernetes-based information systems and at the same time allows for increased efficiency in the use of corporate computing resources, but at the same time, the need for automatic scaling mechanisms such as Thus, the work devoted to creating an effective solution for automating the scaling of Kubernetes clusters in local heterogeneous environments with a dynamic architecture based on corporate computing resources is relevant.

The objective of this work is to develop and experimentally investigate an approach to automatic scaling of Kubernetes clusters.

The object of research is the process of automatic horizontal scaling of Kubernetes clusters in environments with physical computing resources (bare metal).

The subject of the study is methods and technologies for implementing automatic scaling of Kubernetes clusters in physical environments.

The research methods are analysis and expert evaluation of existing

approaches to Kubernetes scaling, design and experimental modeling. A comparative analysis of existing solutions based on cloud platforms (AWS, GCP, Azure), static scaling, OpenStack and KubeVirt technologies is carried out. Based on the results obtained, we developed our own approach using the Cluster API and Metal3, and conducted an experimental study of the developed solution's performance.

Results Obtained. An approach to automated scaling of Kubernetes clusters that provides dynamic integration of physical resources has been developed and successfully tested. The use of the Cluster API and Metal3 made it possible to provide auto-scaling and minimize resource consumption, which allows deploying these technologies in dynamic heterogeneous environments. The proposed approach has demonstrated a significant advantage in terms of flexibility, cost, and applicability compared to other approaches.

Keywords: KUBERNETES, AUTOSCALING, CLUSTER API, METAL3, CLUSTER AUTOSCALER, BARE METAL, HORIZONTAL SCALING, PHYSICAL RESOURCES.

ЗМІСТ

СПИСОК ТЕРМІНІВ І СКОРОЧЕНЬ

CRD Custom Resource Definitions

K8s Kubernetes

BMC Baseboard Management Controller

BM/VM Віртуальні сервери (машини)/Virtual servers (machines)

API Application Programming Interface

On-premises - тип розгортання IT-інфраструктури, при якому сервери, програмне забезпечення та обладнання розміщуються безпосередньо на території компанії або користувача, а не в хмарі.

Bare metal - фізичні сервери без попередньо встановленої операційної системи чи програмного забезпечення віртуалізації, які забезпечують прямий доступ до апаратних ресурсів.

Гіпервізор (Hypervisor) - програмне забезпечення, що створює та керує віртуальними машинами, надаючи їм доступ до ресурсів фізичного обладнання.

CPU Central Processing Unit

CNI Container Network Interface

CSI Container Storage Interface

Control plane - керуюча площина Kubernetes-кластера, набір компонентів, відповідальних за прийняття рішень щодо управління кластером, таких як розподіл навантаження, керування станом кластера та інші операції.

AMI Amazon Machine Image

PXE Preboot eXecution Environment

IPMI Intelligent Platform Management Interface

Kubernetes-native – властивий, притаманний Kubernetes,

ВСТУП

У сучасних інформаційних системах можливість швидко адаптувати інфраструктуру до змін навантаження є одним із показників її ефективності, оскільки дозволяє виконувати ресурсномісткі задачі з одного боку коли це необхідно, а з іншого - економити при відсутності такої необхідності, а технологія Kubenetes, яка є одним з основних інструментаріїв таких інформаційних систем, має вбудовані механізми для автоматичного масштабування обчислювальних вузлів в залежності від навантаження. Однак, ці механізми орієнтовані або на хмарні середовища, де вони інтегруються з можливостями хмарних сервісів щодо автоматичного масштабування віртуальних машин, або на стаціонарні об’єднання фізичних обчислювальних пристроїв у кластери, де автомасштабування відбувається за рахунок реплікування програмних компонентів поміж обчислювальними вузлами кластерів. Використання у якості таких вузлів персональних комп’ютерів, а також інших обчислювальних пристроїв, підключених до корпоративної мережі, які під час своєї роботи не використовують свої обчислювальні ресурси повністю, надають додаткові можливості для підвищення обчислювальної спроможності розподілених інформаційних систем, які мають організації і які побудовані на основі Kubernetes. Однак, використання кластерів Kubernetes на фізичному обладнанні (bare metal), яке, до того ж, може бути увімкнено або вимкнено в будь який час, актуалізує необхідність у методах автоматичного масштабування, які були б ефективними та економічно доцільними в таких умовах. Відсутність універсальних рішень для автоматизації масштабування кластерів Kubernetes у локальних гетерогенних середовищах робить актуальним розробку підходу, який би дозволив поєднати переваги автоматичного масштабування з економічно ефективним використанням персональних ресурсів користувачів.

Метою роботи є підвищення обчислювальної спроможності розподілених інформаційних систем на основі кластерів Kubernetes за рахунок розробки підходу до їх автоматичного масштабування із залученням незадіяних

корпоративних обчислювальних ресурсів.

Об'єктом дослідження є горизонтальне масштабування кластерів Kubernetes у середовищах з гетерогенними невідчужуваними фізичними обчислювальними ресурсами та динамічною архітектурою.

Предметом дослідження є методи та технології реалізації автоматичного масштабування кластерів Kubernetes у фізичних середовищах.

Методами дослідження є аналіз та експертне оцінювання наявних підходів до масштабування Kubernetes, проектування та експериментальне моделювання.

Наукова новизна роботи полягає у розробці нового підходу для автоматичного масштабування кластерів Kubernetes у середовищі bare metal, що передбачає залучення персональних ресурсів користувачів та використання сучасних інструментів управління інфраструктурою. Запропонований у дипломній роботі підхід є відповіддю на проблему відсутності універсального методу автоматичного масштабування фізичних серверів, що дотепер не мала комплексного та детального рішення.

Практичне значення роботи полягає у можливості застосування отриманих результатів для побудови економічно ефективних, гнучких та адаптивних обчислювальних систем, здатних оперативно реагувати на змінні навантаження без залучення додаткових зовнішніх ресурсів.

РОЗДІЛ 1
ПОРІВНЯЛЬНИЙ АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО МАСШТАБУВАННЯ КЛАСТЕРІВ KUBERNETES

1.1. Існуючі підходи до горизонтального масштабування Kubernetes.

На сьогодні існують різні підходи та методи реалізації горизонтального масштабування Kubernetes, кожен з яких має свої переваги та недоліки. Найбільш поширеними з них є використання популярних хмарних платформ (AWS, GCP, Azure), статичне масштабування, рішення на базі OpenStack та KubeVirt, а також порівняно новий підхід, заснований на Cluster API та Metal3.

Підходи на базі хмарних платформ, таких як Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure, надають можливість оперативно та автоматизовано збільшувати чи зменшувати кількість вузлів кластерів за допомогою відповідних API. Вони характеризуються високою гнучкістю, швидкістю реакції на зміну навантаження та простотою інтеграції з Kubernetes. Проте використання цих платформ супроводжується високими операційними витратами та значною залежністю від сторонніх провайдерів.

Статичне масштабування передбачає попереднє визначення та фіксування кількості вузлів кластера, що не змінюється в процесі роботи системи. Цей підхід є простим у реалізації та управлінні, однак він не дозволяє швидко реагувати на коливання навантаження, що призводить до недостатнього або надлишкового використання ресурсів та, відповідно, економічних втрат.

Рішення на базі OpenStack та KubeVirt забезпечують горизонтальне масштабування за допомогою віртуалізації. OpenStack дозволяє керувати віртуальними машинами, інтегруючи їх з Kubernetes через відповідні інтерфейси, тоді як KubeVirt безпосередньо розгортає віртуальні машини в Kubernetes-кластері. Ці рішення є більш економічно вигідними порівняно з традиційними хмарними сервісами, однак вимагають значних інвестицій у побудову інфраструктури та володіють більшою складністю при адмініструванні.

Підхід, заснований на Cluster API у поєднанні з технологією Metal3, дозволяє автоматично керувати життєвим циклом фізичних вузлів (bare-metal) кластера, залучаючи наявні обчислювальні ресурси, наприклад, персональні комп'ютери чи локальні сервери користувачів. Використання даного підходу дозволяє уникнути значних витрат, пов’язаних із закупівлею чи орендою додаткових ресурсів, забезпечуючи при цьому гнучкість, яка, зазвичай, притаманна хмарним рішенням.

Таким чином, актуальність проведення детального порівняльного аналізу перелічених підходів зумовлена необхідністю пошуку найкращого рішення для автоматичного масштабування Kubernetes-кластерів, що поєднувало би економічну ефективність, оперативність реагування на зміни навантаження та незалежність від сторонніх провайдерів. У рамках цього розділу буде здійснено детальний аналіз кожного з наведених підходів з використанням методу експертних оцінок.

1.2. Метод експертного оцінювання ефективності підходів до масштабування Kubernetes.

Після виокремлення актуальних рішень, які на сьогодні використовуються для масштабування Kubernetes-кластерів у різних середовищах, постає необхідність їх об’єктивного порівняння з метою визначення найбільш придатного варіанту для реалізації автоматичного масштабування на фізичному обладнанні. У рамках даного підрозділу пропонується здійснити багатокритеріальну оцінку існуючих підходів за допомогою методу експертних оцінок із подальшим зваженим ранжуванням.

Цей метод дозволяє врахувати як кількісні, так і якісні параметри, що важко вимірюються безпосередньо, але є критичними для реальної експлуатації систем масштабування. Підхід базується на залученні експертного кола для надання оцінок за визначеними критеріями, які попередньо були сформовані з урахуванням вимог до ефективного, масштабованого, стабільного та гнучкого середовища керування ресурсами кластерів.

У межах цієї методики передбачається:

визначення переліку ключових критеріїв, за якими будуть порівнюватися альтернативні рішення;

визначення вагових коефіцієнтів кожного критерію за допомогою парних порівнянь — для цього буде сформована матриця попарної важливості;

формування експертної групи з п’яти учасників, кожен з яких незалежно оцінив усі альтернативи за всіма критеріями.

перемноження оцінок на вагові коефіцієнти для обчислення узагальненого результату;

побудова інтегрального рейтингу, що дозволяє визначити найбільш ефективне рішення.

Застосування такого комплексного підходу забезпечує високу об'єктивність аналізу та дозволяє обґрунтовано обрати найкращий варіант на основі поєднання теоретичних знань і практичного досвіду експертів.

1.2.1. Комплексний аналіз ключових технологічних альтернатив для масштабування Kubernetes-кластерів.

У даному розділі розглядаються чотири ключові підходи до масштабування кластерів Kubernetes, які були відібрані для порівняння на основі їх популярності, технічних можливостей та практичної значущості. Першим із них є рішення, що базуються на використанні хмарних інфраструктур — таких як Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure. Усі ці сервіси реалізують масштабування через механізми інфраструктури як послуги (IaaS), мають потужні API, підтримують розподілене середовище та дозволяють оперативно реагувати на зміну навантаження. Незважаючи на деякі технічні відмінності, їх об'єднує спільна архітектурна логіка, модель витрат, а також залежність від зовнішнього постачальника. Саме тому ці платформи доцільно розглядати як єдину групу — хмарні рішення.

Наступним підходом є статичне масштабування. У таких сценаріях конфігурація кластеру залишається незмінною протягом усього періоду експлуатації, а кількість вузлів задається вручну. Цей метод характерний для

невеликих інфраструктур або середовищ із прогнозованим навантаженням, де автомасштабування або недоступне, або не вважається пріоритетним. Його перевагою є простота реалізації, однак цей підхід не забезпечує адаптивності й може призводити до перевитрат ресурсів або до деградації продуктивності в пікові моменти.

Ще однією альтернативою є використання технологій OpenStack та KubeVirt. Ці платформи дозволяють розгортати віртуальні обчислювальні середовища з можливістю гнучкого управління ресурсами. OpenStack позиціонується як комплексне рішення для створення приватної хмари, що може підтримувати як віртуальні, так і фізичні ресурси. Натомість KubeVirt інтегрується безпосередньо з Kubernetes і дозволяє запускати віртуальні машини поряд із контейнерами, забезпечуючи уніфіковане управління. Обидва варіанти мають широкий функціонал, проте їх складність, залежність від супровідної інфраструктури та високі вимоги до адміністрування можуть бути стримувальним чинником для впровадження.

Останнім підходом є використання Cluster API у поєднанні з Metal3. Це рішення створене для автоматизованого управління кластерами Kubernetes у середовищі bare metal, без необхідності у віртуалізації або використанні хмарних провайдерів. Cluster API забезпечує контроль над життєвим циклом кластерів, а Metal3 реалізує підтримку керованого розгортання фізичної інфраструктури. Такий підхід дозволяє поєднати переваги повної автономії, економічної ефективності та адаптивності до змін навантаження, що робить його надзвичайно привабливим для задач, де недоцільно або неможливо використовувати хмарні сервіси.

Таким чином, розгляд наведених варіантів дозволяє сформувати основу для подальшого аналізу, в якому ці підходи будуть оцінені за рядом критеріїв на основі методу експертного ранжування. Це дасть змогу обґрунтовано визначити найкраще рішення для реалізації системи автоматичного масштабування Kubernetes-кластерів, орієнтоване на використання існуючих фізичних ресурсів користувача.

1.2.2. Вибір критеріїв оцінювання та визначення їх вагомості.

Першим важливим критерієм є вартість. Масштабування кластерів може здійснюватися як за допомогою хмарних сервісів, так і з використанням локального фізичного обладнання. При цьому варто враховувати витрати на інфраструктуру, обслуговування, енергоспоживання, а також на оренду або утримання обчислювальних ресурсів. Рішення, які дозволяють оптимізувати витрати без шкоди для продуктивності, мають суттєву перевагу.

Другим критерієм було визначено рівень контролю над інфраструктурою. У деяких випадках користувач має повний адміністративний доступ до всіх компонентів системи, включно з операційною системою та мережею. В інших випадках керування обмежене інтерфейсами, які надає провайдер. Перевагу отримують підходи, які гарантують максимальну прозорість процесів і можливість тонкої настройки поведінки кластера.

Наступним критерієм виступає гнучкість. Мова йде про здатність системи масштабування адаптуватися до змін навантаження, зміни вимог до сервісів, обмежень середовища та структури самої інфраструктури. Чим вищою є ступінь гнучкості, тим ширше коло сценаріїв, у яких можна безболісно використовувати той чи інший підхід.

Не менш важливим є критерій складності. Він враховує як початкову складність налаштування системи, так і складність її обслуговування та модернізації в майбутньому. Простота впровадження дозволяє зменшити час на адаптацію команди, знизити ризики помилок і забезпечити стабільність процесів у довготривалій перспективі.

Ще одним ключовим фактором є автоматизація. Ефективне масштабування кластерів передбачає наявність механізмів, які дають змогу зменшити або повністю виключити ручне втручання у процес розгортання нових вузлів, балансування навантаження, відновлення після збоїв, тощо. Підходи, що передбачають глибоку інтеграцію з системами автоматичного керування, вважаються переважними.

