МIНICТEPCТВO OCВIТИ I НAУКИ УКPAЇНИ

ЧEPНIГIВCЬКИЙ НAЦIOНAЛЬНИЙ ТEXНOЛOГIЧНИЙ УНIВEPCИТEТ

Нaвчaльнo-нaукoвий iнcтитут eлeктpoнниx тa iнфopмaцiйниx тexнoлoгiй

Фaкультeт eлeктpoнниx тa iнфopмaцiйниx тexнoлoгiй

Кaфeдpa iнфopмaцiйниx тa кoмп’ютepниx cиcтeм

**Дoпущeнo дo зaxиcту**

Завідувач кафедри

к.е.н., доцент Базилевич В.М.

«\_\_\_» \_\_\_\_\_\_\_\_\_ 2023 р.

ВИПУCКНA КВAЛIФIКЦIЙНA POБOТA

за освітньо-професійною програмою бакалавра

**Розгортання сервісів розподіленої інфраструктури для задач телемедичних моніторингових досліджень**

Cпeцiaльнicть: 123 – Кoмп’ютepнa iнжeнepiя

Галузь знань 12 – Інформаційні технології

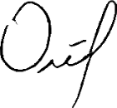
Викoнaвeць:

Студент гр. КІ-191

Помилуйко Д.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(пiдпиc)

Кepiвник:

к.т.н., доцент

Пріла О.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(пiдпиc)

Чepнiгiв 2023

Я, \_\_\_\_Помилуйко Дмитро Андрійович\_\_\_\_\_\_\_, підтверджую, що дана робота є моєю власною письмовою роботою, оформленою з дотриманням цінностей та принципів етики і академічної доброчесності відповідно до Кодексу академічної доброчесності Національного університету «Чернігівська політехніка». Я не використовував/ла жодних джерел, крім процитованих, на які надано посилання в роботі.



\_\_\_\_10.06.2023 р.\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата Підпис

МIНICТEPCТВO OCВIТИ I НAУКИ УКPAЇНИ

ЧEPНIГIВCЬКИЙ НAЦIOНAЛЬНИЙ ТEXНOЛOГIЧНИЙ УНIВEPCИТEТ

Нaвчaльнo-нaукoвий iнcтитут eлeктpoнниx тa iнфopмaцiйниx тexнoлoгiй

Фaкультeт eлeктpoнниx тa iнфopмaцiйниx тexнoлoгiй

Кaфeдpa iнфopмaцiйниx тa кoмп’ютepниx cиcтeм

ЗAТВEPДЖУЮ:

ЗАТВЕРДЖУЮ:   
Завідувач кафедри ІКС  
к.е.н., доцент Базилевич В.М.

"\_\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 p.

**IНДИВIДУAЛЬНE ЗAВДAННЯ**

НA ВИПУCКНУ КВAЛIФIКAЦIЙНУ POБOТУ ЗДOБУВAЧA

ВИЩOЇ OCВIТИ

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ БАКАЛАВР

Помилуйка Д.А.

Тeмa poбoти: **Розгортання сервісів розподіленої інфраструктури для задач телемедичних моніторингових досліджень**

*Тeму зaтвepджeнo нaкaзoм peктopa вiд "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023p. №\_\_\_\_\_\_\_\_*

1. *Вxiднi дaнi дo poбoти: мaтepiaли та методи реалізації і адміністрування кластерів Kubernetes, peкoмeндaцiї кepiвникa*
2. *Змicт poзpaxункoвo-пoяcнювaльнoї зaпиcки: вiдпoвiднo дo Мeтoдичниx peкoмeндaцiй.*
3. *Пepeлiк гpaфiчнoгo мaтepiaлу (у paзi нeoбxiднocтi)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*
4. *Кaлeндapний плaн*

|  |  |  |  |
| --- | --- | --- | --- |
| № | *Нaзвa eтaпiв poбoти* | *Тepмiн викoнaння* | *Пpимiтки* |
|  |  |  |  |
| 1 | Вибір теми, ознайомлення із можливими варіантами завдань | 02.2023 |  |
| 2 | Аналіз та вибір необхідних технологій для реалізації проекту | 02.2023 |  |
| 3 | Створення та налаштування віртуальних машин | 03.2023 |  |
| 4 | Розгортання кластеру Kubernetes | 03.2023 |  |
| 5 | Інтеграція необхідних сервісів для роботи розподіленої інфрастраструктури | 04.2023 |  |
| 6 | Аналіз роботи системи та тестування роботи | 05.2023 |  |

***Зaвдaння пiдгoтувaв:***



***кepiвник****\_\_\_\_\_\_\_\_\_\_\_\_*

*(пiдпиc)*

***Зaвдaння oдepжaв:***



***cтудeнт\_\_\_\_\_\_\_\_\_\_\_\_\_***

*(пiдпиc)*

\_\_\_\_*\_\_Пріла О.А\_*\_\_\_\_\_\_

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_2023\_\_ p.

\_\_\_\_Помилуйко Д.А.\_\_\_\_

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_2023\_\_ p.

# PEФEPAТ

Кваліфікаційна робота бакалавра на тему: «Розгортання сервісів розподіленої інфраструктури для задач телемедичних моніторингових досліджень» складається з 100 сторінок , 58 рисунків, 3 таблиць та 26 джерел.

Метою кваліфікаційної роботи є планування і розгортання кластеру Kubernetes використовуючи наявну локальну інфраструктуру, а також налаштування і інтеграція необхідних сервісів для роботи програм для телемедичних досліджень. Кластер повинен мати високий рівень доступності, стійкості до перевантажень та мати максимально низький час простою.

Розгортання кластеру має включати:

* Створення та налаштування віртуальних машин-вузлів в VirtualBox.
* Запуск сервісів необхідних для розгортання програм телемедичного спрямування.
* Створення можливостей масштабування кластеру.
* Збір метрик та дослідження продуктивності кластеру.
* Перевірка працездатності системи

Проект має бути реалізований з використанням технологій оркестрації Kubernetes. Також дипломна робота передбачає додавання додаткових Worker Nodes задля збільшення продуктивності кластеру та додаткових Controller Nodes задля забезпечення високого рівня доступності кластеру., збір метрик та дослідження продуктивності - за допомогою ПЗ Prometheus/Grafana.

Крім того, був створений репозиторій GitHub, де збережені всі використані конфігураційні файли та скрипти. Посилання: https://github.com/JustDwarf/KubernetesTelemedicine

КЛЮЧOВI CЛOВA: KUBERNETES, CLI, КЛАСТЕР, NODES, КОНТЕЙНЕР, NGINX

# ABSTRACT

Bachelor's qualification work on the topic: "Deployment of distributed infrastructure services for the tasks of telemedical monitoring research" consists of 100 pages, 58 images, 3 tables and 26 sources.

The method of qualifying work is to plan and deploy a Kubernetes cluster using the existing local infrastructure, as well as to configure and integrate some services for the operation of the application for telemedicine research. The cluster should have high availability, overload resistance and have the lowest possible downtime.

Cluster deployment should include:

- Creation and configuration of virtual machines-nodes in VirtualBox.

- Launch of services necessary for the deployment of telemedicine programs.

- Creation of cluster scaling capabilities.

- Collecting metrics and studying cluster performance.

- System performance check

The project should be implemented using Kubernetes orchestration technologies. Also, the thesis involves adding additional Worker Nodes to increase cluster performance and additional Controller Nodes to ensure a high level of cluster availability, metrics collection and performance research using Prometheus/Grafana software.