Швидкість масштабування є критичним параметром у системах з високою динамікою навантаження. Від неї залежить здатність системи вчасно реагувати на зміни у трафіку, забезпечувати SLA (service level agreement) та підтримувати необхідну продуктивність у пікові моменти.

Останнім, але вкрай важливим критерієм виступає можливість впровадження рішення у гетерогенне, динамічне середовище, що базується на фізичних серверах. На відміну від віртуальних платформ, де розширення ресурсів часто відбувається через інтерфейси програмного забезпечення, у bare metal інфраструктурі виникає потреба у реальному фізичному керуванні — зокрема через BMC, IPMI або Redfish-протоколи. Універсальність і підтримка таких середовищ вимагають глибокої інтеграції на рівні низькорівневих сервісів, що далеко не кожен інструмент підтримує.

Усі вищезазначені критерії було винесено на попередню експертну оцінку. Наступним кроком стане визначення вагової значущості кожного з них, що дозволить виконати зважене порівняння альтернатив на основі багатофакторної моделі.

Для проведення об'єктивного багатокритеріального порівняння альтернатив у задачі вибору найсприятливішого підходу до масштабування кластерів Kubernetes було використано метод аналізу ієрархій (AHP), запропонований Т. Сааті. Цей метод дозволяє перетворити суб'єктивні експертні оцінки у кількісні вагові коефіцієнти на основі попарного порівняння критеріїв.

У першу чергу було сформовано множину критеріїв, релевантних для оцінки придатності кожного з підходів у контексті розгортання у гетерогенному динамічному середовищі: вартість, контроль, гнучкість, складність впровадження, автоматизація, швидкість масштабування та придатність для bare metal середовища.

Суть методу полягає у побудові матриці попарного порівняння. Кожен критерій порівнюється з кожним за дев’ятибальною шкалою Сааті, де 1 означає рівнозначну важливість, 3 — помірну перевагу одного критерію над іншим, 5

— істотну перевагу, 7 — значну, 9 — абсолютну. Значення 2, 4, 6, 8 використовуються як проміжні оцінки. Заповнення матриці здійснюється на основі експертного судження з урахуванням специфіки завдання: вибору найкращого підходу до масштабування кластерів Kubernetes.

Після побудови квадратної матриці розміром $n \times n$ (де n — кількість критеріїв), кожне значення в колонках нормалізується відносно суми значень цієї колонки. Далі для кожного рядка обчислюється середнє значення нормалізованих коефіцієнтів — це і є ваговий коефіцієнт відповідного критерію (див. Рис. 1.1).

	Вартість	Контроль	Гнучкість	Складність	Автоматизація	Швидкість	Застосованість у bare metal
Вартість	1	3	3	4	5	4	5
Контроль	0,333	1	2	3	3	3	4
Гнучкість	0,333	0,5	1	2	2	2	3
Складність	0,25	0,333	0,5	1	2	2	3
Автоматизація	0,2	0,333	0,5	0,5	1	2	2
Швидкість	0,25	0,333	0,5	0,5	0,5	1	2
Застосованість у bare metal	0,2	0,25	0,333	0,333	0,5	0,5	1

Рис. 1.1. Загальна форма матриці попарного порівняння.

Формально розрахунок проводиться за наступним алгоритмом:

Будується матриця попарного порівняння A, де кожен елемент a_ij вказує, наскільки критерій i важливіший за j.

Обчислюється сума елементів кожного стовпця:

$$Sj = \sum_{i=1} aij$$

Виконується нормалізація матриці, де кожен елемент a_ij ділиться на відповідну суму стовпця:

$$aij' = \frac{aij}{Sj}$$

Для кожного рядка обчислюється середнє значення нормалізованих елементів:

$$wi = \frac{1}{n}\sum_{j=1}^{n} aij'$$

Отримані значення w_i утворюють вектор вагових коефіцієнтів W, де кожен коефіцієнт відображає відносну важливість відповідного критерію у прийнятті рішення.

Після виконання наведених розрахунків було отримано наступний вектор вагових коефіцієнтів:

Вартість — 0.313

Контроль — 0.115

Гнучкість — 0.175

Складність — 0.050

Автоматизація — 0.094

Швидкість — 0.105

Реалізованість у гетерогенному середовищі — 0.147

Ці коефіцієнти надалі використовуються для підсумкового ранжування альтернатив за формулою зваженої суми:

$$\mathrm{Rk} = \sum_{i=1}^{n} (\mathrm{wi} \cdot \mathrm{rik})$$

де *R_k* — підсумковий бал альтернативи *k*, *w_i* — ваговий коефіцієнт критерію *i*, а *r_{ik}* — оцінка альтернативи *k* за критерієм *i*, виставлена експертами. Альтернатива з найменшим значенням *R_k* вважається найбільш оптимальною.

Таким чином, застосування методу AHP дозволило уникнути суб’єктивності у виборі критеріїв та забезпечило обґрунтовану основу для прийняття рішення щодо вибору найбільш придатного підходу.

Для визначення найкращого підходу до масштабування Kubernetes-кластерів було залучено групу із п'яти експертів, кожен з яких мав суттєвий досвід роботи в галузі хмарних обчислень та управління контейнерною інфраструктурою. До складу експертної групи входили провідні інженери DevOps, системні архітектори, спеціалісти з інфраструктурних рішень, що мають глибокі знання щодо Kubernetes, а також викладачі, які займаються

дослідженням сучасних технологій автоматизації та масштабування у рамках навчального процесу.

Експерти незалежно один від одного виставили оцінки по кожному критерію для усіх досліджуваних альтернатив. Після цього проведено процедуру узагальнення результатів, де було визначено середнє значення кожного критерію. Узагальнені оцінки зведені до єдиної таблиці (див. Рис. 1.2.), яку й було використано для подальшого аналізу. Наступним кроком став розрахунок підсумкової оцінки, який здійснювався шляхом множення отриманих узагальнених оцінок за кожним критерієм на відповідний ваговий коефіцієнт. В результаті аналізу методом експертних оцінок найвищу узагальнену оцінку отримав підхід з використанням Cluster API у поєднанні з Metal3, що свідчить про його переваги у всіх досліджених критеріях та доцільність подальшого використання у дипломній роботі.

№		A_1: Хмарні рішення	A_2: Статичні кластери	A_3: OpenStack+KubeVirt	A_4: Metal3		Вага коефіцієнту
1	Вартість	3	1	4	5		0,31
2	Контроль	1	5	5	5		0,12
3	Гнучкість	5	0	2	4		0,18
4	Складність	5	4	2	3		0,05
5	Автоматизація	5	4	4	5		0,09
6	Швидкість	5	4	3	5		0,11
7	Застосованість	0	5	5	5		0,15
	Загалом	3,17	2,62	3,70	4,72		

Рис. 1.2. Таблиця порівнянь із результатами вагових коефіцієнтів та загальною оцінкою підходів.

1.2.3. Обґрунтування результатів експертного аналізу підходів до автоматичного масштабування кластерів Kubernetes.

Використання хмарних сервісів для Kubernetes (наприклад, керованих кластерів типу Amazon EKS, Google GKE чи Azure AKS) характеризується моделлю оплати за споживання ресурсів. Такий підхід дозволяє уникнути значних початкових інвестицій, проте може призвести до вищих витрат у довгостроковій перспективі. Зокрема, за інтенсивного постійного навантаження сумарна вартість оренди хмарних ресурсів може перевищити витрати на володіння власним обладнанням[8][7]. Хмарні провайдери стягують плату за кожну годину роботи вузлів та додаткові послуги, тому при масштабуванні до великих обсягів або обробці «важких» даних рахунки можуть стрімко зрости[7]. Вигода хмари проявляється у відсутності капітальних затрат і можливості сплачувати лише за ті ресурси, що фактично використовуються, що особливо доцільно при несталому навантаженні. Однак, як відзначається у джерелах, компанії часто переймаються ризиком неконтрольованого зростання витрат у публічній хмарі[7], тому економічний фактор хмарних рішень не завжди є позитивним у довготерміновому розрізі.

Розгортання Kubernetes-кластера на власних фізичних серверах вимагає значних одноразових вкладень у обладнання та його інсталяцію, але надалі дозволяє знизити експлуатаційні витрати. Якщо інфраструктура вже наявна або була придбана, запуск Kubernetes на ній практично не генерує прямих періодичних витрат на сторонні сервіси[8]. Власний датацентр дає можливість уникнути щомісячних рахунків за оренду ресурсів у хмарі, обмеживши витратами на електропостачання, охолодження та підтримку обладнання. За умови повного завантаження потужностей та довгострокового використання, такий підхід може бути більш економічно ефективним[7]. Крім того, власна інфраструктура забезпечує передбачуваність витрат: після закупівлі серверів основні затрати стають фіксованими, на відміну від хмарної моделі з потенційними піковими стрибками вартості при зростанні навантаження[7]. Водночас, слід враховувати опосередковані витрати на утримання штату

фахівців для обслуговування кластеру та оновлення програмного забезпечення, що при обмежених ресурсах може нівелювати економію на оплаті хмарних сервісів.

Підхід, що поєднує приватну хмару OpenStack з технологією KubeVirt для запуску віртуальних машин у середовищі Kubernetes, також базується на власному обладнанні. З точки зору прямих витрат, він успадковує переваги on-premise рішення: відсутність рахунків публічним провайдерам і можливість використати наявні сервери. Ба більше, OpenStack дозволяє ефективно розподілити ресурси між різними проєктами і користувачами, потенційно підвищуючи коефіцієнт використання обладнання та економічність. Деякі джерела зазначають, що за наявності дешевих або вже амортизованих серверів власна хмара може бути вигіднішою за публічну, особливо при тривалому використанні[11]. Водночас, розгорнення OpenStack потребує виділення частини ресурсів для сервісних компонентів (контролерів, мережевих вузлів тощо) і несе значні операційні витрати. Підтримка OpenStack в робочому стані потребує висококваліфікованого персоналу[11], тому витрати на оплату праці та супровід системи можуть бути суттєвими. Крім того, додатковий шар віртуалізації (KubeVirt) споживає частину ресурсів на забезпечення роботи гіпервізора та управління віртуальними машинами. Таким чином, за критерієм вартості підхід OpenStack + KubeVirt отримав середню оцінку: він дешевший у довгостроковій перспективі за чисто хмарне рішення, проте дорожчий у впровадженні та підтримці, ніж простий статичний кластер чи більш легковагові інструменти автоматизації.

Cluster API + Metal3 підхід передбачає використання відкритих інструментів для автоматизованого розгортання кластерів на «голому металі» (без гіпервізора). Cluster API є проєктом спільноти Kubernetes, що надає декларативні API та контролери для створення і керування кластерами, а провайдер Metal3 інтегрує підтримку фізичних серверів через OpenStack Ironic[13]. Фактично це дозволяє керувати фізичними вузлами як ресурсами Kubernetes, не розгортаючи повну хмарну платформу. З огляду на вартість,

Cluster API + Metal3 є привабливим варіантом, оскільки є повністю відкритим (open-source) і не вимагає ліцензійних платежів. Він також не потребує настільки значних апаратних накладних витрат, як OpenStack, адже використовує лише окремі його компоненти (головним чином Ironic для керування залізом) без розгортання всіх сервісів приватної хмари[13]. Це спрощує інфраструктуру і зменшує витрати на її підтримку. Звичайно, залишається потреба у самих фізичних серверах та їх мережевій інфраструктурі, але ці витрати є спільними для всіх on-premises рішень. В результаті, за критерієм вартості підхід Cluster API + Metal3 було оцінено найвищим балом, оскільки він максимально скорочує накладні витрати на програмну інфраструктуру, дозволяючи зосередитися на ефективному використанні наявного обладнання.

Перенесення Kubernetes в хмару означає передачу значної частки контролю сторонньому провайдеру. Компанія користується готовим сервісом і не має прямого доступу до апаратного рівня чи налаштування управління керуючим кластером – ці аспекти цілком під відповідальністю провайдера. Такий підхід спрощує експлуатацію, але обмежує можливості тонкого налаштування і впровадження нестандартних рішень. За словами дослідників, розгортання Kubernetes у публічній хмарі не дає того рівня контролю над інфраструктурою, який можна отримати на власних серверах[7]. Наприклад, в адміністраторів відсутній доступ до налаштувань ядра ОС вузлів, до систем зберігання поза межами запропонованих сервісів, обмежена можливість вибору специфічного обладнання (процесорів, прискорювачів) тощо. Хоча провайдери зазвичай пропонують кілька конфігурацій вузлів, ці опції стандартизовані і не враховують усіх індивідуальних потреб. Отже, за критерієм контролю хмарні Kubernetes-рішення отримують найнижчу оцінку: вони суттєво поступаються on-premises рішенням, де організація сама визначає всі аспекти роботи кластеру.

Розгортання кластеру Kubernetes на власному обладнанні забезпечує максимальний ступінь контролю. У цьому випадку вся інфраструктура – як

апаратна, так і програмна – знаходиться під повним управлінням команди експлуатації. Адміністратори можуть вільно обирати апаратну конфігурацію серверів, налаштовувати мережеві параметри, застосовувати будь-які сумісні версії програмних компонентів Kubernetes, плагінів, CNI/CSI-драйверів тощо. Власний кластер дозволяє впроваджувати спеціальні політики безпеки, контролю доступу та відповідності нормативним вимогам, чого важче досягти в публічній хмарі[7]. Так, наприклад, якщо існують жорсткі вимоги щодо зберігання даних у визначеному географічному регіоні або ізоляції певних вузлів, локальне розгортання надає всі необхідні важелі контролю. В підсумку, статичний on-premises кластер отримав найвищий бал за контролем, адже дозволяє реалізувати принцип повного володіння набором технологій – від «заліза» до оркестрації контейнерів.

Поєднання OpenStack з KubeVirt також розгортається у власному середовищі, тому природно забезпечує високий рівень контролю. OpenStack – це платформа, відома майже необмеженими можливостями налаштування та керування ресурсами інфраструктури[11]. Через панель OpenStack або його API адміністратори можуть детально конфігурувати обчислювальні вузли (наприклад, резервування ядер CPU), мережеві сегменти, політики безпеки, сховища даних і багато іншого[10]. Зокрема, компонент Nova OpenStack надає прямий контроль над гіпервізорами та навіть дозволяє виділяти фізичні машини (bare metal) з необхідними атрибутами продуктивності[10]. Додавши KubeVirt, отримуємо єдину точку управління як контейнерами, так і віртуальними машинами через Kubernetes API[10], що потенційно спрощує експлуатацію змішаних навантажень. Втім, варто зауважити, що KubeVirt, будучи шаром абстракції над віртуалізацією, дещо обмежує можливості порівняно з прямим використанням Nova: зокрема, не всі низькорівневі функції (такі як тонке налаштування продуктивності, онлайн міграції або специфічні драйвери) доступні через інтерфейс KubeVirt[10]. Проте це обмеження стосується переважно роботи з окремими VM-вузлами; на рівні загального контролю інфраструктури зв’язка OpenStack + KubeVirt залишається одним із найбільш

потужних рішень. Таким чином, за критерієм контролю даному підходу теж можна присвоїти близький до максимального бал (лише трохи поступаючись статичному кластеру через додаткову складність шару KubeVirt).