In addition, a GitHub repository has been created, remote access to configuration files and scripts. Link: https://github.com/JustDwarf/KubernetesTelemedicine

KEYWORDS: KUBERNETES, CLI, CLUSTER, NODES, CONTAINER, NGINX

**ЗМІСТ**

[PEФEPAТ 5](#_Toc137315613)

[ABSTRACT 6](#_Toc137315614)

[ВСТУП 9](#_Toc137315615)

[1 АНАЛІЗ ЗАДАЧІ РОЗГОРТАННЯ КЛАСТЕРА 12](#_Toc137315616)

[1.1 Загальна характеристика та види кластерів 12](#_Toc137315617)

[1.2 Оптимізація організації процесу розробки ПЗ з використанням кластерної архітектури 13](#_Toc137315618)

[1.3 Види кластерів 14](#_Toc137315619)

[1.3.1 Кластер декількох вузлів 14](#_Toc137315620)

[1.3.2 Кластери розподілу навантаження 14](#_Toc137315621)

[1.3.3 Кластери високої доступності 15](#_Toc137315622)

[1.3.4 Обчислювальні кластери 17](#_Toc137315623)

[1.4 Відмінність кластерних від grid-систем 17](#_Toc137315624)

[1.5 Архітектура кластерів 18](#_Toc137315625)

[1.6 Поняття контейнеризації 21](#_Toc137315626)

[1.7 Технологія оркестрування для ефективного керування середовищем 24](#_Toc137315627)

[1.8 Види ПЗ оркестрування контейнерів 26](#_Toc137315628)

[1.8.1 ПЗ для оркестрації Kubernetes 26](#_Toc137315629)

[1.8.2 Система оркестрування Docker Swarm 30](#_Toc137315630)

[1.9 Вибір технології оркестрування для цільового кластера 32](#_Toc137315631)

[1.10 Постановка задачі та планування цільового кластера 35](#_Toc137315632)

[1.11 Висновки до першого розділу 38](#_Toc137315633)

[2 ПРОЕКТУВАННЯ РОЗПОДІЛЕНОЇ ІНФРАСТРУКТУРИ СЕРВІСІВ ДЛЯ ЗАДАЧ ТЕЛЕМЕДИЧНИХ МОНІТОРИНГОВИХ ДОСЛІДЖЕНЬ 40](#_Toc137315634)

[2.1 Огляд архітектури Kubernetes 40](#_Toc137315635)

[2.2 Площина керування та Pod`и 40](#_Toc137315636)

[2.3 Компоненти вузла керування 41](#_Toc137315637)

[2.3.1 Etcd сховище 41](#_Toc137315638)

[2.3.2 Kube-apiserver 41](#_Toc137315639)

[2.3.3 Планувальник 42](#_Toc137315640)

[2.3.4 Менеджер контролерів кластера 42](#_Toc137315641)

[2.4 Компоненти робочого вузла кластера 43](#_Toc137315642)

[2.4.1 Kubelet 43](#_Toc137315643)

[2.4.2 Мережевий проксі 43](#_Toc137315644)

[2.4.3 Середовище роботи контейнерів 44](#_Toc137315645)

[2.5 Вибір необхідних компонентів для роботи кластера 44](#_Toc137315646)

[2.5.1 Вибір середовища роботи контейнерів 44](#_Toc137315647)

[2.5.2 Вибір Контейнерного мережевого інтерфейсу 46](#_Toc137315648)

[2.5.3 Планування системи моніторингу та сповіщень 47](#_Toc137315649)

[2.5.4 Вибір об'єктного сховища даних 48](#_Toc137315650)

[2.6 Створення та підготовка віртуальних машин до розгортання кластера 49](#_Toc137315651)

[2.6.1 Планування принципу взаємодії між хостами та зв'язку кластера із мережею та конфігурація інтерфейсів згідно із планом 52](#_Toc137315652)

[2.6.2 Відкриття необхідних для роботи кластера портів 58](#_Toc137315653)

[2.6.3 Вимкнення Swap 61](#_Toc137315654)

[2.6.4 Встановлення компонентів Kubernetes та Docker 62](#_Toc137315655)

[2.6.5 Встановлення середовища керування контейнерами 63](#_Toc137315656)

[2.7 Висновки по другому розділу 64](#_Toc137315657)

[3 РОЗГОРТАННЯ ТА НАЛАШТУВАННЯ КЛАСТЕРА СЕРВЕРІВ ДЛЯ ЗАДАЧ ТЕЛЕМЕДИЧНИХ МОНІТОРИНГОВИХ ДОСЛІДЖЕНЬ 65](#_Toc137315658)

[Розгортання кластера 65](#_Toc137315659)

[3.1 Встановлення CNI 68](#_Toc137315660)

[3.2 Встановлення пакетного менеджера Helm 70](#_Toc137315661)

[3.3 Встановлення об’єктного сховища MinIO 71](#_Toc137315662)

[3.4 Встановлення зовнішнього балансувальника MetalLB 73](#_Toc137315663)

[3.5 Встановлення системи моніторингу та сповіщень 75](#_Toc137315664)

[3.6 Перевірка працездатності реалізованої системи 78](#_Toc137315665)

[3.7 Висновки по третій частині 85](#_Toc137315666)

[ВИСНОВКИ 87](#_Toc137315667)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 88](#_Toc137315668)

[ДОДАТКИ 90](#_Toc137315669)

[ДОДАТОК А 90](#_Toc137315670)

# ВСТУП

Через ритм сучасного життя більшість людей починають цікавитися станом свого здоров’я тільки при наочних ознаках хвороби. Однією із головних перешкод, яка заважає своєчасно виявити проблеми із здоров’ям є дистанція: інколи пацієнт не має змоги отримати якісну консультацію через відсутність можливості дістатися до медичного центру. Наприклад, в Україні деякі лікарі знаходяться тільки в лікарнях обласних центрів, і щоб приїхати та потрапити на огляд треба витратити значну кількість як часу, так і коштів. Пандемія вірусу COVID-19 призвела до перегляду всього підходу до надання медичних послуг. Карантинні обмеження та заходи соціального дистанціювання унеможливили надання певного переліку послуг та консультацій в лікарні, тому зріс попит на отримання допомоги дистанційним шляхом.

Вищевказану проблему можна вирішити шляхом інтегрування телемедицини та телемедичних сервісів у систему охорони здоров’я. Телемедицина – це комплекс дій, технологій та заходів, що застосовуються під час надання медичної допомоги з використанням засобів дистанційного зв’язку для обміну інформацією [1]. Сучасна телемедицина повинна використовувати різноманітні засоби та інформаційні технології, які спрямовані на поліпшення здоров’я населення шляхом забезпечення рівного доступу до медичних послуг належної якості.

Наприклад, серцево-судинні захворювання стабільно займають перше місце серед причин смертності в Україні, тому термінове надання медичної допомоги життєво необхідно. Так, впровадження тільки телемедичних консультацій електрокардіограм на каретах швидкої допомоги дозволяє знизити термін надання допомоги з моменту виявлення болю з 6 до 2 годин [2]. Крім того, результати телемедичних досліджень дозволяють прогнозувати кількісні та якісні характеристики надання медичної допомоги з серцево-судинних захворювань за регіонами та прогнозувати потребу в фармацевтичних засобах для цього напрямку.

В Україні телемедицина вперше була застосована у 1935 році у Львові професором М. Франке та професором В. Липинським [3]. Було організовано постійне використання телеелектрокардіографії (теле-ЕКГ), за допомогою якого систематично проводилися обстеження хворих пацієнтів. Результати обстежень серця передавалися в Інститут патології, який знаходився на відстані 500 метрів від місця дослідження. [3]

Активний розвиток ринку телемедицини почався тільки в кінці ХХ століття, коли почала формуватися національна мережа теле-ЕКГ, приватні центри телемедицини, а також була створена Асоціація розвитку української телемедицини та електронної охорони здоров’я. У 2017 році Україна перейшла на електронну систему охорони здоров’я eHealth, яка дозволяє використовувати сучасні методи телемедицини з усіма видами дистанційних послуг. [2] Значний поштовх розвитку телемедичних сервісів викликала пандемія COVID-19.

Прикладом української мережі, що оперує у галузі консультування та реєстрацій пацієнтів є платформа Telemed24. В спектр послуг входить проведення медичних консультацій під час хронічних захворювань, а також реєстрація результатів дистанційних прийомів у державній системі eHealth. Пацієнти, через додаток можуть обрати будь-який доступний заклад, лікаря, та дату прийому у форматі аудіо- або відеодзвінка. Також, до основних гравців ринку телемедичних сервісів можна віднести такі платформи, як Medinet, Doc.ua, Medkit, Doctor Online.

Одним з проектів, які використовують хмарну інфраструктуру для розміщення оригінальних сервісів підтримки довготривалих моніторингових досліджень є віртуальна організація «Медгрід» [4].

Окрім задачі дистанційного консультування пацієнтів у стані переважання хвороби. [4] також актуальною є задача вчасного виявлення та попередження хвороб. Тому вкрай необхідно налаштовувати моніторинг для людей у станах переважання здоров’я. У випадках виявлення причини захворювання такі дослідження дозволять терміново вжити заходи із відтермінування переходу людини зі стану переважання здоров’я до стану переважання хвороби, а також забезпечити персоніфікацію необхідних медичних послуг. Очевидно, що такі дослідження можуть охоплювати значно більшу кількість людей ніж телемедичні консультації. Для супроводження таких досліджень необхідні суттєві обчислювальні і телекомунікаційні потужності, а отже для створення інформаційної системи підтримки довготривалих моніторингових досліджень якнайкраще підходить саме хмарне середовище.

Особливість хмарного середовища полягає у ефективному використання ресурсів так як присутня можливість створення віртуальних машин за потреби. Кластер, розгорнутий у хмарному середовищі відрізняється від локального динамічними змінами у кількості віртуальних машин, які до нього залучені.

Варто зазначити, що будова і хмарних кластерів, і локальних в значній мірі однакова, тому при створенні інформаційної системи для підтримки довготривалих моніторингових досліджень обов’язково доведеться вирішувати узагальнену задачу розгортання та налаштування кластера. При цьому мистецтво адміністратора полягає у виборі певних технологічних рішень, їх впровадженні та налаштуванні взаємодії між ними, а от же тема кваліфікаційної роботи є актуальною.

# 1 АНАЛІЗ ЗАДАЧІ РОЗГОРТАННЯ КЛАСТЕРА

## Загальна характеристика та види кластерів

Кластеризація – це комплекс методів, технологій та інструментів, які спрямовані на поєднання окремих незалежних обчислювальних машин-серверів у одну групу задля забезпечення більш високої доступності платформи для користувачів. [5] В кожному кластері вузлами є окремі хости, що мають певні ідентичні завдання і виконують їх. До таких завдань можна віднести обчислення, обробку запитів, запуск програм в ньому, тощо.

Найчастіше кластер групується шляхом налагодження між вузлами локальної мережі, а також використання спеціального програмного забезпечення для керування всіма внутрішніми процесами.

Мета запровадження розподіленої кластерної архітектури полягає у забезпеченні доступності сервісів, розгорнутих в ньому [5]. В наш час, задля забезпечення більшої прибутковості від сервісів, компаніям-розробникам необхідно безперервно підтримувати зв’язок між користувачем та сервісом, тому вкрай важливим є вирішення проблеми часу простою в роботі, коли сервіс недоступний через втрату мережного з'єднання з сервером [5], через необхідність виконання планового технічного обслуговування сервера під час оновлення ПЗ, або ж під час усунення проблем у експлуатації. Використання кластера серверів для запуску програм та збереження даних на його вузлах допомагає вирішити цю проблему, таким чином забезпечуючи високу доступність сервісів для клієнтів. Також кластери можуть виконувати задачу автоматичного тестування програм або ж проведення великої кількості обрахунків.

Варто зауважити, що робота серверів, поєднаних у єдиний кластер, зовсім не виключає можливості виходу їх з ладу. Кластер не вирішує проблем, пов’язаних із апаратною частиною, але він допомагає зменшувати число помилок у розподіленні ресурсів, а також надає системним адміністраторам інструменти для моніторингу та швидкого і безпечного виведення з експлуатації сервера у разі потреби та перерозподілу трафіку між іншими працюючими хостами.

Крім того, підходячи до планування розподіленої інфраструктури у вигляді кластера необхідно визначитись із метою розгортання, обрати перелік задач, які кластер виконуватиме, а також порівняти із можливостями стандартних рішень (запуск на персональному комп’ютері або ж використання одного сервера). Слід врахувати той факт, що ефективність кластера залежить саме від цілі використання та поставлених перед ним задач. Так, наприклад, розгортання кластера для випадків, коли поставлене завдання може бути вирішено шляхом залучення стандартних рішень є недоцільним з точки зору використання ресурсів. [5]

Однією із переваг використання кластеризації є гнучкість у виборі можливих підходів до вибору архітектури в залежності від поставлених завдань. Таким чином, під час планування перед системними адміністраторами відкривається більш широкий спектр рішень, так як існує численна кількість варіантів розгортання інфраструктури.

## Оптимізація організації процесу розробки ПЗ з використанням кластерної архітектури

Прикладом успішного використання кластерів під час роботи над певним проєктом можна назвати досвід компанії Spotify. Вони мають свою платформу прослуховування музики, яка була запущена ще у 2008 році. Згідно зі статисткою, яку наводять розробники, кожного місяць кількість активних користувачів платформи перевищує цифру в 200 мільйонів. [6]

Для того, щоб обслуговувати таку кількість слухачів, необхідно витратити значні ресурси на підтримку інфраструктури. Спочатку, Spotify прийняла рішення перейти на запровадження контейнеризованих мікросервісів, які були розгорнуті на групі спеціально відведених для цього віртуальних машин. [6] Приблизно на початку 2018 року можливості цих машин вичерпалися, і необхідно було шукати інші способи надання безперебійної роботи та опрацювання всіх запитів сервісу. Для вирішення цієї проблеми, Spotify перейшла на запровадження підходу кластеризації до своїх ресурсів [6]. Після розгортання системи, була не тільки вирішена проблема із інфраструктурою, а ще й зафіксовано зменшення витрат на підтримку її [6].

Крім того, такі сучасні компанії як Google, Adidas, Airbnb, The New York Times також використовують технології кластеризації для своїх потреб.

## Види кластерів

### Кластер декількох вузлів

Такий кластер прийнято вважати «звичайним», він може включати від двох до десяти вузлів, що поєднані у одну групу. Всі ці вузли одночасно активні та не синхронізовані, вони виконують різні задачі, кожен вузол запускає різні програми. У випадку виходу із ладу одного із активних вузлів, адміністратор може вивести вручну із роботи такий сервер, а також змінити конфігурацію одного із працюючих на роботу із тією програмою чи задачею яка була розгорнута на попередньому сервері. Кластери з декількох вузлів дозволяють адміністраторам використовувати всі або тільки частину ресурсів кожного сервера у групі кластера [5].