Використання Cluster API з провайдером Metal3 також передбачає повну автономність у керуванні. Цей підхід, по суті, розширює модель Kubernetes на рівень інфраструктури, дозволяючи декларативно описувати бажаний стан не лише подів і сервісів, а й самих вузлів кластеру. Оскільки кластер розгорнуто на власному обладнанні, організація зберігає повний контроль над серверами, мережами та налаштуваннями програмного стеку. Важливою перевагою є уникнення «прив'язки до виробника»: Cluster API є універсальним інструментом з відкритим кодом, який підтримує різні платформи – від хмарних провайдерів до фізичних серверів[12]. Це означає, що компанія може уніфікувати процес керування кластерами у різних середовищах, зберігаючи контроль над тим, де і як розгортати робочі навантаження. На практиці, Cluster API + Metal3 дає змогу встановлювати свої політики автоматизації (наприклад, правила масштабування, оновлення вузлів), не залежачи від обмежень зовнішніх сервісів. Додатково, оскільки Metal3 використовує Ironic для роботи з «залізом», він успадковує багатий досвід цього проєкту у підтримці різноманітного обладнання та надійності операцій з ним[13]. У сукупності це робить контроль над інфраструктурою максимально гнучким і повним. Отже, підхід Cluster API + Metal3 отримує найвищу оцінку за контроль, не поступаючись традиційному статичному розгортанню, але надаючи при цьому більш зручні засоби управління.

Головною перевагою хмарних Kubernetes-сервісів є їх надзвичайна гнучкість у масштабуванні та конфігурації. Провайдери дозволяють швидко змінювати кількість вузлів, обирати типи серверів із різними параметрами продуктивності, використовувати додаткові керовані сервіси (балансування навантаження, бази даних, сховища) для побудови комплексної інфраструктури. Як відзначають аналітики, хмара пропонує безпрецедентну гнучкість і простоту масштабування порівняно з локальними розгортаннями[9].

В кілька кроків або API-запитів можна розгорнути новий кластер чи додати десятки вузлів до наявного, тоді як у фізичному середовищі це потребувало б значно більше зусиль. Крім того, хмарні провайдери постійно впроваджують нові можливості (наприклад, спеціалізовані обчислювальні сервери для машинного навчання, безсерверні обчислення тощо), що стають доступними користувачам без необхідності оновлювати власне обладнання. Зазначені фактори обумовили присвоєння хмарним підходам найвищої оцінки за гнучкість.

Водночас слід згадати і про обмеження: ця гнучкість реалізується в межах пропозиції конкретного провайдера, тож користувач певною мірою прив’язаний до його екосистеми сервісів. Перехід між різними хмарами або комбінування їх у мульти-хмарних рішеннях може виявитися нетривіальним завданням. Проте в контексті одного обраного провайдера можливості адаптації ресурсів під потреби додатку справді є найширшими, тому отримана оцінка максимальна.

Традиційний on-premises кластер доволі сильно поступається хмарі за гнучкістю. Після розгортання статичної конфігурації змінювати масштаб чи характеристики кластеру оперативно важко. Кількість вузлів визначається на етапі закупівлі та встановлення серверів; якщо робоче навантаження раптово зросте понад заплановане, додати нові вузли швидко не вдасться – доведеться проходити цикл закупівлі, доставки та інсталяції обладнання, що може тривати тижні. Так само, можливості «перекроювання» наявних ресурсів обмежені: фізичний сервер не можна просто розділити на дві частини для різних задач, на відміну від масштабування контейнерів або запуску додаткових віртуальних машин у хмарі. Отже, статичний кластер отримав низьку оцінку за гнучкість, оскільки його ресурси жорстко фіксовані. Це підтверджується і в літературі: локальна інфраструктура типово є менш масштабовною та гнучкою порівняно з хмарними альтернативами[7].

OpenStack + KubeVirt підхід можна вважати спробою поєднати гнучкість хмари з контролем локального середовища. OpenStack надає користувачу

хмароподібні можливості – динамічне створення віртуальних машин різної конфігурації, гнучке керування мережами (ізоляція, віртуальні маршрутизатори, плаваючі IP-адреси) та сховищами (типи дисків, різні класи зберігання) тощо[10]. З допомогою OpenStack можна швидко масштабувати інфраструктуру: запуск нового VM-вузла для кластеру займає лише кілька хвилин, за умови наявності вільних ресурсів на гіпервізорах. KubeVirt доповнює цю картину, дозволяючи запускати VM безпосередньо всередині Kubernetes, тобто розширює сферу застосування кластера на ті робочі навантаження, які не були контейнеризовані або потребують повноцінної віртуалізації. Таким чином, OpenStack + KubeVirt демонструє високу гнучкість: він підтримує одночасну роботу як хмарних контейнерних сервісів, так і традиційних VM, що дає змогу розміщувати різнорідні компоненти системи в сприятливому середовищі. Наприклад, критично важливу монолітну програму можна запустити у вигляді віртуальної машини з вимогами до апаратного прискорення або спеціальної ОС, тоді як інші мікросервіси працюють у контейнерах на тих же фізичних вузлах. Така універсальність підвищує гнучкість всієї платформи.

Однак слід зауважити, що впровадження двох рівнів оркестрації (OpenStack для VM і Kubernetes для контейнерів та VM через KubeVirt) істотно ускладнює систему. Це може обмежити оперативність змін: будь-які модифікації потребують узгодження між рівнями та ретельного адміністрування. Крім того, KubeVirt, попри свою корисність, не покриває всіх випадків – він не замінює цілковито можливості OpenStack Nova, як зазначалося вище, що може зменшувати гнучкість у специфічних сценаріях (наприклад, відсутність онлайн міграції для VM у KubeVirt може ускладнити динамічне балансування навантаження між вузлами). В цілому підхід OpenStack + KubeVirt отримав високу оцінку за гнучкість, хоча й дещо нижчу за чисто хмарний, враховуючи внутрішню складність цього рішення.

Гнучкість підходу Cluster API + Metal3 проявляється у здатності уніфікувати керування різними інфраструктурами та автоматизувати життєвий

цикл кластерів. Cluster API спроєктовано як абстракцію над конкретними провайдерами: завдяки йому однакові декларативні конфігурації можуть застосовуватися для розгортання кластерів як на VM у хмарі, так і на фізичних серверах[12]. У контексті Metal3 це означає, що bare-metal вузли керуються так само, як і будь-який інший ресурс Kubernetes, через Kubernetes API. Адміністратор може легко описати у YAML маніфесті додавання нового вузла і контролер автоматично виконає необхідні дії (вмикання сервера, завантаження образу ОС, включення вузла до кластеру). Таким чином, досягається гнучкість, раніше не властива фізичним середовищам: масштабування кластеру на «голому залізі» стає майже таким же простим, як у хмарі. Більше того, той самий інструментарій можна використовувати для керування різнорідними кластерами – наприклад, комбінувати вузли на VMware, OpenStack і bare metal в єдиній системі керування, якщо того вимагає інфраструктура підприємства[14][12].

Варто підкреслити, що Cluster API розроблений із використанням найкращих практик Kubernetes (контролери, оператори) для усунення людського фактору і забезпечення повторюваності процесів. Це підвищує гнучкість у налаштуванні кластерів: можна легко застосовувати зміни конфігурації (оновлення версій Kubernetes, заміну параметрів вузлів) через декларативні шаблони, і інструмент самостійно виконає необхідні зміни послідовно та узгоджено. Підхід Cluster API + Metal3 отримав високу оцінку за гнучкість, наближену до хмарної. Хоча фізичні сервери все ще не можна «клонувати» так само швидко, як віртуальні машини, дана технологія знімає більшість обмежень звичайного on-premises і дозволяє гнучко керувати кластером навіть у середовищі, далекому від публічної хмари.

Користування керованими Kubernetes-сервісами у хмарі мінімізує інженерні зусилля, необхідні для розгортання та підтримки кластеру. Провайдер бере на себе більшість складних завдань: налаштування головних-вузлів (control plane), оновлення версій Kubernetes, забезпечення високої доступності, відмовостійкості тощо. Для команди розробників кластер постає

як «чорний ящик», яким можна керувати через веб-консоль або API, не заглиблюючись у внутрішню реалізацію. У спрощеній аналогії, це подібно до оренди автомобіля напрокат: користувач отримує готовий до поїздки транспорт і не турбується про технічне обслуговування[8]. Практично, запуск нового кластеру або вузла у хмарі зводиться до виконання однієї команди або кількох натискань, у той час як on-premises рішення вимагало б розгортання серверів, налаштування Kubernetes та інтеграції всіх компонентів вручну. Всі ці фактори знижують операційну складність для кінцевого користувача. Як наслідок, хмарні підходи одержали найвищий бал за простоту (найнижчу складність), адже значною мірою абстрагують та автоматизують керування кластером, зменшуючи навантаження на команду експлуатації[8].

Звичайно, існують і зворотні сторони: використання хмари вимагає довіри до провайдера і може обмежувати можливості відлагодження складних проблем, що ховаються під капотом керованого сервісу. Проте з точки зору більшості користувачів це прийнятна плата за значне спрощення процесів розгортання та підтримки.

Власноручне розгортання Kubernetes на фізичних серверах є нетривіальним завданням, яке потребує експертизи та часу. Необхідно підготувати операційні системи на кожному вузлі, налаштувати компоненти control plane (APIServer, etcd, контролери, планувальник), забезпечити мережеву взаємодію між вузлами, встановити додаткові доповнення (CNI-плагіни, ingress-контролери тощо). У подальшому адміністраторам потрібно самостійно здійснювати моніторинг кластера, оновлення версій Kubernetes, виправлення проблем з вузлами. Це схоже на володіння власним автомобілем: власник має сам дбати про всі аспекти його роботи – від технічного обслуговування до усунення несправностей[8]. Такий рівень відповідальності означає вищу складність підтримки статичного кластеру порівняно з керованим хмарним. Згідно з практичними спостереженнями, команди повинні мати належний рівень підготовки, щоб успішно експлуатувати on-premises Kubernetes та не допускати критичних збоїв[8]. Відсутність автоматизованих

інструментів життєвого циклу (притаманних хмарі або Cluster API) підвищує ризик людських помилок і вимагає розробки власних скриптів або процедур для рутинних задач. Таким чином, за критерієм складності статичний підхід оцінений нижче за хмарний – як більш трудомісткий і складний у реалізації.

Проте слід зазначити, що статичний кластер – не означає хаотичний: є усталені інструменти на кшталт kubeadm чи Kubespray, які полегшують початкове розгортання. Але навіть з ними відповідальність за кожен аспект роботи кластера лежить на команді, що відрізняє цей варіант від керованих сервісів.

Комбінація OpenStack з Kubernetes (та ще й з інтегрованим KubeVirt) є, мабуть, найбільш складним у реалізації і супроводі варіантом серед розглянутих. OpenStack сам по собі – потужна, але й дуже комплексна система, що включає десятки сервісів (Nova, Neutron, Keystone, Glance, Cinder, Ironic тощо), які повинні бути правильно встановлені та взаємодіяти між собою[11]. Історично OpenStack має репутацію рішення, розгортання та утримання якого вимагає значних зусиль і глибокої експертизи, через що його використовують переважно великі організації, спроможні утримувати відповідні команди фахівців[11]. Додавання до цього середовища Kubernetes-кластера і KubeVirt фактично накладає один складний шар на інший. Команда має розбиратися і в нюансах OpenStack (для керування інфраструктурою), і в Kubernetes (для оркестрації контейнерів та VM через KubeVirt), а також забезпечувати коректну інтеграцію між ними. Наприклад, щоб автоматизувати створення вузлів для Kubernetes-кластера в OpenStack, можна використовувати сервіс Magnum або власні модулі – їх налаштування теж вимагає додаткових зусиль. У цілому, література вказує на високу порогову складність OpenStack-рішень: необхідні значні навички для встановлення, конфігурування, оновлення і усунення неполадок у такому середовищі[11].

Отже, за критерієм складності підхід OpenStack + KubeVirt отримує найнижчу оцінку (найвищу складність). Хоча він забезпечує чудову гнучкість і контроль, ці переваги досягаються ціною значного ускладнення системи. Таке

рішення виправдане лише тоді, коли організація має достатньо ресурсів для його підтримки або коли необхідні можливості не можуть бути отримані простішими способами.

Автоматизація, закладена в основу Cluster API, помітно спрощує багато операцій з керування кластером, які в статичному підході виконувались вручну. Cluster API впроваджує Kubernetes-подібний підхід до керування інфраструктурою: описавши бажаний стан (наприклад, «кластер з 5 вузлів таких-то характеристик»), адміністратор делегує всю важку роботу контролерам, які самостійно виконають потрібні дії – налаштують вузли, приєднають їх до кластеру тощо. Це суттєво знижує складність масштабування і оновлення кластерів. Замість того, щоб налаштовувати кожен сервер окремо, інженери працюють з декларативними конфігураціями, а система піклується про правильне послідовне застосування змін. У підсумку, впровадження Cluster API дозволяє уникнути багатьох ручних кроків і виключити типові помилки, пов’язані з людським фактором.

Втім, слід врахувати, що початкове освоєння Cluster API також вимагає певного часу та зусиль. Необхідно розгорнути керуючий кластер з контролерами Cluster API, ознайомитися з форматами CRD, налаштувати провайдер Metal3, який взаємодіє з апаратним забезпеченням. Для інженерів, незнайомих з Kubernetes-операторами, це може бути викликом. Але порівняно з OpenStack, архітектура Cluster API значно легша, а основні труднощі зосереджені на етапі впровадження. Подальша експлуатація у типових випадках потребує менше зусиль: наприклад, масштабування кластера або оновлення версії Kubernetes здійснюється редагуванням кількох параметрів у декларативному маніфесті, без ручного втручання до кожного вузла.

Зважаючи на наведене, за критерієм складності підхід Cluster API + Metal3 отримує оцінку вищу, ніж статичний кластер або OpenStack-рішення (що підтверджується його позитивною оцінкою в літературі як засобу для спрощення управління кластерами), але дещо нижчу, ніж у повністю керованого хмарного сервісу. Це компроміс між гнучкістю/контролем та

простотою: Cluster API суттєво автоматизує процеси, хоча й вимагає підтримки власної системи керування.