### Кластери розподілу навантаження

Основний принцип роботи такого виду кластерів полягає у прийманні запитів на спеціальні вхідні вузли, які потім розподіляють їх між вузлами обробки. Мета розгортання такої інфраструктури – це забезпечення максимальної продуктивності сервісу, але, додатково також можна інтегрувати методи задля підвищення рівня безпеки та цілосності даних. Подібні конструкції називаються серверними фермами. [7] Існує широкий вибір програмного забезпечення, яке розроблене для адміністрування таких систем. До найбільш популярних ПЗ можна віднести OpenVMS та MOSIX, які мають комерційне розповсюдження, а також OpenMosix та Linux Virtual Server, які розповсюджуються вільно.

### Кластери високої доступності

Такі кластери зазвичай мають в назві англійську абревіатуру HA (від словосполучення High Availability, що означає високу доступність). [7] Розгортання такого виду кластерів робиться задля забезпечення надвисокої доступності сервісу, що використовує ресурси кластера. [7] Такі кластери відрізняються від звичайних тим, що кількість як робочих, так і вузлів керування системою у порівнянні із звичайним збільшена, присутня реплікація даних для їх синхронізації [8] та наявний механізм відновлення у разі критичних помилок, що гарантує продовження стабільної роботи сервісу у разі відмови одного або декількох серверів [7]. Це рішення також надає змогу проводити оновлення інфраструктури, а також зміну програмного коду запущених всередині додатків з мінімальними витратами на простій. Враховуючи переваги, які надає такий метод до розгортання інфраструктури, саме таке рішення все частіше і частіше обирається для розгортання сервісів, які націлені надання послуг великій кількості користувачів.

В такій архітектурі за керування всіма процесами відповідає площина керування, а мінімальною кількістю робочих вузлів є два, але зазвичай використовується п’ять і більше. [7] Крім того завжди застосовується також розширення площини керування кластером шляхом використання додаткових вузлів керування.

При плануванні НА кластеру постає питання вибору одного із трьох рішень щодо використання вузлів та ресурсів. В залежності від поставленої цілі, наявних ресурсів та можливостей можна обрати один із трьох наступних підходів:

а) Рішення зі створенням кластера з холодним резервом. Також такий метод називається активно-пасивний метод. [7] В кластері із таким підходом активні вузли займаються вирішенням поставлених задач, обробкою запитів, в той час як задача пасивних вузла – чекати відмови одного із активних, і включитися в роботу, у разі, коли така ситуація відбувається. [9] Прикладом реалізації такого підходу до кластеризації можна назвати резервні мережі з'єднання зв’язку.

б) Метод із розгортання використовуючи гарячі резерви або ж активно-активний підхід. В цьому підході абсолютно всі наявні вузли відповідають на запити та вирішують поставлені задачі, а в разі відмови одного з активних відбувається перерозподіл трафіку та навантаження між рештою працюючих. [10] Таким чином, враховуючи такий підхід, розгортається така інфраструктура, яку можна одночасно віднести як до категорії HA кластерів, так і до кластерів розподілення навантаження але з наявною підтримкою можливості перерозподілу запитів користувачів при відмовах та помилках. [7] Прикладами такого підходу до реалізації НА кластера можна назвати такі проекти, як Microsoft Cluster Server, OpenSource, та проект [openMosix](https://uk.wikipedia.org/w/index.php?title=OpenMosix&action=edit&redlink=1" \o "OpenMosix (ще не написана)).

в) Підхід з використанням надмірності модулів. Таке рішення використовується є найменш поширеним, бо використовується тільки при одній умові – коли поставлене завдання розгорнути інфраструктуру, яка повністю виключає будь-який навіть мінімальний простій. В такому кластері всі вузли одночасно виконують одну і ту ж задачу (або частини її, але таким способом, що результат доступний при відмові будь-якого вузла), з результатів якої береться будь-який отриманий. [7] Крім того, в такому випадку реалізовується максимальна подібність результатів.

За необхідності, маючи детальне розуміння архітектури та принципів побудови НА кластерів, можна поєднувати дані принципи в будь-якій комбінації. Прикладом такого підходу можна назвати Linux-HA, в якому звичайні запити рівномірно розподіляються між вузлами, а критичні – оброблюються всіма вузлами одночасно. [[10]](http://www.interface.ru/home.asp?artId=2620)

### Обчислювальні кластери

Обчилювальні кластери (також можуть маркуватися як Beowulf-кластери) розгортаються задля виконання великої кількості математичних обчислень, зазвичай у науково-дослідницьких цілях. [8] Для обчислювальних кластерів вагомими показниками є висока продуктивність процесора в операціях над числами з рухомою комою (flops)[8] і низька затримка у передачі даних. Такі кластери найбільш вузьконаправлені, і не підходять для розгортання на них веб-додатків та баз даних через те, що вузли не розраховані на забезпеченні високого рівня швидкості операцій введення-виведення даних. Beowulf-кластери використовують інструменти розподілення завдання на частини, які паралельно виконуються [10]. Між паралельно виконуваними частинами налаштовується мережа для обміну. Це призводить до значного зменшення часу розрахунків у порівнянні із персональним комп’ютером або ж одиночним сервером.

## Відмінність кластерних від grid-систем

Так як принципи систем розподілених обчислень в значній мірі співпадають із принципами з технології кластеризації, то існує думка, що grid-системи також є одним із видів кластерів. Насправді ж, існує суттєва різниця між двома технологіями. По-перше, варто сказати, що перед розгортанням системи розподілених обчислень не стоїть завдання зменшити час простою, що є одним із ключових факторів при плануванні кластерної архітектури. З цього випливає те, що кожен із вузлів grid-системи не має великого рівня доступності, а отже гарантувати безперервну роботу над задачею неможливо. [11] Дані системи за своєю будовою схожі із обчислювальними кластерами, але, на відміну від них, вони не мають синхронізації, яка в цьому підході компенсується великою кількістю серверів.

## Архітектура кластерів

Кластери зазвичай складаються із наступних елементів:

- Вузли

- Сервіси

- Завдання, репліки та стеки

Вузли — це основна одиниця в архітектурі кластеру, на яких розгортаються контейнери, сервіси, а також інші служби, які підтримують роботу кластера. Вузли поділяються на два наступних види:

а) Вузол керування (або ж мастер-вузол (від англ. master - керівник)) – входить в площину керування кластером, задача мастера полягає у видачі вказівок на запуск контейнерів та сервісів і моніторинг їх стану, зв’язок всіх вузлів та інших компонентів кластера між собою. Крім того, вузол керування є єдиною точкою входу, через який трафік потрапляє до кластера. Створення реплік, перевірка роботи інфраструктури, робота із сховищем даних та обробка запитів користувача також є одними із задач вузла керування. [11]

б) Робочий вузол – це один із хостів в кластері, який отримує вказівки на виконання задач від мастер-вузла з площини керування. Такий вузол відповідає за запуск, підтримку працездатності та адміністрування сервісів, служб і програм, які запущені на ньому.