У контексті автоматизації хмарні Kubernetes-сервіси є безсумнівним лідером, оскільки практично всі аспекти життєвого циклу кластеру автоматизовані провайдером. Створення нового кластеру, додавання або видалення вузлів, заміна несправного вузла, масштабування під навантаженням – усі ці дії виконуються автоматично або напівавтоматично за ініціативою користувача, часто через прості інтерфейси або командні утиліти. Наприклад, в Amazon EKS чи Google GKE є функції авто-масштабування робочих вузлів: кластер сам збільшує або зменшує кількість нод залежно від навантаження, без участі адміністратора. Контрольний кластер також керується повністю автоматизовано: провайдер оновлює його до нових версій, застосовує виправлення без простоїв. Це суттєво зменшує адміністративні завдання та сприяє стабільності роботи. У результаті хмарний підхід отримав найвищий бал за автоматизацію, адже кінцевому користувачеві майже не потрібно виконувати рутинні операції вручну – вони винесені «за кадр» у площину відповідальності провайдера[8].

У on-premises кластері рівень автоматизації визначається зусиллями самої команди. «З коробки» Kubernetes надає певний мінімум – наприклад, Kubernetes має власний планувальник подів, механізми перезапуску впалих контейнерів, може трохи балансувати навантаження. Проте такі операції, як додавання нових вузлів до кластеру, заміна вузла, що вийшов з ладу, чи оновлення версії Kubernetes, автоматично не виконуються. Без додаткових інструментів адміністратори змушені вручну встановлювати нові сервери, запускати на них kubeadm (або інший інструмент) для приєднання до кластеру, видаляти неробочі вузли, слідкувати за сумісністю версій і т.д. Автоматизацію цих процесів можна досягти шляхом написання власних скриптів чи використання систем конфігураційного менеджменту (Ansible, Terraform), але це виходить за межі базового функціоналу. Таким чином, за критерієм автоматизації статичний підхід отримує низьку оцінку. У ньому майже відсутня вбудована

автоматизація масштабування – масштабування здійснюється «вручну» інженерними зусиллями.

Підхід із залученням OpenStack + KubeVirt пропонує певний рівень автоматизації завдяки наявності зрілих API для управління інфраструктурою. OpenStack надає повноцінні API для створення, видалення та керування віртуальними машинами (Nova), мережами (Neutron), сховищами (Cinder), а також додаткові сервіси типу Heat (оркестрація на основі шаблонів) чи Magnum (керування Kubernetes-кластерами). За правильної інтеграції це дозволяє значною мірою автоматизувати життєвий цикл інфраструктури. Наприклад, для масштабування Kubernetes-кластеру, що працює на OpenStack, можна залучити OpenStack Magnum: він здатен автоматично створювати нові VM-вузли і додавати їх до кластеру при збільшенні навантаження. KubeVirt, у свою чергу, дає можливість запускати VM через Kubernetes, використовуючи звичні механізми Kubernetes для управління їхнім життєвим циклом. Це означає, що розробники можуть описати вимоги до віртуальної машини (кількість CPU, RAM, образ диску) в Kubernetes-маніфесті, і кластер сам створить цю VM – подібно до того, як створює поди. Таким способом досягається автоматизація запуску віртуальних робочих навантажень поряд із контейнерними.

Однак повнота цієї автоматизації залежить від складності налаштування і зрілості інструментів. OpenStack потребує значних зусиль, щоб синхронізувати його авто-масштабування з Kubernetes. У літературі наголошується, що успішне впровадження подібного «схрещеного» рішення залежить не стільки від наявності окремих інструментів, скільки від вміння побудувати узгоджений процес між ними. Інакше кажучи, OpenStack + KubeVirt надає потужні засоби для автоматизації, але вимагає від команди чітко налаштованих «рейок», по яких ця автоматизація відбуватиметься. З урахуванням цього, даному підходу було виставлено середню оцінку за автоматизацію: кращу, ніж у статичного кластеру (через наявність API і вбудованих можливостей OpenStack), але нижчу за хмарні сервіси чи Cluster API, де автоматизація є більш цілісною і безшовною.

Головна ціль Cluster API – максимально автоматизувати управління Kubernetes-кластерами за рахунок Kubernetes-засобів. Це проявляється у всіх основних операціях. Розгортання нового кластеру? – Достатньо створити необхідні CRD-об’єкти (Cluster, MachineDeployment тощо) і контролери самі запустять вузли, встановлять Kubernetes і об’єднають їх у кластер. Масштабування? – Потрібно лише змінити параметр реплік у відповідному об’єкті, після чого Cluster API додасть або прибере вузли автоматично. Оновлення версії? – Створюється новий шаблон вузла з потрібною версією і ініціюється «rolling update», який послідовно замінює застарілі ноди на нові з мінімальним простоєм. Всі ці процеси виконуються автоматично контролерами Cluster API, що працюють у кластері-менеджері, зводячи роль адміністратора до формулювання бажаного стану системи[12]. Провайдер Metal3 інтегрується з IPMI/Redfish інтерфейсами серверів через Ironic, тому навіть такі низькорівневі операції, як ввімкнення фізичної машини, завантаження PXE-образу і встановлення ОС, – усе це теж відбувається у автоматичному режимі під контролем Kubernetes-оператора[13]. Фактично, Metal3 робить з фізичних серверів «ресурси першого класу» у Kubernetes-середовищі.

Завдяки цьому рівень автоматизації підходу Cluster API + Metal3 дуже високий. Особливо він проявляється у динамічних середовищах: якщо потрібно швидко розгорнути додаткові потужності, достатньо подати декларативну конфігурацію, і система сама виконає розподілену послідовність дій, яка б вручну зайняла набагато більше часу та була б схильна до помилок. Це підтверджує і практика великих кластерів: технологія Cluster API дозволяє керувати масштабуванням сотень кластерів та тисяч вузлів у стандартизований спосіб. Отже, за критерієм автоматизації підхід Cluster API + Metal3 отримав максимальний бал – нарівні з хмарними керованими сервісами. Він значно випереджає загальні on-premises рішення, оскільки забезпечує автоматизацію життєвого циклу bare-metal кластерів на рівні, наближеному до хмари.

Використання публічної хмари забезпечує найвищу швидкість масштабування та розгортання ресурсів. Кластер Kubernetes у хмарі може

динамічно додавати нові вузли за лічені хвилини або навіть секунди (у випадку безсерверних підходів), оскільки кожен вузол – це віртуальна машина, що стартує з заздалегідь підготовленого образу. Провайдери оптимізували цей процес: шаблони віртуальних машин містять усі необхідні компоненти, а мережа і сховища налаштовані для швидкого підключення. Наприклад, у AWS використовується механізм AMI з попередньо налаштованими параметрами для вузлів EKS, що значно скорочує час їх запуску. В результаті кластер може масштабуватися практично на льоту відповідно до потреб додатку. Той факт, що «хмара надає неперевершену легкість масштабування» відзначається і в оглядах технологій[9]. Тому за критерієм швидкості реакції на потребу в ресурсах хмарні рішення очікувано отримують найвищий бал.

У разі власного апаратного середовища швидкість масштабування значно поступається хмарі. Якщо всі наявні сервери вже задіяні і виникає потреба збільшити обчислювальні потужності, доведеться спочатку придбати новий сервер, дочекатися його доставки, встановити в стійку, налаштувати – лише після цього він зможе увійти до кластеру. Це дуже повільний процес у порівнянні з автоматизованим розгортанням VM. Навіть у випадку, коли певний запас незадіяних серверів є наперед (що рекомендується як найкраща практика), підключення їх до кластеру не є миттєвим – потрібно налаштувати ОС, Kubernetes і мережу. Відтак, статичний кластер отримує найнижчу оцінку за критерієм швидкості масштабування. Як жартують інколи адміністратори, «нелегко масштабувати кластер, якщо додавання вузла залежить від кур’єрської служби, що везе нове залізо».

Підхід із залученням OpenStack + KubeVirt покращує ситуацію зі швидкістю в порівнянні зі статичним кластером, адже впроваджує шар віртуалізації і автоматизації поверх фізичних серверів. При наявності вільних ресурсів на гіпервізорах OpenStack можна досить швидко розгорнути нову віртуальну машину: зазвичай це займає кілька хвилин (переважно витрачається час на завантаження образу диску та ініціалізацію ОС). Якщо інтегровано Magnum або інший інструмент, новий вузол Kubernetes може автоматично

приєднуватись до кластеру невдовзі після створення VM. Таким чином, час масштабування вимірюється хвилинами, що значно швидше, ніж у випадку очікування постачання обладнання.

Використання KubeVirt для запуску додаткових робочих навантажень (як VM) всередині кластера також відбувається досить швидко – приблизно з такою ж затримкою, як старт контейнера, помноженою на час завантаження гостьової ОС. За сприятливих умов (коли образи легкі, а ОС оптимізована для швидкого завантаження) це може бути небагато довше запуску поду. Проте, якщо йдеться саме про збільшення самого кластеру за рахунок нових вузлів, то OpenStack + KubeVirt все одно залежить від розгортання VM або фізичних серверів. У цілому, цей підхід отримує високу оцінку за швидкість масштабування – він наближається до хмарного за рахунок можливості динамічно створювати VM, хоча й може трохи програвати через накладні витрати на ініціалізацію гостьових систем.

Автоматизоване розгортання bare-metal вузлів через Metal3 дещо повільніше, ніж запуск віртуальних машин, але значно швидше, ніж традиційне ручне підключення серверів. Коли є попередньо встановлені (або підготовлені до PXE-завантаження) фізичні сервери, Cluster API може за командою почати процес їх введення в кластер: контролер BareMetalOperator увімкне машину через BMC, завантажить на неї образ операційної системи, встановить потрібні пакети Kubernetes і додасть вузол у кластер. Всі ці кроки займають певний час – в залежності від мережевої швидкості та обсягу образу ОС, це може бути 10–30 хвилин на вузол. Для фізичного середовища це дуже хороші показники, оскільки в минулому подібне налаштування вручну могло тривати годинами. До того ж, процес повністю автоматизований і може виконуватися паралельно на кількох серверах, що масштабно прискорює розгортання великих кластерів.

З точки зору швидкості реакції на непередбачене навантаження, Cluster API значно перевершує статичний підхід – при наявності вільних «stand-by» серверів він самостійно включить їх у роботу за відносно короткий проміжок часу. Звичайно, фізичним серверам властивий більш тривалий час ініціалізації

порівняно з VM, тому за абсолютною швидкістю масштабування Cluster API + Metal3 дещо поступається хмарним рішенням. Проте в контексті bare metal середовища це рішення є максимально швидкодіючим. Ми оцінили його близько до максимальної оцінки, з невеликим застереженням на користь хмари.

Якщо розглядати сценарій дуже варіативного, різнорідного середовища, де одночасно присутні різні типи обладнання, нестандартні конфігурації та потрібна тісна інтеграція з фізичною інфраструктурою, публічна хмара виглядає менш придатною. Хмарні провайдери пропонують уніфіковані стандартизовані ресурси (віртуальні машини певних розмірів, диски, мережеві рішення), що чудово працює для типових випадків, але може не задовольнити специфічні потреби, які виникають у гетерогенному середовищі підприємства. Наприклад, якщо в інфраструктурі є спеціалізовані пристрої або унікальні вимоги до мереж (приватні сегменти, нестандартні протоколи), інтегрувати їх з публічною хмарою може бути складно або неможливо. Також bare-metal середовище часто передбачає, що організація вже має власний парк серверів різних поколінь та виробників – перенесення всього цього у хмару може бути економічно або технічно недоцільним. Відтак, за критерієм пристосованості до гетерогенної динамічної on-premises інфраструктури, хмарні рішення отримують найнижчу оцінку. Вони розраховані переважно на однорідне середовище всередині конкретної платформи (AWS, Azure тощо), тоді як вимоги до безпосереднього керування «різношерстим» фізичним обладнанням не входять у їх модель.

Простий Kubernetes-кластер на власних серверах може працювати у гетерогенному середовищі в тому сенсі, що до нього можна включити вузли з різним апаратним оснащенням, різних моделей і навіть архітектур (за умови сумісності на рівні Kubernetes). Kubernetes досить гнучкий щодо цього – існують механізми label/taint для розрізнення вузлів, розподілу навантаження з урахуванням їхніх особливостей. Тому статичний кластер цілком застосовний у «різношерстому» датацентрі. Проте ключова проблема – динамічність: якщо середовище дуже динамічне, з частими змінами, додаванням і вилученням

різного обладнання, то статичний підхід потребує постійних ручних дій для адаптації кластера до цих змін. Це важко масштабувати. У гетерогенному середовищі часто потрібно інтегрувати нетипові компоненти (наприклад, GPU певної моделі, масиви зберігання певного виробника). Статичний підхід дає всю повноту контролю для цього (можна встановити потрібні драйвери, налаштувати вузол як завгодно), але не дає інструментів автоматичного керування при зміні складу обладнання. Тому ми оцінили статичний кластер середнім балом за даним критерієм: він придатний до гетерогенного середовища, але слабо пристосований до його високої динамічності.

Однією з причин використання OpenStack в корпоративних середовищах є саме потреба працювати з гетерогенною інфраструктурою. OpenStack спроєктовано як «універсальний оркестратор» для різноманітних ресурсів: він підтримує різні гіпервізори (KVM, VMware, Hyper-V тощо), може керувати фізичними вузлами через Ironic, інтегрується з різними мережевими обладнанням завдяки гнучкому Neutron, а також працює з широким спектром систем зберігання. Це робить його надзвичайно придатним для сценаріїв, де в датацентрі співіснує різнорідне обладнання і потрібен єдиний шар управління[10]. Крім того, OpenStack дозволяє створювати середовище з ізоляцією різних проектів, що корисно у великій організації. Додавання KubeVirt додає ще одну грань гетерогенності – можна на одних і тих самих серверах запускати і контейнери, і повноцінні VM. Це особливо корисно для застарілих чи специфічних додатків, які неможливо швидко контейнеризувати: їх можна «вписати» у сучасну Kubernetes-орієнтовану інфраструктуру через KubeVirt, під управлінням OpenStack. Таким чином, OpenStack+KubeVirt отримує дуже високий бал за застосовність у гетерогенних середовищах. Практично будь-яка різнорідна конфігурація може бути абстрагована і керована цим інструментарієм. Єдина причина, чому ми не ставимо йому беззастережно максимальний бал – це складність підтримки цієї гетерогенності. Але за функціональністю він закриває потреби в динамічному керуванні bare metal і віртуальними ресурсами на відмінно.

Підхід з Cluster API на bare metal спеціально задуманий для керування фізичними серверами у хмарний спосіб, що вже натякає на високу придатність до гетерогенних і динамічних середовищ. Провайдер Metal3 фактично є Kubernetes-обгорткою над OpenStack Ironic[13], що означає підтримку великої кількості різного обладнання (Ironic розроблявся з урахуванням роботи з серверами різних виробників, архітектур, з різними особливостями завантаження тощо). Metal3 також надає додаткові корисні можливості для різнорідного середовища – наприклад, контролер hardware-classification, який може автоматично маркувати вузли залежно від їх апаратних характеристик (кількість CPU, обсяг RAM, наявність GPU тощо)[13]. Це дозволяє легко керувати кластером, де вузли різні: система знає, чим вони відрізняються, і можна спрямовувати навантаження відповідно (через NodeSelector/Affinity у Kubernetes).

Cluster API як загальна концепція також підтримує множинні провайдери одночасно. Це означає, що в межах однієї реалізації можна мати, наприклад, кластер, частина вузлів якого керується через Metal3 (фізичні), частина через vSphere-провайдер (віртуальні на VMware) – подібні сценарії вже описані в спільноті[14]. Така гнучкість є надзвичайно важливою у гетерогенному середовищі, де різні підсистеми можуть вимагати різних типів ресурсів. Cluster API дозволяє об’єднати це під одним «дахом» Kubernetes API.

Отже, за критерієм застосовності в гетерогенно-динамічному (особливо bare metal) середовищі підхід Cluster API + Metal3 отримує найвищу оцінку. Він спеціально орієнтований на те, щоб принести хмарні підходи автоматизації в світ фізичних серверів, і успішно з цим справляється. Досвід впровадження Metal3 показує, що більшість ситуацій з різнорідним обладнанням, нестандартними мережами чи протоколами можна вирішити в його рамках, завдяки гнучкості Ironic та можливості розширення провайдера.

1.3. Висновки за проведеним експертним оцінюванням ефективності підходів до масштабування Kubernetes.

Проведений аналіз чотирьох підходів до масштабування Kubernetes-кластерів – хмарних сервісів, статичних on-premises кластерів, OpenStack + KubeVirt та Cluster API + Metal3 – за сімома ключовими критеріями продемонстрував суттєві відмінності між ними. Оцінки було виставлено п’ятьма фахівцями у галузі. Найкращі результати за більшістю критеріїв показав підхід Cluster API + Metal3. Він поєднує високу ступінь контролю та гнучкості власної інфраструктури з рівнем автоматизації та швидкодії, близьким до хмарних рішень, що підтверджено сучасними дослідженнями[13]. У конкурентному розгляді OpenStack+KubeVirt виявився потужним, але занадто складним у реалізації та підтримці; статичний кластер забезпечив повний контроль, проте поступився за гнучкістю та автоматизацією; хмарні сервіси відзначилися простотою та швидкістю, але мають недоліки у контролі витрат і використанню у bare metal середовищі. Сумарний рейтинг підтвердив доцільність вибору Cluster API + Metal3 як підходу для задач дипломного проєкту, але потребує більш поглибленого вивчення та перевірки у тестовому середовищі.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ ВИКОРИСТАННЯ CLUSTER API З METAL3 ДЛЯ АВТОМАСШТАБУВАННЯ КЛАСТЕРІВ KUBERNETES

2.1. Обґрунтування застосування Cluster API та Metal3 для автоматизації масштабування фізичних кластерів Kubernetes.

У попередньому розділі було доведено, що виникає потреба в уніфікованому підході до масштабування кластерів, який би працював на bare-metal подібно до хмарного середовища [1]. На відміну від публічних хмар, де Kubernetes може через API автоматично отримати додаткові вузли, у випадку фізичних (bare-metal) серверів відсутній простий інтерфейс для такого керування [2]. Це робить задачу автомасштабування on-premise кластера нетривіальною. У попередньому розділі було обґрунтовано, що рішенням цієї проблеми є використання Cluster API разом із компонентом Metal3 – такий підхід переносить логіку керування інфраструктурою у площину Kubernetes. Комбінація Cluster API + Metal3 дозволяє представляти фізичні сервери у вигляді Kubernetes-об’єктів і керувати ними декларативно. У результаті навіть у локальному середовищі, обмеженому у використанні хмарних API, можна досягти динамічного автомасштабування кластерів Kubernetes на рівні фізичного обладнання [2] . Це рішення є актуальним для підприємств, що експлуатують власні сервери і прагнуть отримати переваги хмарної гнучкості без залежності від зовнішніх провайдерів. Тому доцільно буде проаналізувати ці технології, чому і буде присвячений другий розділ.

2.2. Архітектура Cluster API.

Cluster API (CAPI) – це підпроєкт Kubernetes, що надає декларативні API та контролери для спрощення життєвого циклу кластерів (створення, оновлення, масштабування) [3]. Головна ідея полягає у тому, щоб застосувати “kubernetes-стиль” управління до самих кластерів: всі компоненти кластерної інфраструктури описуються у вигляді Kubernetes-ресурсів, а спеціальні контролери забезпечують приведення реального стану у відповідність до

бажаного [3]. Такий декларативний підхід забезпечує уніфіковане і узгоджене керування кластерами незалежно від типу підкладної інфраструктури (хмара чи фізичні сервери) [3]. Cluster API вводить низку Custom Resource Definitions (CRD), що моделюють кластер та його вузли. Зокрема, ресурс Cluster описує сам Kubernetes-кластер (наприклад, його ім’я, версію Kubernetes, мережеві параметри тощо) та містить посилання на специфічні для інфраструктури налаштування кластера. Ресурс Machine виступає декларативною специфікацією вузла інфраструктури, на якому має працювати Kubernetes-нод – наприклад, віртуальної чи фізичної машини [4] . Створення нового об’єкта Machine ініціює роботу відповідного провайдера інфраструктури, який виділяє новий хост і автоматично приєднує його до кластеру як ноду [4] . Важливо, що деталей про конкретне середовище (хмарного провайдера чи обладнання) немає в самому Machine – вони винесені у пов’язаний об’єкт InfrastructureRef, аби відокремити загальні параметри Kubernetes-вузла від специфіки провайдера [4]. Для групового управління вузлами визначено ресурс MachineSet, який подібно до ReplicaSet підтримує задану кількість машин, та MachineDeployment, що відповідає за оновлення наборів машин (аналогічно до Deployment для Pod) [4]. Наприклад, MachineDeployment дозволяє декларативно задати потрібну кількість робочих вузлів певної конфігурації, а контролери Cluster API будуть створювати або видаляти Machine/MachineSet об’єкти, щоб досягти вказаної кількості реплік. Таким чином, Cluster API розширює модель керування, знайому з Kubernetes (Deployment, ReplicaSet, тощо), на рівень цілих кластерів і їх машин. Архітектура Cluster API є модульною: вона складається з кількох типів провайдерів, які реалізують конкретні аспекти управління кластером [5] . Infrastructure Provider відповідає за взаємодію з інфраструктурою – тобто безпосереднє створення/видалення машин, налаштування мереж та балансувальників навантаження. Для різних середовищ існують різні провайдери інфраструктури: наприклад, AWS, Azure, GCP, vSphere, OpenStack, а також провайдер для bare-metal інфраструктури metal3 [5] . Bootstrap Provider забезпечує початкову ініціалізацію вузлів кластера – типовим є kubeadm

(Cluster API Bootstrap Provider Kubeadm), що генерує необхідні скрипти cloud-init (скрипти, що запускаються при першому старті серверу) для налаштування нових вузлів та приєднання їх до кластеру. Control Plane Provider керує розгортанням і станом компонентів control plane (головної ноди) (апі-сервер, контролер-менеджер, etcd) на виділених машинах; за замовчуванням використовується KubeadmControlPlane, який автоматично створює необхідну кількість control-plane вузлів і підтримує їх готовність [5]. Усі ці провайдери встановлюються як контролери в так званому кластері управління (management cluster) – Kubernetes-кластері, який стежить за ресурсами Cluster API. Звідти вони можуть створювати цільові кластер(и) (workload clusters) згідно з описаними користувачем маніфестами. Подібна організація дозволяє відокремити управління від робочих навантажень і забезпечити універсальність: завдяки провайдерам інфраструктури Cluster API абстрагується від конкретного типу ресурсів і здатен працювати як із віртуальними машинами в хмарі, так і з фізичними серверами в локальному середовищі [3].

2.3. Metal3 та управління bare-metal інфраструктурою.

Metal3 (Metal³) – це екосистемний проєкт, що розширює Cluster API підтримкою bare-metal інфраструктури. По суті, Metal3 надає реалізацію провайдера інфраструктури для фізичних серверів, інтегруючи можливості OpenStack Ironic у Kubernetes. Основними компонентами Metal3 є Bare Metal Operator та Cluster API Provider Metal3 (CAPM3), які спільно дозволяють Kubernetes-операторам декларативно керувати фізичними вузлами.

Bare Metal Operator (BMO) - це Kubernetes-контролер, що керує фізичними хостами, представленими у кластері через кастомні ресурси типу BareMetalHost (BMH) [6]. Кожен BareMetalHost містить інформацію про реальний сервер (наприклад, адресу BMC – контролера управління обладнанням, облікові дані для доступу, апаратні характеристики тощо). BMO використовує можливості сервісу Ironic – компонента для забезпечення “голого заліза” з OpenStack – і через BMC здійснює необхідні дії:

ввімкнення/вимкнення живлення сервера, завантаження образу ОС на диск, конфігурацію мережевого завантаження, перевірку стану заліза тощо [6] . Таким чином, BMO виступає проміжним шаром, який “перекладає” операції Ironic у вигляді подій і статусів Kubernetes-об’єктів BareMetalHost. Користувач, створюючи або змінюючи BareMetalHost CR, фактично задає кластеру, який сервер (з яких) доступний для використання і як його налаштувати, а BMO через Ironic виконує цю роботу.

CAPM3 інтегрує Metal3 з Cluster API, дозволяючи останньому керувати BareMetalHost-ами як інфраструктурними вузлами кластера [6]. CAPM3 додає спеціалізовані CRD (наприклад, Metal3Cluster, Metal3MachineTemplate та інші), які доповнюють стандартні ресурси Cluster API для врахування специфіки bare-metal середовища [6] . Принцип роботи такий: коли користувач через Cluster API запитує нову машину (створює об’єкт Machine у кластері управління), провайдер CAPM3 створює пов’язаний об’єкт Metal3Machine. Контролер CAPM3 на основі Metal3Machine підбирає один із доступних BareMetalHost (знаходиться у стані “Available”) і асоціює його з новою машиною. Далі Bare Metal Operator ініціює через Ironic процес підготовки цього сервера: завантажує на нього образ Kubernetes-вузла (зазвичай попередньо підготовлений образ з kubeadm-агентом), після чого новий вузол автоматично приєднується до Kubernetes-кластера. Аналогічно, при видаленні Machine, провайдер звільняє відповідний BareMetalHost (відмикає живлення або очищує його для подальшого використання). Важливо підкреслити, що всі ці дії відбуваються Kubernetes-засобами – через створення/видалення об’єктів і роботу контролерів – без необхідності прямого адміністрування обладнання. Комбінація Cluster API + Metal3 фактично перетворює фізичні сервери на керовані ресурси Kubernetes-кластеру, що значно спрощує експлуатацію локальної інфраструктури та уніфікує процеси з хмарними середовищами.

2.4. Автоматичне масштабування кластерів за допомогою Cluster Autoscaler.

Cluster Autoscaler – це компонент, що автоматично змінює кількість вузлів у кластері на основі поточного навантаження: він додає вузли, коли деякі поди не можуть бути заплановані через брак ресурсів, і видаляє вузли, коли ті простоюють. У типовому випадку (хмарна інфраструктура) Autoscaler викликає API хмарного провайдера для створення або видалення віртуальних машин. Для on-premise кластерів без хмарного API можливість автоматичного додавання фізичних вузлів була довгий час відсутня [2] . Cluster API розв'язує цю проблему, виступаючи проміжним шаром між Autoscaler і інфраструктурою. Замість прямої взаємодії з обладнанням, Autoscaler може керувати об'єктами Cluster API, які відповідають за вузли.

Зокрема, Cluster Autoscaler має режим cloud-provider = cluster-api, при якому він сприймає групи машин Cluster API як скейлінг-групи. В нашому контексті такою групою буде, наприклад, MachineDeployment, що відповідає за певний пул робочих вузлів. Autoscaler періодично перевіряє стан кластера: якщо знаходяться незаплановані поди, яким бракує ресурсів, він збільшує значення реплік у відповідному MachineDeployment (але не вище максимального, заданого політиками) [2]. Це призводить до створення нових об'єктів Machine, які через провайдер Metal3 ініціюють виділення додаткових BareMetalHost і запуск нових фізичних вузлів, як описано вище. Нові вузли автоматично реєструються в Kubernetes-кластері, і поди, що чекали ресурси, розподіляються на них. У протилежному випадку, коли є вузли, що тривалий час недовантажені, Autoscaler може зменшити кількість реплік MachineDeployment до мінімально допустимого рівня. Тоді Cluster API позначає відповідні Machine на видалення, і провайдер Metal3 через BMO відмикає або виводить зі складу кластера вибрані фізичні сервери. Важливо, що Autoscaler враховує налаштування користувача щодо мінімальної і максимальної кількості вузлів, а також пріоритети відключення, аби не порушити роботу критичних подів при масштабуванні вниз.

Через таку інтеграцію досягається автоматичне масштабування Kubernetes-кластера на фізичному обладнанні без прямої участі адміністратора

[2]. Kubernetes автоматично визначає, коли потрібні додаткові ресурси, і через Cluster API запускає процес їх виділення. Цей процес повністю прозорий для користувача додатків: розробники як подали нові навантаження, так і отримують під них ресурси, навіть якщо для цього кластеру довелося підняти новий сервер. Таким чином, на локальних кластерах, оснащених Cluster API + Metal3, досягається рівень еластичності, подібний до хмарного: кластер здатен адаптивно розширюватися і скорочуватися відповідно до потреб.

2.5. Переваги декларативного управління інфраструктурою в Kubernetes.

Комбінація Cluster API з Metal3 уможливлює декларативне управління інфраструктурою за тими ж принципами, що й управління звичайними Kubernetes-об’єктами. Всі бажані зміни – додати вузол, оновити версію Kubernetes, вилучити вузол – задаються у вигляді змін в YAML-маніфестах (ресурсах Cluster, Machine тощо). Контролери автоматично застосовують ці зміни до реальної інфраструктури. Такий підхід зменшує кількість ручних дій і, відповідно, ризик помилок, замінюючи їх автоматизованими та відтворюваними процесами [5]. Найкращі практики Kubernetes (декларативність, контрольований цикл узгодження, автоматизація) переносяться на рівень управління кластерами [5] . Це особливо важливо для on-premise середовищ, де раніше адміністрування серверів було повністю ручним – впровадження Cluster API дозволяє впорядкувати цей процес і керувати фізичними серверами так само, як контейнеризованими застосунками.

Другою суттєвою перевагою є “рідна” інтеграція з Kubernetes (Kubernetes-native infrastructure). Інфраструктурні ресурси (віртуальні чи фізичні машини, мережі) стають частиною кластера Kubernetes у вигляді об’єктів API, якими оперує кластерний контролер [5] . Це означає, що для їх керування не потрібні окремі сторонні інструменти – достатньо стандартних механізмів на кшталт kubectl (утиліта, що використовується для керування кластером K8s) або GitOps-підходів. Адміністратор може, до прикладу, зберігати цілковиту конфігурацію кластеру (включно з описом усіх вузлів і їх параметрів) в

репозиторії Git і застосовувати зміни через CI/CD конвеєр. Kubernetes виступає єдиним центром управління як додатками, так і інфраструктурою для цих додатків [5]. Централізація управління контейнерними робочими навантаженнями і інфраструктурою під ними в одній системі спрощує експлуатацію і моніторинг, усуваючи необхідність узгоджувати між собою різні платформи [5] .