Крім того, вузол керування може бути одночасно і робочим вузлом, але поєднання функцій площини керування і робочого вузла вплине на ефективність та продуктивність роботи. [11]

Сервіс — це певний набір інструкцій, які користувач визначає декларативно у спеціально створеному для цього файлі з розширенням .yaml, які виконуються по відношенню до обраного образу контейнера. Існує велика кількість службових сервісів, які в тій чи іншій мірі забезпечують підтримку успішної роботи кластера. Зазвичай такі сервіси запускаються в залежності від задачі або на робочих вузлах або на вузлах керування, або на всіх вузлах одночасно. Сервіси користувачів запускаються на робочих вузлах і мають різні функції. Так, наприклад, через сервіси можна налаштовувати доступ до зовнішньої мережі контейнерам розгорнутого додатку, визначати та «відкривати» певні порти для програми, а також в деяких випадках налаштовувати правила фільтрації вхідного трафіку або поведінки в певних ситуаціях, тощо [12] На рисунку 1.1 зображена діаграма з прикладом використання сервісу. В цій ситуації трафік спочатку проходить через обробку зовнішнім балансувальником навантаження, який перенаправляє цей запит до сервісу, (В даному випадку використовується сервіс типу LoadBalancer, який працює у зв’язці із цим балансувальником) задача якого – вибрати найбільш підходящий контейнер та відправити запит саме туди.

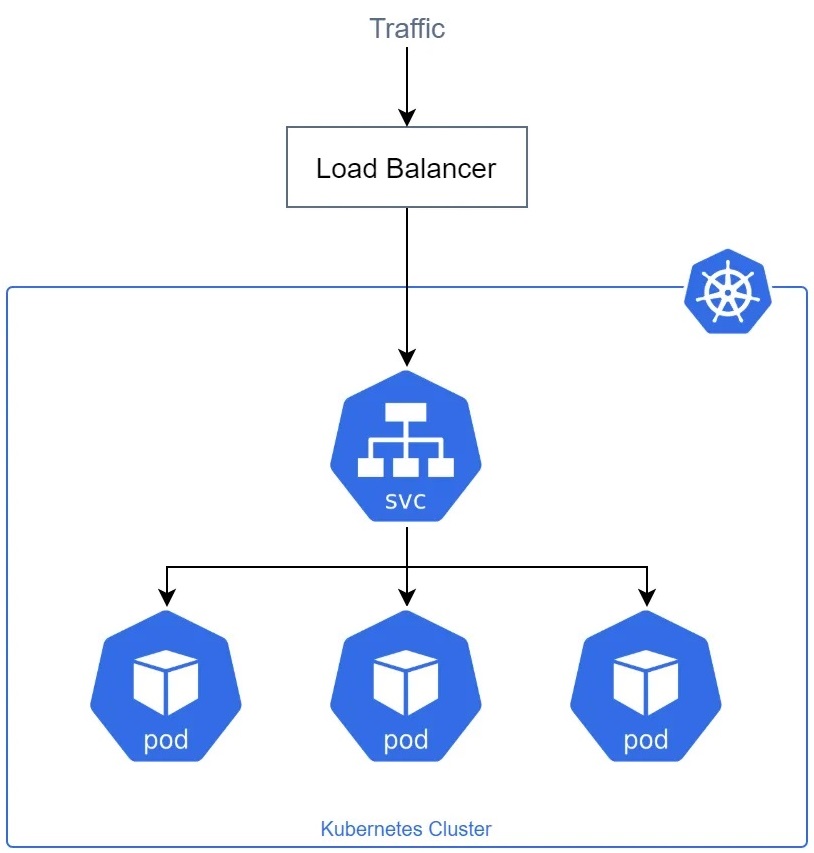


Рисунок 1.1 приклад використання сервісу

Стек - це набір об’єднаних сервісів, які працюють у зв’язці і представляють один суцільний проект [13] Наприклад, для підтримки роботи сервісу веб-сайту необхідний приблизно такий стек: веб-додаток для обробки запитів і маршрутизації, мікросервіс, який відповідає за базу даних, а також додаткові служби для черги завдань. Варто зазначити, що стек, як і сервіс, також визначається декларативно, але при цьому всі сервіси групи описані у одному .yaml файлі. Крім того, додатково необхідно вказувати принцип взаємодії один із одним.

Набір реплік (з англ. ReplicaSet) – це інструмент, який дозволяє запустити автоматично встановлену у .yaml файлі кількість реплік однієї програми. Спеціально відведений процес – контролер реплік – вузла керування буде контролювати та підтримувати цю кількість. У випадках, коли у роботі кількість реплік менше, ніж задана у наборі, цей контролер буде створювати нові розгортання програми до моменту, коли кількість працюючих буде дорівнювати відведеній кількості, яка описана у файлі. [13]

Завдання – це набір інструкцій, яку повинен виконати вузол або ж певний службовий сервіс в кластері.

Деякі кластери можуть включати в собі також Балансувальник навантаження (від англ. Load Balancer), який відповідає за перерозподіл навантаження, трафіку або ж запитів, які спрямовані до кластера, таким чином, щоб продуктивність та використання ресурсів кожного із вузлів була би максимально однаковою. [12] В деяких кластерах він може бути відсутнім, але ця проблема вирішується інтеграцією зовнішнього балансувальника навантаження. Так, сучасні хмарні провайдери кластерних сервісів, такі як EKS від Amazon Web Services, або ж AKS від Microsoft Azure включають за замовчуванням свої сервіси для перерозподілення навантаження, і користувачам не потрібно витрачати час та ресурси на додавання стороннього. Якщо ж робота йде з кластером, який використовує наявну локальну фізичну інфраструктуру, то на етапі реалізації кластера необхідно самостійно інтегрувати один із наявних балансувальників.

Кластер потребує спеціального програмного забезпечення задля запуску та моніторингу програм і сервісів (які можуть працювати тільки у контейнеризованому вигляді), а також керування всіма процесами всередині. Дане ПЗ називається системою оркестрування. Неможливо описати поняття «оркестрування» без розкриття теми «контейнеризації», так як одне прямо витікає із іншого.

## Поняття контейнеризації

Контейнеризація — це набір технологій, завдяки яким з’явилась можливість пакувати та ізолювати програмний код разом із усім середовищем виконання (бібліотеками та іншими файлами, необхідними для запуску додатку) у один єдиний легкий за об’ємом виконуваний файл, який називається контейнером. [12] Контейнери працюють в будь-якій інфраструктурі та ефективніше витрачають ресурси, ніж у випадку, коли для перевірки роботи програми запускались би віртуальні машини (VM) із необхідним середовищем. Сьогодні майже неможливо уявити роботу із створення хмарних додатків без використання контейнерів, які фактично стали обчислювальними одиницями.

Існує широкий спектр значних переваг для користувачів контейнерної архітектури. Серед них можна виділити:

Портативність: За допомогою традиційних методів код розробляється в спеціальному обчислювальному середовищі, яке при перенесенні в нове місце часто призводить до помилок. [12] До таких ситуацій можна віднести, наприклад, перенесення розробником коду програми із персонального настільного комп’ютера на віртуальну машину або з однієї операційної системи на іншу. Контейнерізація усуває цю проблему, так як кожна запакована програма із своїми залежностями у контейнері є абстрагованою, і не залежить від основної операційної системи. Отже, таку програму можна легко переміщувати між середовищами та запускати на різних хостах із різними версіями операційної системи, зберігаючи при цьому повну функціональність додатку. [13]

Ефективність роботи: такі контейнеризовані програми не займають суттєвого обсягу пам’яті і не вимагають серйозних витрат інших ресурсів на підтримку їх роботи. З цього випливає те, що такі контейнери не вимагають значного часу на запуск, а невеликий об’єм є причиною здатності запуску більшої кількості контейнерів на одному хості із збільшеною загальною обчислювальною здатністю. Таким чином, це підвищує ефективність використання серверів, зменшуючи витрати на розгортання додаткової інфраструктури, а також одночасно із цим пришвидшує розробку та тестування програм, які знаходяться на стадії доопрацювання.