Третя група переваг стосується гнучкості та адаптивності до наявних ресурсів користувача. Cluster API розроблений як агностичний до хмарного провайдера або типу обладнання інструмент – завдяки системі провайдерів він підтримує широкий спектр інфраструктури [3]. Це дозволяє легко застосувати єдиний підхід в гетерогенних середовищах або при міграції між ними. Наприклад, компанія може частково використовувати публічну хмару, частково власні сервери – Cluster API надасть уніфікований шар управління для обох випадків. Бібліотека готових провайдерів постійно зростає і охоплює не лише основні хмарні платформи, а й специфічні системи (напр., провайдери для VMware vSphere, OpenStack, MAAS, Bare Metal та ін.) [5]. Крім того, відкритий код і стандартні інтерфейси дозволяють створити власний провайдер під унікальні потреби, якщо існуючі не підходять. У випадку Metal3 це означає здатність врахувати особливості різного “заліза” – від різних протоколів BMC (IPMI, Redfish тощо) до інтеграції з користувацькими системами інвентаризації серверів. Незалежність від виробників і хмарних сервісів означає відсутність блокування на конкретного вендора: компанія сама контролює свою інфраструктуру і може розгортати кластер на будь-яких ресурсах, що є в її розпорядженні. Це актуально для організацій з підвищеними вимогами безпеки чи регуляторними обмеженнями, яким недоступні публічні хмари – Cluster API + Metal3 надає їм альтернативний шлях автоматизації в межах власного датацентру.

З точки зору ефективності, автоматичне масштабування на основі Cluster API забезпечує найсприятливіше використання фізичних ресурсів. Кластер динамічно збільшується під навантаження, уникаючи простою обладнання у

періоди низької активності. Це дозволяє будувати само адаптивні приватні хмари на базі Kubernetes, де апаратні ресурси додаються саме тоді, коли вони потрібні, і вивільняються, коли потреба зникає. У підсумку, підхід на основі Cluster API та Metal3 поєднує переваги хмарної гнучкості, декларативного управління та контролю над власною інфраструктурою. Він закладає науково обґрунтований фундамент для побудови систем автоматичного масштабування Kubernetes-кластерів у локальних середовищах [3] [5].

2.6. Висновки про використання Cluster API з Metal3 для автомасштабування кластерів Kubernetes.

В даному розділі здійснено детальний огляд поєднання Cluster API та Metal3 як основного підходу до реалізації автомасштабування Kubernetes-кластерів для гетерогенного середовища. Cluster API забезпечує декларативне, уніфіковане керування кластером незалежно від типу інфраструктури, а Metal3 дозволяє застосувати це керування до фізичних серверів, інтегруючи їх як ресурс кластера. Спільно з Cluster Autoscaler дані компоненти утворюють завершений цикл автомасштабування, при якому кластер самостійно реагує на зміни навантаження, додаючи або видаляючи вузли в режимі реального часу. Такий підхід є актуальним, технологічно прогресивним і практично значущим для локальних Kubernetes-інсталяцій. Він поєднує переваги хмарних практик з контролем над власним обладнанням, що робить його доцільним вибором для подальшого дослідження експериментально.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ АВТОМАТИЧНОГО МАСШТАБУВАННЯ KUBERNETES-КЛАСТЕРА ЗАСОБАМИ CLUSTER API, METAL3 ТА CLUSTER AUTOSCALER

3.1. Створення тестового стенду на рівні ОС та Kubernetes.

Для експериментального дослідження було розгорнуто тестове середовище на сервері ОС Linux Ubuntu 22.04, Kubernetes v1.32.0.

Першим кроком у підготовці тестового середовища стало встановлення всіх необхідних пакунків та залежностей для коректної роботи інструментарію. Після цього відбулася конфігурація серверу шляхом запуску скрипту 02_configure_host.sh, який автоматично створив набір віртуальних машин. Ці машини в подальшому розглядаються системою як фізичні вузли (bare metal hosts) і слугують повноцінною імітацією реального обладнання. Разом із цим було завантажено необхідні образи операційних систем, що використовуються для подальшого розгортання вузлів через PXE-завантаження за допомогою служби Ironic. Наступним етапом став запуск керуючого Kubernetes-кластера, який реалізується через виконання скрипту 03_launch_mgmt_cluster.sh. Залежно від операційної системи хоста, для цього використовується Minikube або Kind. Саме в цьому кластері розгортається оператор Bare Metal Operator (BMO), що забезпечує взаємодію з фізичними або віртуальними вузлами. Цей кластер відіграє роль керуючого середовища, у якому працюють контролери Cluster API, а також відбувається управління цільовими кластерами, що будуть розгортатися на вузлах bare metal. Завершальним кроком первинного налаштування стало виконання скрипту 04_verify.sh, який перевірив коректність усіх попередніх дій. У результаті виконання перевірки система підтвердила, що всі компоненти інфраструктури розгорнуті правильно, і середовище готове до подальшої роботи.

3.2. Налаштування тестового середовища Metal3.

Після завершення налаштування, у керуючому кластері повинні з’явитися об’єкти типу BareMetalHost у просторі імен metal3, які відображають наявні доступні вузли. За замовчуванням ці BareMetalHost-об’єкти переходять у стан Ready (готові до використання), що означає успішну інтеграцію віртуальних машин як ресурсів для подальшого розгортання кластерів (див Рис. 3.2).

```
jesferred@diploma:~$ kubectl get po -A
NAMESPACE                           NAME                                                             READY   STATUS    RESTARTS   AGE
baremetal-operator-system           baremetal-operator-controller-manager-7b6df9b6d9-82ktk           1/1     Running   0          29d
baremetal-operator-system           baremetal-operator-ironic-846cf6c9f5-vt47p                       5/5     Running   0          29d
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-manager-677474c566-ql2qk       1/1     Running   0          29d
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-manager-74c57ff875-lhtlm   1/1     Running   0          29d
capi-system                         capi-controller-manager-86dfb54d56-d75bg                         1/1     Running   0          29d
capm3-system                        capm3-controller-manager-69b9948699-7r2ll                        1/1     Running   0          29d
capm3-system                        ipam-controller-manager-675d75788c-s77cp                         1/1     Running   0          29d
cert-manager                        cert-manager-767f578ff-h74k6                                     1/1     Running   0          29d
cert-manager                        cert-manager-cainjector-c7fdb4dbf-jrzfc                          1/1     Running   0          29d
cert-manager                        cert-manager-webhook-768bf9d966-clnmn                            1/1     Running   0          29d
default                             high-load-deploy-7b795ccf84-6jcgh                                1/1     Running   0          29d
default                             high-load-deploy-7b795ccf84-nrkbn                                0/1     Pending   0          29d
kube-system                         calico-kube-controllers-749b79b486-cf5zj                         1/1     Running   0          29d
kube-system                         calico-node-8t627                                                1/1     Running   0          29d
kube-system                         calico-node-xrdz5                                                1/1     Running   0          29d
kube-system                         cluster-autoscaler-clusterapi-cluster-autoscaler-cdcf8fbb8smqll  1/1     Running   0          29d
kube-system                         coredns-668d6bf9bc-trlpn                                         1/1     Running   0          29d
kube-system                         coredns-668d6bf9bc-v76gb                                         1/1     Running   0          29d
kube-system                         etcd-test1-dlwrd                                                 1/1     Running   0          29d
kube-system                         kube-apiserver-test1-dlwrd                                       1/1     Running   0          29d
kube-system                         kube-controller-manager-test1-dlwrd                              1/1     Running   0          29d
kube-system                         kube-proxy-fs2mh                                                 1/1     Running   0          29d
kube-system                         kube-proxy-kpmkh                                                 1/1     Running   0          29d
kube-system                         kube-scheduler-test1-dlwrd                                       1/1     Running   0          29d
monitoring                          graf-grafana-5bc7c7dfb7-r9wwf                                    1/1     Running   0          26d
monitoring                          prom-alertmanager-0                                              0/1     Pending   0          26d
monitoring                          prom-kube-state-metrics-7b576477cc-ks727                         1/1     Running   0          26d
monitoring                          prom-prometheus-node-exporter-bwtpv                              1/1     Running   0          26d
monitoring                          prom-prometheus-node-exporter-mmpxm                              1/1     Running   0          26d
monitoring                          prom-prometheus-pushgateway-66f7c88c99-scmh2                     1/1     Running   0          26d
monitoring                          prom-prometheus-server-54d9887dd5-k6ffk                          2/2     Running   0          26d
```

Рис. 3.1. Всі компоненти кластеру Kubernetes, Metal3 та Cluster API.

```
jesferred@diploma:~/metal3-dev-env$ kubectl get bmh -A
NAMESPACE   NAME     STATE         CONSUMER            ONLINE   ERROR   AGE
metal3      node-0   provisioned   test1-2gjnn-96wz9   true             3h8m
metal3      node-1   provisioned   test1-g4hlr         true             3h8m
jesferred@diploma:~/metal3-dev-env$ kubectl get -A cluster
NAMESPACE   NAME    CLUSTERCLASS   PHASE         AGE   VERSION
metal3      test1                  Provisioned   37m
jesferred@diploma:~/metal3-dev-env$ kubectl get -A kubeadmControlplane
NAMESPACE   NAME    CLUSTER   INITIALIZED   API SERVER AVAILABLE   REPLICAS   READY   UPDATED   UNAVAILABLE   AGE   VERSION
metal3      test1   test1     true          true                   1          1       1         0             28m   v1.32.0
jesferred@diploma:~/metal3-dev-env$ kubectl get -A MachineDeployment
NAMESPACE   NAME    CLUSTER   REPLICAS   READY   UPDATED   UNAVAILABLE   PHASE     AGE   VERSION
metal3      test1   test1     1          1       1         0             Running   18m   v1.32.0
jesferred@diploma:~/metal3-dev-env$ kubectl get - A KubeconfigTemplate
error: the server doesn't have a resource type "-"
jesferred@diploma:~/metal3-dev-env$ kubectl get -A KubeconfigTemplate
error: the server doesn't have a resource type "KubeconfigTemplate"
jesferred@diploma:~/metal3-dev-env$ kubectl get -A Metal3Cluster
NAMESPACE   NAME    AGE   READY   ERROR   CLUSTER   ENDPOINT
metal3      test1   38m   true            test1     {"host":"192.168.111.249","port":6443}
jesferred@diploma:~/metal3-dev-env$ kubectl get -A Metal3DataTemplate
NAMESPACE   NAME                              CLUSTER   AGE
metal3      test1-controlplane-template               29m
metal3      test1-workers-template                    19m
jesferred@diploma:~/metal3-dev-env$ kubectl get -A Metal3MachineTemplate
NAMESPACE   NAME                AGE
metal3      test1-controlplane   30m
metal3      test1-workers        19m
```

Рис. 3.2. Перевірка створених ресурсів (BareMetalHosts, Cluster, KubeadmControlPlane, MachineDeployment, Metal3Cluster, Metal3DataTemplate, Metal3MachineTemplate. KubeconfigTemplate відсутній, оскільки workload cluster ще не з'явився).

Кількість створених BareMetalHost (BMH) визначається конфігураційною змінною, що дозволяє задавати потрібну кількість вузлів для експерименту. У нашому випадку тестовий стенд було конфігуровано з кількома вузлами, щоб забезпечити запас ресурсів для автоматичного масштабування (як мінімум один додатковий вузол понад початкові потреби кластера).

3.3. Компоненти тестового середовища у Metal3.

У підсумку, після виконання скриптів, тестовий стенд складався з таких основних компонентів:

Керуючий Kubernetes-кластер (ephemeral cluster) – невеликий кластер (Minikube або Kind) для запуску контролерів Cluster API. Він виступає як «bootstrap»-кластер, з якого ми керуємо створенням основного (цільового) кластера.

Контролери Cluster API – контролери інфраструктури (зокрема Cluster API Provider Metal3, CAPM3) розгорнуті в керуючому кластері(див. Рис. 3.1). CAPM3 є провайдером Cluster API, який дає можливість розгортати Kubernetes-кластери на фізичній інфраструктурі (bare metal) із використанням проекту Metal3. Іншими словами, CAPM3 оптимізує Cluster API працювати з BareMetalHost через Metal3.

Bare Metal Operator (BMO) – оператор Kubernetes, що відповідає за управління BareMetalHost-ресурсами. BMO реалізує Kubernetes API для фізичних вузлів та веде облік наявних серверів у вигляді ресурсів BareMetalHost. Він інтегрується з OpenStack Ironic для виконання операцій над «залізом»: інспектування апаратного забезпечення, завантаження ОС на вузол, очищення дисків тощо. У нашому випадку BMO було розгорнуто у просторі імен metal3 керуючого кластеру(див. Рис. 3.1).

OpenStack Ironic – сервіс, що безпосередньо здійснює provisioning (PXE-завантаження та установку образу ОС) на bare metal. У нашому кластері Ironic запускається автоматично в контейнерах і працює разом з BMO. Ролі BMO та

Ironic у зв’язці такі: BMO через CRD BareMetalHost повідомляє Ironic, який образ і на який вузол потрібно розгорнути, та відстежує статус цього процесу.

Віртуальні «фізичні» вузли – набір ВМ під керуванням libvirt/QEMU, кожна з яких має свій віртуальний BMC і мережевий інтерфейс для PXE-завантаження. Ці ВМ представляються у Kubernetes як ресурси BareMetalHost. Для нашого експерименту, наприклад, було підготовлено два такі BMH (умовно назвемо їх node-0, node-1), що відповідає одному використаному вузлу і одному запасному для масштабування.

Компонентну діаграму, що ми отримали, можна побачити на рисунку 3.3.

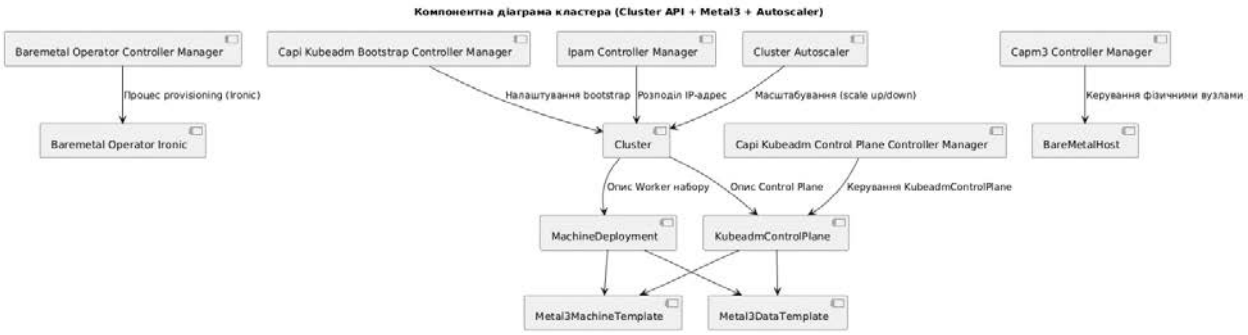

Рис. 3.3. Компонентна діаграма Cluster API + Metal3 + Autoscaler.