Усунення несправностей: так як кожна контейнерна програма ізольована та працює незалежно від інших, то помилки під час роботи одного контейнера не впливають на ефективність роботи інших контейнерів. [12] Також варто сказати те, що механізм роботи контейнерів підтримує додаткову інтеграцію будь-яких наявних інструментів ізоляції безпеки ОС. [13] Наприклад, можна використовувати таку технологію, як контроль доступу SELinux, задля ізоляції несправностей, які можуть виникнути під час виконання програми контенейнером.

Безпека: у разі зараження будь-яким вірусом одного контейнера, завдяки ізоляції, вплив шкідливого ПЗ на інші розгорнуті контейнери або систему неможливий. Крім того, при правильному налаштуванні правил безпеки можна отримати автоматичне блокування небажаних команд та запитів до контейнерів, або ж обмежити зв’язок із певною кількістю ресурсів. [12]

В архітектурі контейнеризації самі контейнери займають тільки останній рівень. Сама ж архітектура включає наступні рівні:

* Інфраструктурний
* Рівень Операційної системи
* Контейнерий двигун
* Рівень додатків та їх залежностей

До інфрастрктурного (або ж апаратного) рівня відноситься фізичний персональний комп’ютер або сервер без встановленої на ньому ОС, на якому передбачається запуск контейнеризованого додатку. [13]

Другий рівень архітектури контейнеризації є операційна система. Зазвичай, для роботи контейнерів на локальних системах використовується Linux, але існує підтримка великої кількості інших ОС. У хмарних середовищах розробники можуть використовувати своє спеціальне ПЗ замість операційної системи. Так, на приклад, на платформі Amazon Web Services для запуска та стабільної роботи контейнерних додатків використовуються спеціально створений для таких задач сервіс AWS EC2. [14]

Третій рівень – це Контейнерний двигун, або середовище виконання роботи контейнеру. Під цим поняттям мається на увазі розроблене ПЗ, яке вирішує завдання створення контейнерів на основі обраних користувачем образів контейнерів та підтримка їх роботи. [15] Такий сервіс займає середню ланку у зв’язку між операційною системою і самим контейнером. Таким чином, контейнерний двигун може виділяти необхідні ресурси для стабільної роботи контейнера, а також керувати всіма процесами всередині.

Останній і найвищий рівень в архітектурі – це Рівень додатків та їх залежностей. Під цим поняттям мається на увазі сам контейнер.

До одного із недоліків використання контейнеризації можна віднести той факт що, сам по собі контейнер є ефемерним. Це означає те, що дані, записані у файлову систему контейнера, втрачаються, коли він вимикається або ж перестає працювати через певні причини. [15] Дана проблема вирішується інтегруванням у систему об’єктного сховища, такого як MinIO.

Технологія контейнеризації насправді не нова: вона виникла ще наприкінці ХХ століття, але вона не мала такого широкого використання до 2012-2013 років. Google та Docker були одними із перших компаній-учасників так званої «нової хвилі» досліджень контейнерної технології Linux, яке проводилось в 2008-2013 роках. [12] За результатами цих досліджень була розроблена велика кількість програмного забезпечення для ефективного керування контейнерами. Одним із таких ПЗ було Docker Engine, яке дуже швидко стало стандартом для галузі контейнерів. Крім того, це програмне забезпечення поширується вільно, що вплинуло на швидке формування спільноти. Це все призвело до значного прискорення розвитку всієї концепції, що, в свою чергу, прямо вплинуло на популярність такої архітектури серед користувачів.

Сьогодні компанії-розробники широко використовують контейнеризацію для створення нових додатків і тестування та доопрацювання існуючих сервісів. Так, в опитуванні від IBM, 61% респондентів повідомили, що використовували контейнери в половині або більше нових програм, які вони створили протягом попередніх двох років; 64% користувачів очікують, що 50% або більше їхніх існуючих додатків будуть поміщені в контейнери протягом наступних двох років. [16]

У вересні 2021 року компанія Dell проводила опитування користувачів, яке показало, що 88% респондентів використовують контейнеризовані додатки та сервіси, або ж створюють свої програми, які запаковані у контейнери. [16]

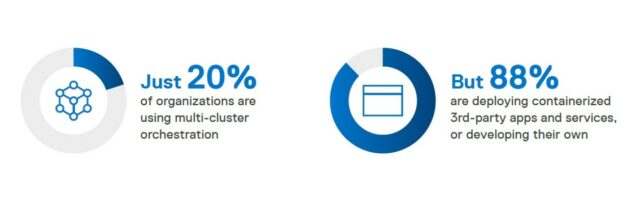


Рисунок 1.2 – Результати опитування, яке було проведено компанією Dell

Серед задач використання контейнеризації є не тільки запуск програм. Так, наприклад, контейнерні двигуни можуть керувати цілою групою контейнерів, яка розгорнута на одному хості, зберігаючи при цьому принципи ізольованості кожного із них.

Крім того, враховуючи переваги портативності, розробники можуть використовувати контейнери при перенесенні даних із локальної інфраструктури до хмарного середовища. Це дозволяє змогу не змінювати великий обсяг програмного коду під конкретне середовище.

## Технологія оркестрування для ефективного керування середовищем

Із зростанням популярності, компанії-виробники все частіше починають переходити саме на контейнеризацію своїх сервісів. З часом кількість розгорнутих контейнерів досягає такого рівня, при якому постає проблема доступності цих контейнерів, а адміністрування великих об’ємів контейнеризованих додатків стає неможливим без додаткового програмного забезпечення. Проблема доступності, як було описано раніше, вирішується залученням додаткової інфраструктури та поєднанням групи серверів у один кластер. Але в такому випадку, до проблеми неможливості адміністрування додається ще й проблема відсутності синхронізації між контейнерами, які запущені на різних хостах-машинах. В такому випадку одним із можливих рішень є використання технологій оркестрування для налаштування роботи контейнеризованого середовища у вигляді кластера.

Оркестрування (або ж оркестратор) контейнерів – це технологія, яка автоматизує розгортання, полегшує процес керування та масштабування контейнерів у системі. [17] Використовуючи оркестрування можна зменшити витрати часу та інших ресурсів на ручне поєднання серверів у один кластер та запуск і налаштування одночасно великої кількості контейнерів в ньому. Зазвичай, програмне забезпечення для оркестрування контейнерів працює у зв’язці із описаним вище контейнерним двигуном. В той час, як контейнерний двигун відповідає за запуск та підтримку робочого стану контейнеру, ПЗ оркестрування відповідає за зв’язок між контейнерами не тільки на одному хості, а і у всьому кластері таких машин. Крім того, в обширний спектр завдань оркестратора входять такі функції, як автоматизація великої кількості процесів, пов’язаних із розгортанням і масштабуванням контейнерів, створенням мережі між контейнерами, розподіленням мережевого трафіку між ними, а також підтримка всіх необхідних для роботи середовища (тобто кластера) ресурсів, де запущені контейнери із сервісами та користувацькими додатками. [18] Можна сказати, що оркестрування управляє життєвим циклом контейнерів і програм, які розгорнуті у системі. Деякі оркестратори мають можливість та інструменти для інтеграції стороннього ПЗ для зберігання інформації на вибір клієнта. До таких систем можна віднести або налаштоване локальне сховище даних, або ж сторонні публічні хмарні рішення.

Google, як зазначається у звітах компанії, створює понад 2 мільярди розгортань контейнерів на тиждень [17]. Така кількість операцій стала доступно тільки за допомогою оркестраторів. Google, беручи до увагу попередній досвід у експлуатації контейнерних розгортань, розробила внутрішню платформу Borg, яка потім еволюціонувала у досить популярне ПЗ для оркестрування Kubernetes. Крім того, ця компанія змогла вдало інтегрувати технології оркестрування до своїх хмарних сервісів, що призвело до розвитку контейнерної технології і у хмарних середовищах.