Завдяки описаному стенду ми отримали Kubernetes-середовище, здатне керувати «фізичними» вузлами програмно, використовуючи декларативні об’єкти. Це створило необхідну основу для подальшого розгортання тестового Kubernetes-кластера і дослідження механізмів його автоматичного масштабування.

3.4. Налаштування компонентів та розгортання Kubernetes-кластера.

На цьому етапі в керуючому кластері було описано та запущено цільовий Kubernetes-кластер за допомогою Cluster API. Розгортання здійснювалося декларативно: користувач створює необхідні Kubernetes-об’єкти (Custom Resources), а контролери Cluster API та Metal3 автоматично провадять всі дії для запуску нового кластеру на наявних фізичних вузлах. Ми сфокусувалися на

успішному сценарії розгортання, тому всі компоненти були попередньо налаштовані правильно, що дозволило уникнути помилок на цьому кроці.

Декларативна специфікація кластера. Для розгортання кластера було підготовлено маніфести, що описують такі ресурси Cluster API:

Cluster – ресурс верхнього рівня, який задає конфігурацію цільового кластеру (назву, параметри мережі тощо). Наш кластер отримав назву, наприклад, test1. Паралельно з ним створюється пов’язаний ресурс Metal3Cluster, який містить специфічні для Metal3 параметри (наприклад, посилання на адреси DHCP/Ironic, якщо потрібно).

KubeadmControlPlane – ресурс, що описує контрольну площину (master-вузли) кластера, використовуючи kubeadm. Ми задали бажану кількість контрольних вузлів = 1 (один master) та версію Kubernetes для них. Цей ресурс посилається на шаблон Metal3MachineTemplate для контрольних вузлів – шаблон визначає, яке апаратне забезпечення та образ ОС використати для кожного майстер-вузла. У нашому випадку Metal3MachineTemplate містив посилання на попередньо завантажений образ Ubuntu для Kubernetes (наприклад, cloud-образ Ubuntu 22.04 з встановленим kubeadm) та базові характеристики вузла (CPU, RAM, диск – відповідно до параметрів наших ВМ).

MachineDeployment – ресурс, що описує групу робочих (worker) вузлів кластеру, аналогічно до Deployment для Pod’ів. Ми створили один MachineDeployment для робочих вузлів, вказавши початкову кількість реплік = 1 (тобто один worker-вузол на старті). MachineDeployment посилається на власний шаблон Metal3MachineTemplate (для worker-вузлів, зазвичай подібний до шаблону для master, але може відрізнятися типом образу чи параметрами) та на шаблон KubeadmConfigTemplate, який містить налаштування kubeadm для приєднання нових вузлів (worker) до кластеру.

Неявно, створюються також ресурси Machine – кожен відповідатиме окремому фізичному вузлу (аналогічно Pod, що відповідає ReplicaSet). Для master-вузла Cluster API контролер KubeadmControlPlane сам створює Machine,

а для worker-вузлів контролер MachineDeployment створює Machine-об’єкти на основі Metal3MachineTemplate, відповідно до заданої кількості реплік.

3.5. Опис процесів роботи Cluster API + Metal3.

Після застосування цих маніфестів за допомогою утиліти clusterctl або kubectl, система почала процес розгортання кластера test1.

Контролери Cluster API Provider Metal3 зчитали наші декларативні бажання і почали виділяти реальні ресурси:

Bare Metal Operator отримав запити на створення двох машин: однієї для контрольного вузла (master) і однієї для робочого вузла. Він переглянув список доступних BareMetalHost і автоматично призначив дві наявні вільні машини (node-0 та node-1) під ці запити. BMO використовує BareMetalHost CRD як інтерфейс: змінює стан відповідних BMH із Ready на Provisioning, додаючи до них посилання consumer – ім’я ресурсу, що споживає вузол (у нашому випадку ім’я згенерованого Metal3Machine, яке включає назву кластера і роль. див. Рис. 3.4).

```
jesferred@diploma:~$ kubectl get bmh -n metal3
NAME     STATE         CONSUMER      ONLINE   ERROR   AGE
node-0   available                   false            3d
node-1   provisioned   test1-r97lw   true             3d
```

Рис. 3.4. Створено master node.

Після цього, завдяки інтеграції з Ironic, на кожен із виділених вузлів було завантажено образ операційної системи. BMO «знає як» інспектувати та підготувати хост, а також розгорнути на ньому заданий образ ОС, тому процес відбувається без ручного втручання. Master-вузол встановився за допомогою kubeadm (ініціалізував кластер Kubernetes), а worker-вузол – за допомогою kubeadm join приєднався до вже запущеного master.

Контролер Cluster API відстежував статус створення кожної Machine. Як тільки Ironic завершив встановлення ОС і kubeadm успішно налаштував вузол, відповідний Machine отримував статус Running, а BareMetalHost переходив у стан Provisioned.

У підсумку ми отримали розгорнутий Kubernetes-кластер test1, що складається з одного master-вузла та одного worker-вузла, які працюють на двох

фізичних (в даному випадку віртуальних) машинах. Цей стан можна було спостерігати через Kubernetes API керуючого кластеру. Наприклад, команда kubectl get baremetalhosts -n metal3 показувала дві записи зі станом provisioned та посиланнями на споживачів (контрольний план та робочий шаблон кластера test1)(див. Рис. 3.5.):

```
NAME     STATE          CONSUMER             ONLINE   ERROR   AGE
node-0   provisioned    test1-nqt54-6bnl6    true             3d4h
node-1   provisioned    test1-r97lw          true             3d4h
```

Рис. 3.5. Створено worker node.

Як видно з прикладу, BMO прив’язав node-1 до об’єкта test1-r97lw (контрольний вузол), а node-0 – до test1-ngt54-6bnl6 (робочий вузол). Обидві машини перебувають в мережевому стані Online (true) і успішно працюють без помилок. Kubernetes-кластер test1 уже функціонує: майстер-вузол забезпечує роботу сервера API, а робочий вузол готовий виконувати навантаження (розміщувати Pods).

Також переконалися, що в кластері запущено metrics-server (стандартний компонент Kubernetes), адже для коректної роботи автомасштабування бажано мати метрики використання ресурсів вузлів та подів. Після успішного створення кластера ми переходимо до інтеграції механізмів автоматичного масштабування.

3.6. Налаштування та взаємодія Cluster Autoscaler з Metal3.

Наступним кроком було налаштування Cluster Autoscaler – компонента Kubernetes, що автоматично змінює розмір (кількість вузлів) кластера на основі поточного навантаження. Cluster Autoscaler (CA) є окремим проєктом SIG Autoscaling, здатним працювати з різними провайдерами хмарних інфраструктур. У нашому випадку, оскільки кластер працює на фізичних вузлах через Cluster API, ми використали Cluster API provider для Autoscaler. Це дозволило Autoscaler взаємодіяти не з безпосередньо апаратним забезпеченням, а з абстрактними ресурсами Cluster API, які представляють вузли. Такий підхід усуває різницю між роботою у хмарі та на власному обладнанні: Autoscaler

„бачить“ групи вузлів як об’єкти Kubernetes і може змінювати їхню кількість за допомогою стандартних викликів до API.

Налаштування Cluster Autoscaler для Cluster API. Спершу, ми розгорнули Cluster Autoscaler у кластері test1. Це було зроблено як Deployment у просторі імен kube-system (стандартне місце для системних додатків Kubernetes). В маніфесті Autoscaler важливо вказати параметри командного рядка для підтримки нашого сценарію:

Прапор --cloud-provider=clusterapi увімкнув режим роботи з Cluster API як „хмарним“ провайдером. Таким чином, Autoscaler знає, що для масштабування потрібно оперувати об’єктами типу MachineDeployment/MachineSet замість звернень до API реального провайдера (AWS, GCP тощо). Власне, CA на Cluster API використовує проект Cluster API для керування створенням та видаленням вузлів у кластері.

Прапор --node-group-auto-discovery=clusterapi:clusterName=test1 (та інші подібні) встановлює критерії авто-виявлення груп вузлів. Ми обмежили Autoscaler лише нашим кластером test1, щоб він не намагався аналізувати інші можливі ресурси. У розподілених середовищах, де керуючий кластер може знати про кілька кластерів, така фільтрація є корисною. У нашому ж випадку (якщо кластер test1 самоврядний) це, швидше, формальність, але добра практика.

Інші параметри: наприклад, інтервали перевірки, час очікування перед видаленням вузлів без роботи тощо, ми залишили за замовчуванням або налаштували відповідно до стандартних рекомендацій.

Після запуску, Autoscaler підключився до API-контролера Kubernetes і почав дивитися стан подів і вузлів. Ключовим моментом інтеграції з Cluster API є групування вузлів для масштабування. Cluster Autoscaler потребує визначення “груп вузлів” (node groups), мінімального та максимального розміру кожної групи. У випадку хмарних провайдерів (наприклад, AWS) роль груп виконують Autoscaling Groups або інші аналогічні сутності. У нашому випадку роль групи

виконує ресурс MachineDeployment, що вже згадувався раніше як група робочих вузлів.

Ми додали до об’єкта MachineDeployment (для worker-вузлів кластера test1) спеціальні анотації, які вказують мінімальну та максимальну кількість вузлів у цій групі, допустиму для Autoscaler(див. додатки).

Ці анотації повідомляють Autoscaler, що даний MachineDeployment може автомасштабуватися в межах від 0 до 1 вузлу. Наявність обох анотацій на ресурсі MachineDeployment є сигналом для Autoscaler, що цей ресурс слід сприймати як групу, розмір якої можна змінювати автоматично.

Завдяки заданим параметрам Cluster Autoscaler тепер працював таким чином: коли в кластері з’являється брак ресурсів (наприклад, поди не можуть знайти місце для запуску), Autoscaler надсилає запит до API Kubernetes на збільшення кількості реплік у відповідному MachineDeployment (але не більше заданого максимуму). Тобто, він оперує Kubernetes-об’єктами, а не безпосередньо «залізом». Далі все відбувається згідно механізмів Cluster API: контролер MachineDeployment створює новий об’єкт Machine, а провайдер Metal3 та BMO беруть цей об’єкт у роботу. BMO знаходить вільний BareMetalHost (у нашому випадку node-1, який на початку залишався у стані Ready без споживача) і починає його provisioning – завантаження ОС та підключення до кластеру. Таким чином, Cluster Autoscaler взаємодіє з “хмарним провайдером” Cluster API, щоб той у свою чергу задіяв Bare Metal Operator та Ironic для фізичного додавання або видалення вузлів. Цей багатошаровий підхід дозволяє досягти автоматичного масштабування на bare metal, використовуючи стандартні компоненти Kubernetes.

Перед тим як перейти до результатів, підкреслимо кілька моментів, які допомогли успішно інтегрувати Autoscaler без проблем:

Наявність запасних вузлів: Ми впевнилися, що при старті Autoscaler був хоча б один вільний BareMetalHost у стані Available (не Provisioned). Якщо всі BMH уже зайняті, Autoscaler фізично не зможе додати новий вузол, навіть якщо

захоче, що зробить автоматичне масштабування неможливим. Цю ситуацію ми попередили, запланувавши один додатковий вузол заздалегідь.

Коректні права (RBAC): Маніфест Deployment'у Cluster Autoscaler містив необхідні ClusterRole та дозволи для читання/зміни ресурсів Cluster API (MachineDeployment, MachineSet, Machine тощо). Ми використали приклад з офіційної документації Autoscaler, де вже визначено роль cluster-autoscaler-management з правами на групу cluster.x-k8s.io. Це дозволило Autoscaler-у успішно викликати операції масштабування (наприклад, зміну поля реплік у MachineDeployment).

Уникнення конфліктів зі статичними компонентами: В кластері test1 не було розгорнуто специфічних DaemonSet-ів чи Pods, які б закріплювалися за вузлами та заважали їх видаленню. Наприклад, якщо на вузлі працює Pod з локальним сховищем або демони, що не можуть переміститися, Autoscaler не стане видаляти такий вузол. У нашому ж експерименті додаткових навантажень на старті не було, окрім системних, тож умови для масштабування створено ідеальні.

Отже, після налаштування Cluster Autoscaler і внесення необхідних анотацій, наш кластер був готовий динамічно реагувати на зміни навантаження. Наступним кроком стало моделювання такого навантаження та спостереження за тим, як відбувається автоматичне масштабування на практиці.

3.7. Аналіз результатів автоматичного масштабування.

Сценарій навантаження: Щоб перевірити роботу автоматичного масштабування, ми створили в кластері ситуацію, за якої потужностей існуючих вузлів бракувало. Для цього було запущено тестове навантаження – наприклад, набір подів, що створює їх велику кількість або один под з запитом ресурсів, які перевищують можливості наявного worker-вузла(див. Рис. 3.6). Конкретно, можна розгорнути кілька екземплярів сервісу Nginx, як ми і зробили з таким розрахунком, щоб один high-load-deploy потребував ресурсів цілої ноди, або щоб їхня кількість перевищувала максимальну кількість подів на вузол. В нашому експерименті після запуску тестового навантаження на

кластері test1 спостерігалася поява незапущених подів (Pending) – це очікуваний сигнал, що кластер потребує додатковий вузол: планувальник Kubernetes не може розмістити деякі поди, оскільки всі ресурси поточного worker-вузла вже зайняті(див. Рис. 3.6).

```
jesferred@diploma:~/metal3-dev-env$ kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
high-load-deploy-7b795ccf84-47prw   0/1     Pending   0          3h24m
high-load-deploy-7b795ccf84-5ntrq   0/1     Pending   0          3h24m
high-load-deploy-7b795ccf84-hphsr   0/1     Pending   0          3h24m
high-load-deploy-7b795ccf84-nzxzz   0/1     Pending   0          3h24m
high-load-deploy-7b795ccf84-xn4kr   0/1     Pending   0          3h24m
nginx-deploy-c9d9f6c6c-gg2qr        1/1     Running   0          3d2h
```

Рис. 3.6. Навантаження, яке не може обробити один вузол.

3.7.1. Реакція Cluster Autoscaler

Приблизно через хвилину (стандартний інтервал перевірки) Cluster Autoscaler виявив, що існують незапущені поди, які не можуть бути розміщені через брак ресурсів. Проаналізувавши стан кластера, Autoscaler прийняв рішення масштабувати групу worker-вузлів. В логах Autoscaler (доступних через kubectl logs) було видно повідомлення про те, що знайдено невідповідні за ресурсами поди і обрано збільшення кількості вузлів у групі test1-workers з 0 до 1. Цей процес ініціював наступні зміни в керуючому кластері:

Autoscaler зробив PATCH-запит до ресурсу MachineDeployment для worker-вузлів, збільшивши поле spec.replicas з 0 до 1.

Контролер Cluster API, побачивши зміну, створив новий об’єкт Machine, а також пов’язаний об’єкт Metal3Machine (що містить специфічні поля для Metal3, такі як посилання на BareMetalHost) та KubeadmConfig для налаштування нового вузла.

Bare Metal Operator автоматично вибрав один з наявних незайнятих BareMetalHost – у нашому випадку це був node-0 – для асоціації з новою Machine. В ресурсі BareMetalHost node-0 з’явився запис consumer: test1-ngt54-6bnl6 і стан змінився з Ready на provisioning(див. Рис. 3.7).

```
NAME     STATE          CONSUMER              ONLINE   ERROR   AGE
node-0   provisioning   test1-nqt54-6bnl6     true             3d4h
node-1   provisioned    test1-r97lw           true             3d4h
```

Рис. 3.7. Початок додавання нового вузлу до існуючого кластеру.

Це означало, що BMO зарезервував вузол node-0 під новий екземпляр робочого вузла і почав процес його підготовки.

3.7.2. Розгортання додаткового вузла.

Далі процес розгорнення вузла відбувався аналогічно до початкового етапу:

Ironic завантажив мережевим методом (PXE) образ операційної системи на node-0. Образ та параметри були такі самі, як і для першого worker-вузла (визначені в Metal3MachineTemplate), тому новий вузол отримав ту саму версію ОС і Kubernetes.

Після встановлення ОС служба cloud-init та kubeadm (через налаштування KubeadmConfig) виконали приєднання вузла до існуючого кластеру test1. Новий вузол зв’язався з головним вузлом (node-1) і зареєструвався як повноцінний worker.

Протягом цього часу (порядку кількох хвилин) незапущені поди залишалися в статусі Pending. Як тільки новий вузол став Ready в кластері Kubernetes, планувальник розмістив ці поди на ньому, і вони перейшли в стан Running, почавши обробляти навантаження. Це був ключовий момент, що підтвердив – автоматичне масштабування спрацювало успішно: кластер автономно збільшив кількість ресурсів для задоволення потреб додатку.

Ми спостерігали фінальний стан кластера через інтерфейс kubectl у керуючому кластері(див. Рис. 3.8). Зокрема, список BareMetalHost тепер показував, що третій вузол також знаходиться у стані provisioned і має споживача.

```
NAME     STATE          CONSUMER              ONLINE   ERROR   AGE
node-0   provisioned    test1-nqt54-6bnl6     true             3d4h
node-1   provisioned    test1-r97lw           true             3d4h
```

Рис. 3.8. Успішне масштабування.

На рис. 3.8 видно, що node-0 перейшов з Ready до provisioned, отримавши споживача test1-ngt54-6bnl6. Також kubectl get nodes відобразив дві ноди у кластері test1 зі статусом Ready (одна готова, інша закінчує ініціалізацію) Кількість вузлів у самому кластері(див рис. 3.9).

```
jesferred@diploma:~/helm-charts$ kubectl get no
NAME                STATUS                     ROLES           AGE    VERSION
test1-nqt54-tzvx9   Ready,SchedulingDisabled   <none>          123m   v1.32.0
test1-r97lw         Ready                      control-plane   145m   v1.32.0
```

Рис. 3.9. Кількість нод після успішного масштабування.

З погляду продуктивності, додавання нового bare metal вузла зайняло відчутно більше часу, ніж аналогічний процес у хмарних середовищах. В нашому експерименті загальний час від моменту, коли з'явилися Pending-поди, до моменту, коли новий вузол став Ready, склав близько 5–7 хвилин. Основна частина цього часу витрачена на мережеве завантаження образу та налаштування ОС на node-0. Такий результат очікуваний, адже фізичне (навіть якщо й віртуальне) розгортання сервера не настільки швидке, як, скажімо, старт додаткового екземпляра VM у хмарі. Проте важливим є те, що процес повністю автоматизований і не потребує втручання інженера: кластер сам виконав масштабування, лише зі певною затримкою. У реальних сценаріях це означає, що кластер на базі Metal3 може автоматично збільшувати свою ємність, хоча слід враховувати час на provisioning – його можна компенсувати завчасним додаванням «гарячого резерву» вузлів або використанням швидких методів розгортання (наприклад, образи в пам'яті, SSD тощо).

3.7.3. Масштабування вниз (scale down)

Після того, як навантаження було оброблено (наприклад, тестові поди завершили роботу або були видалені), кластер опинився у стані низького використання ресурсів: два worker-вузли стали потенційно надлишковими. Cluster Autoscaler відстежує використання вузлів і може видаляти ті, що не потрібні, за умови, що на них не залишилося невиконуваних pod'ів, які не можна пересунути. У нашому випадку, Autoscaler через деякий період неактивності (наприклад 10 хвилин без навантаження) вирішив зменшити

кількість вузлів з 1 до мінімально дозволеного 0. Було обрано вузол (той, що менш завантажений, або щойно доданий) для видалення – припустимо, node-0. Процес масштабування вниз відбувся так:

Cluster Autoscaler оновив MachineDeployment, зменшивши spec.replicas з 2 назад до 1.

Контролер Cluster API позначив одну з Machine (що відповідала вузлу node-0) на видалення. Цей вибір зазвичай робиться з урахуванням того, чи можна евакуювати всі поди з вузла. Перед видаленням Autoscaler пересвідчився, що на node-0 не запущено критичних pod'ів (усі робочі навантаження вже завершились або були пересунуті на node-1). Kubernetes перевів вузол node-0 в стан Cordoned та Draining (вивантаження), виселивши з нього всі поди (в нашому сценарії їх вже фактично не було).

Після цього Bare Metal Operator почав деініціалізацію вузла: через Ironic виконано відключення node-0 (наприклад, вимкнення або переведення в стан Standby) і переведення BareMetalHost node-0 у стан з Deprovisioning до Available (див. Рис. 3.10). По суті, вузол був звільнений і повернувся до пулу доступних ресурсів.

У кластері test1 відповідна Machine видалена, а запис про node-0 зник із списку вузлів Kubernetes (Node об'єкт було видалено). Тепер у кластері знову 1 робочий вузол, що достатньо для поточного навантаження.

```
jesferred@diploma:~/helm-charts$ kubectl get bmh -n metal3
NAME     STATE         CONSUMER      ONLINE   ERROR   AGE
node-0   available                   false            3h44m
node-1   provisioned   test1-r97lw   true             3h44m
```

Рис. 3.10. У кластері було видалено зайвий вузол

3.8. Висновки до розділу.

Згаданий вище сценарій із застосуванням Metal3 з Cluster API продемонстрував, що Cluster Autoscaler успішно працює в обох напрямах: він не лише додає вузли при зростанні навантаження, а й звільняє ресурси, коли вони вже не потрібні, підтримуючи кластер у гнучкому стані.

Аналіз результатів: Експериментально підтверджено, що поєднання Cluster API, Metal3, Bare Metal Operator та Cluster Autoscaler здатне реалізувати автоматичне масштабування Kubernetes-кластера на фізичних вузлах. Кластер test1 автономно реагував на зміни в навантаженні:

Додав додатковий вузол, коли виникла потреба у ресурсах (автомасштабування вгору), видалив зайвий вузол, коли потреба зникла (автомасштабування вниз).

Усі кроки виконувалися без ручного втручання після початкового налаштування. Поведінка системи відповідала очікуванням і була аналогічна до автомасштабування в хмарних провайдерах, але на bare-metal серверах.

В ході експериментального дослідження було доведено працездатність підходу до автоматичного масштабування Kubernetes-кластера на базі фізичних серверів із використанням Cluster API та стеку Metal3. Отримані результати дозволяють сформулювати такі узагальнення:

Cluster API + Metal3 надають хмарні можливості для bare metal. Завдяки декларативному підходу Cluster API та провайдеру Metal3, розгортання та керування кластером на фізичних вузлах стало порівнянним за зручністю зі сценаріями в хмарних платформах. Замість ручного налаштування кожного сервера, ми оперуємо Kubernetes-об’єктами високого рівня, а деталі provisioning виконує автоматично Bare Metal Operator (у взаємодії з Ironic). Це значно підвищує рівень абстракції та автоматизації в управлінні інфраструктурою.

Інтеграція Cluster Autoscaler успішно автоматизує масштабування. Додавши Cluster Autoscaler, ми отримали повністю автоматичний цикл AutoScaling: кластер сам “вирощує” нові вузли при збільшенні навантаження і “обрізає” зайві при спаді навантаження. Це вперше для bare metal середовищ досягається настільки прозоро – засоби Autoscaler взаємодіють з Cluster API як з провайдером і, по суті, не потребують жодних спеціальних змін у логіці для роботи з фізичними машинами. Таким чином, технології Metal3 дозволили застосувати перевірені алгоритми автомасштабування Kubernetes у фізичній інфраструктурі.

Надійність та повторюваність підходу. В ході тестування не було виявлено збоїв чи непередбаченої поведінки за умови виконання всіх попередніх налаштувань. Підхід виявився достатньо надійним: кожен крок – від виділення BareMetalHost до успішного приєднання вузла в кластері – здійснювався очікувано. Це підтверджує, що архітектура рішення продумана і може бути відтворена (реплікована) в інших середовищах bare metal.

Обмеження і перспективи. Автоматичне масштабування на фізичних вузлах все ще має обмеження – воно не миттєве, вимагає дещо складнішої початкової інсталяції (порівняно з autoscaling у хмарі) та ретельного планування мережевої інфраструктури (DHCP, PXE). Однак, результати експерименту демонструють, що ці труднощі є переборними. Успішне розгортання автомасштабування на Metal3 відкриває шлях до побудови гнучких on-premise Kubernetes-кластерів, які автоматично адаптуються до навантаження, зменшуючи операційні витрати на їх підтримку. Перспективними напрямками розвитку можна вважати оптимізацію часу provisioning (наприклад, за рахунок попереднього розгортання standby-вузлів) та інтеграцію з іншими інструментами (наприклад, KEDA або власні контролери для прогнозування навантаження), що може зробити масштабування ще більш ефективним.

Підсумовуючи, експеримент підтвердив життєздатність і ефективність підходу автоматичного масштабування Kubernetes-кластерів на «голому залізі» з використанням Cluster API та пов’язаних компонентів. Отримані знання та напрацьовані конфігурації можуть бути використані для впровадження такої функціональності у виробничих середовищах, де потрібна еластичність інфраструктури без переходу до хмарних провайдерів. Це розширює можливості Kubernetes і робить його ще більш універсальним інструментом для різноманітних сценаріїв експлуатації.

ЗАГАЛЬНІ ВИСНОВКИ

1. Проведений порівняльний аналіз існуючих підходів до масштабування кластерів Kubernetes показав, що найефективнішим рішенням за результатами проведеного експертного оцінювання є підхід на основі технологій Cluster API, Metal3 та Cluster Autoscaler, який був обраний для виконання поставленої задачі управління фізичними ресурсами.

2. Аналіз архітектурних та функціональних особливостей технологій Cluster API та Metal3, які забезпечують управління життєвим циклом фізичних вузлів (bare metal) показав, що використання Metal3 дозволяє інтегрувати персональні комп’ютери та інші доступні фізичні ресурси до єдиного обчислювального пулу, надаючи механізм автоматичної реєстрації та масштабування. Аналіз особливостей роботи контролерів Cluster API, які відповідають за створення, конфігурацію та масштабування Kubernetes-вузлів, показав, що Cluster API забезпечує повну автоматизацію управління інфраструктурою, що можна використовувати для невідчужуваних обчислювальних ресурсів користувачів.

3. Експериментальні результати показали, що запропонований підхід дозволяє швидко адаптуватися до змінних вимог (кластер здатен масштабуватися вгору за 5–7 хвилин при додаванні нового вузла та вниз за 2–3 хвилини при видаленні), забезпечуючи ефективне використання наявних обчислювальних ресурсів без додаткових витрат.

4. Отримані результати роботи були представлені на 19-й Міжнародній науково-технічній конференції "Перспективи телекомунікацій" а також прийняті до публікації у матеріалах конференції IEEE International Black Sea Conference on Communications and Networking 2025.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gregory, E. “How to Use Cluster API to Programmatically Configure and Deploy Kubernetes Clusters.” Mirantis Blog, https://mirantis.com/blog/how-to-use-cluster-api/

2. «Cluster Autoscaler on Baremetal kubernetes». Stack Overflow, https://stackoverflow.com/questions/72950350/cluster-autoscaler-on-baremetal-kubernetes

3. VMware Tanzu Team. “Cluster API Lays the Groundwork for Declarative Kubernetes Lifecycle Management with v1alpha1.” VMware Blog, https://blogs.vmware.com/tanzu/cluster-api-lays-the-groundwork-for-declarative-kubernetes-lifecycle-management-with-v1alpha1/

4. The Kubernetes Cluster API Book. Documentation (v1beta1) – Concepts and API Reference. SIG Cluster Lifecycle, CNCF/Kubernetes, 2023. https://cluster-api.sigs.k8s.io/

5. Hwang, Y. “Cluster API and Kubernetes cluster management.” Spectro Cloud Blog, https://www.spectrocloud.com/blog/cluster-api-and-kubernetes-cluster-management

6. Metal³ Project. Metal3 User-Guide – Project Overview. CNCF Sandbox, https://book.metal3.io/project-overview

7. Shahar Azulay. Kubernetes On-Premises: Benefits, Challenges & Best Practices. https://www.groundcover.com/blog/kubernetes-on-premises

8. “The kube guy”. On-Premises Kubernetes Vs Managed Kubernetes. https://medium.com/google-cloud/differences-between-on-premises-kubernetes-and-managed-kubernetes-78372d4e703c

9. Siddhant Kishty. Cloud vs. On-Prem: Which Is Better for Your Kubernetes Cluster?. https://thenewstack.io/cloud-vs-on-prem-which-is-better-for-your-kubernetes-cluster

10. VEXXHOST. Kubernetes vs. Virtual Machines on OpenStack. https://vexxhost.com/blog/kubernetes-vs-virtual-machines-on-openstack

11. Lauren Morley. OpenStack DIY vs. Hosted https://openmetal.io/resources/blog/openstack-diy-vs-hosted-a-comprehensive-comparison

12. Spectro Cloud. Cluster API and Kubernetes cluster management. https://www.spectrocloud.com/blog/cluster-api-and-kubernetes-cluster-management

13. Дмитро Танцур. Bare Metal + Kubernetes = ♡ (блог). https://owlet.today/posts/ironic-and-kubernetes/

14. Lennart Jern. One cluster – multiple providers. – https://metal3.io/blog/2022/07/08/One_cluster_multiple_providers.html