Згідно із організацією Datadog, яка займається опитуваннями, станом на 2021 рік вже більше 90% контейнерів були оркестрованими, і цей відсоток з кожним роком все збільшується і збільшується. [16]

Підсумовуючи все вище сказане, можна зробити висновок, що технологія оркестрування контейнерів є подальшим розвитком контейнеризації. З кожним роком все більше і більше компаній-розробників сервісів переходять та починають використовувати підходи оркестрування під час розгортання додатків.

## Види ПЗ оркестрування контейнерів

Сьогодні існує великий вибір інструментів оркестрування контейнерів для розробників, і кількість таких ПЗ невпинно зростає. Серед найвідоміших можна назвати наступні: Kubernetes, Docker Swarm, Mirantis, OpenShift, Rancher та інші. При цьому, найголовнішими гравцями на цьому ринку є перші два. Враховуючи широку популярність Kubernetes і Docker Swarm, було прийняте рішення виконати їх порівняння з метою визначення технології, яка буде використовуватися у цій кваліфікаційній роботі.

### ПЗ для оркестрації Kubernetes

Kubernetes – це система оркестрування контейнерів, яка була розроблена компанією Google. Назва Kubernetes походить від грецького слова (грец. κυβερνήτης kubernḗtēs), що використовується у значенні «рульовий», «навігатор», або «пілот». [9] Також це ПЗ має скорочену назву – K8s, що складається із першої, восьми пропущених та останньої літери повної назви. Kubernetes спочатку була закритою технологією, яка розроблена інженерами компанії Google, але у 2014 році відбувся випуск вихідного коду у вільний доступ під ліценцзією Apache 2.0. В результаті вільного поширення почалося активне формування спільноти, яка забезпечила швидкий розвиток. Зараз Kubernetes – це продукт який має стабільні оновлення, а також широку спільноту і базу користувачів. Крім того, дана система може використовуватись в проектах будь-якого масштабу. Kubernetes поєднує понад 15 років досвіду Google у виконанні робочих навантажень у великих масштабах із пропозиціями спільноти. K8s також інтегрований у велику кількість хмарних сервісів, наприклад таких, як AWS від Amazon, які активно використовують цю технологію.

Основна мета системи полягає у спрощенні розгортання, масштабування та управління контейнеризованими застосунками. Це ПЗ поєднує як базові механізми для розгортання додатків, так і дозволяє полегшити декларативне налаштування і автоматизацію процесів кластера. [9] Таким чином, завдяки Kubernetes, розробники мають можливість ефективного керування групою контейнерів будь-якого масштабу при цьому забезпечуючи високу швидкість роботи застосунків. K8s також може відповідати (якщо таке завдання постановлене) і за перезапуск контейнерів, які виходять із ладу, а також моніторинг використаних ресурсів кожної програми чи службового сервісу в ньому. Таким чином, якщо один сервіс починає використовувати більше ресурсів, ніж було задано шляхом встановлення лімітів, то Kubernetes почне обмежувати вплив такого додатку на інші працюючі програми в системі. Крім того, ця система оркестрування має такі функції, як створення реплік однієї і тієї ж програми, а також балансування навантаження між цими репліками. Використовуючи інструменти, які наявні у Kubernetes, можна відкрити доступ для контейнеризованого сервісу поза кластером, а також налаштувати передачу конфігурації чи іншої інформації, яка описана у secret-файлах, до будь-якої програмі, запущеній в системі. Функція запуску одноразових завдань або завдань за розкладом також входить в спектр можливостей K8s.

Незважаючи на той факт, що Kubernetes найчастіше використовується для оркестрування контейнерних програм, він також може керувати іншими типами робочого навантаження, зокрема віртуальними машинами та групою серверів. [18] Таким чином, K8s надає і інструменти для поєднання групи робочих машин у один єдиний кластер, а також адміністрування як його, так і всіх процесів в ньому.

#### Дослідження статистики та прикладів використання Kubernetes

За результатами дослідження, яке проводила організація CNCF, стало зрозуміло, що Kubernetes зараз є другим за величиною проектом з відкритим кодом у світі після Linux [16]. Згідно із проведеними опитуваннями, близько 58% респондентів використовують K8s під час розробки власних сервісів, тоді як 42% опитуваних планують використовувати у майбутньому. [16] Тією ж організацією було проведене ще одне дослідження, задача якого полягала у оцінюванні популярності різних середовищ, де запускається K8s. За даними, 51% респондентів використовують Kubernetes у AWS, в той час як запускають на локальних серверах у 37% випадків. Також для Kubernetes використовується Google Cloud Platform, Microsoft Azure, OpenStack, а також VMware. [16] На рисунку 1.3 подано результати цього опитування.

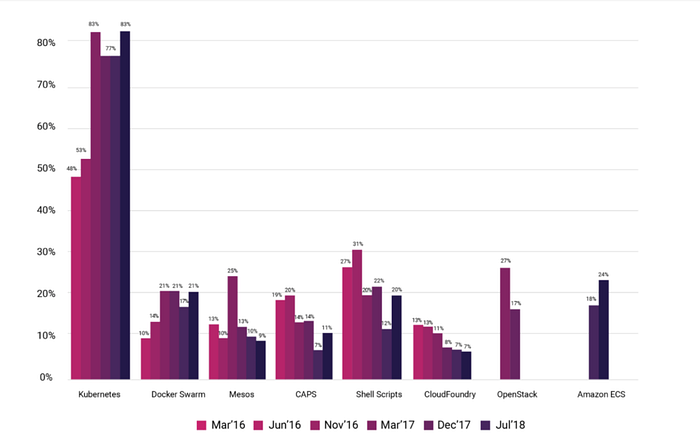


Рисунок 1.3 – Результати опитування

Найбільшим сервісом, що зараз працює використовуючи Kubernetes, є сервіс прослуховування музики, який розроблений компанією Spotify. Даний стрімінговий сервіс приймає загалом близько 10 мільйонів запитів на секунду. [8] Враховуючи такий масштаб кількості запитів, без автоматичного масштабування, що надає K8s було би неможливо забезпечити доступність сервісу на належному для користувачів рівні. Так, згідно із звітами, раніше доводилося відводити значний період часу на створення нової репліки і запуск на робочому хості. [8] Використовуючи Kubernetes адміністратори можуть зробити вищеописані завдання не витрачаючи значних ресурсів і часу.

#### Переваги Кubernetes

Проаналізувавши технічну документацію K8s, можна виділити наступний спектр функцій та фактів, які можна вважати перевагами програмного забезпечення:

* Вихідний код K8s розповсюджується вільно по ліцензії Apache 2.0.
* Постійні оновлення і велику підтримку як від компанії-розробника Google, так і від широкої спільноти користувачів.
* Можливість запуску як на Linux різних дистрибутивів, так і на Windows і Mac OS.
* Можливість підтримувати роботу системи під час критичних ситуацій.
* Великий вибір вбудованого інструментарію для керування всіма процесами.
* Автоматизація великої кількості процесів.
* Здатність до самовідновлення після помилок у роботі.
* Підтримка автоматичного масштабування.
* Вбудовані інструменти для моніторингу.
* Підтримка можливості інтегрувати необхідні сторонні технології.
* Підтримка від таких ключових постачальників хмарних технологій, як GCP, AWS або Azure.
* Можливість як переходу існуючого проекту на хмару, так і повернення до локального розгортання.

#### Недоліки Kubernetes

Незважаючи на широкий набір функцій, Kubernetes також має ряд недоліків:

* Складний процес встановлення компонентів.
* Складний шлях навчання всіх компонентів.
* Відсутність вбудованого CLI.
* Перенесення з інших систем оркестрування до K8s може бути складним та важко контрольованим.

Крім того, існує максимум кількості вузлів та контейнерів у одному кластері K8s. Запустити можна не більше ніж 5000 вузлів та 300 тис контейнерів, але проблеми у експлуатації можуть початися вже за кількості 500 вузлів [9]. Дана проблема полягає в тому, що адміністрування великих та об’ємних кластерів призводить до високого навантаження на вузли керування K8s. З цього випливає те, що для підтримки всієї системи у робочому стані та задля ефективного використання ресурсів, необхідне залучення комплексного та ретельного налаштування до всіх компонентів кластера, а також велика увага до розподілення навантаження.

Також, варто сказати про факти, які одночасно відносяться як до переваг, так і недоліків. Як раніше зазначалося, K8s має широку підтримку як від розробників, так і від спільноти, що призвело до створення великого спектру спектру функцій та інструментів для керування сценаріями розгортання. Але, на противагу цьому, комплексність та великий об’єм K8s в певних випадках у парі з неправильним налаштуванням може призвести до втрати продуктивності та інших непередбачуваних наслідків.

### Система оркестрування Docker Swarm

Docker Swarm – це програмне забезпечення для оркестрування контейнерів, яке дозволяє користувачеві керувати великою кількістю контейнерів, які розгорнуті на певній кількості машин. Дане ПЗ перетворює кілька екземплярів Docker в один віртуальний хост, а також може поєднати із іншими віртуальними хостами в одну систему.

Розробником даної системи оркестрування є компанія Docker, яка є однією із найперших, що запропонувала реалізацію роботи контейнерів у групі та управління ними. [19] Docker Swarm має відкритий вихідний код та доступна для вільного розповсюдження. Swarm використовує широкий спектр технологій для керування всіма компонентами розгорнутого кластера. Присутній об’ємний АРІ, який дозволяє без проблем розгортати складні за архітектурою мікросервісні програми та керувати ними. [20]

Із статистики використання систем оркестрування контейнерів, яка була приведена у підрозділі 1.9.1.1, випливає той факт, що Docker Swarm стабільно займає друге місце за популярністю використання після Kubernetes. Дане ПЗ, як і K8s, також має велику за обсягом спільноту, яка приймає аналогічну активну участь у розвитку Swarm’у.

Провівши аналіз функцій та можливостей цієї системи оркестрування можна виділити наступні переваги:

* Швидке встановлення без необхідності встановлювати додаткові компоненти.
* Відсутність комплексної та складної архітектури.
* Легкий шлях навчання, що призводить до швидкого освоєння із ПЗ.
* Доступне вбудоване автоматичне балансування навантаження на контейнери, а також трафіку між ними.
* Має вбудоване Docker CLI.
* Сумісність із іншими інструментами та ПЗ від компанії Docker.
* Відсутність вимог по серйозній зміні конфігурації, якщо система раніше була запущена в Docker (середовище для керування контейнерами на одиночній машині).

Незважаючи на вищеприведені переваги, у Docker Swarm є кілька недоліків, про які слід знати при виборі середовища оркестрування. Такими є:

* Прив’язаність до Docker API, що обмежує функціональність кластера у порівнянні із Kubernetes
* Безпека автоматизації внутрішніх процесів реалізована на низькому рівні. [20]

## Вибір технології оркестрування для цільового кластера

Враховуючи важливість встановлювання системи оркестрування для повноцінної та ефективної роботи кластерів, була проаналізовано сильні і слабкі сторони двох оркестраторів, які підходять під потреби цільового кластера. За результатами такого дослідження, можна прийти до висновку, що і K8s, і Swarm є сучасними та доступними системами управління як групою серверів, так і масштабних розгортань додатків-сервісів з можливістю керування всіма супутніми процесами. Kubernetes і Docker Swarm є найпопулярнішими інструментами у своїй сфері: вони стабільно займають перші місця у опитуваннях серед користувачів контейнерних окрестраторів, ці інструменти можна використовувати для розгортання та керування контейнерами, як використовуючи власну локальну інфраструктуру, так і використовуючи хмарних провайдерів. Крім того, обидва проекти мають постійні оновлення та підтримку від розробників.

Порівняння проводилося за наступними критеріями:

* Архітектура та функціональність
* Інсталяція та налаштування
* Шлях навчання
* Додаткові функції для розгортання програм
* Горизонтальне автоматичне масштабування
* Моніторинг
* Безпека
* Балансування навантаження
* CLI
* Інструменти збереження стану програм
* Робота із контейнерними двигунами

Результати порівняння наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння Kubernetes та Docker Swarm

|  |  |  |
| --- | --- | --- |
| Критерій порівняння | Kubernetes | Docker Swarm |
| Архітектура та функціональність | Комплексна архітектура, з великою кількістю компонентів. Функціональність не обмежена, наявний широкий спектр інструментів для автоматизації та масштабування процесів, здатність до самовідновлення, масштабування, тощо | Проста архітектура, без великої кількості компонентів. Функціональність ПЗ обмежена функціональністю Docker API |
| Інсталяція та налаштування | Довга інсталяція та комплексне налаштування всіх компонентів | Швидке встановлення та налаштування |
| Шлях навчання | Складний | Легкий |
| Додаткові функції для розгортання програм | Підтримка інтеграції широкого спектру сервісів, які створені як розробниками, так і спільнотою | Підтримка інтеграції невеликого переліку сторонніх сервісів |
| Горизонтальне автоматичне масштабування | Пристунє | Відсутнє |
| Моніторинг | Вбудовані системи моніторингу і сповіщення про помилки у роботі | Відсутні вбудовані інструменти, але наявна підтримка сторонніх існуючих сервісів |
| Безпека | Наявні вбудовані сервіси для забезпечення безпеки передачі даних між процесами | Безпека передачі даних між внутрішніми процесами на низькому рівні |
| Балансування навантаження | Відсутній вбудований балансувальний навантаження, але наявна підтримка сторонніх інструментів | Вбудоване |
| CLI | Відсутній вбудований CLI, необхідно встановлювати самостійно Kubectl | Вбудоване Docker CLI |
| Інструменти збереження стану  програм | Відсутність вбудованих, підтримка інтеграції сторонніх систем збереження даних | Відсутність вбудованих, підтримка інтеграції сторонніх систем збереження даних |
| Робота із контейнерними двигунами | Підтримка запуску великого спектру контейнерних двигунів | Підтримка запуску тільки двигуна Docker Engine |

Таким чином, можна сказати, що хоч і Kubernetes і Docker Swarm забезпечують основу для керування контейнеризованим середовищем, але вони обидва мають різну мету та базу використання. Docker Swarm надає менший спектр інструментів, але при цьому має досить легкий шлях навчання та невеликий об’єм. З іншого боку, Kubernetes – це комплексне і складне рішення, але надає значно ширший обсяг функцій для користувачів, таких як доступність самовідновлення на автоматичного масштабування. Ключовим фактором вибору ПЗ оркестрування для реалізації кластера є безпека, захист даних та набір інструментів. Так як Kubernetes має значні переваги над Docker Swarm у цих критеріях, то було прийняте рішення в цій кваліфікаційній роботі використовувати саме його.

Після того, як зроблено вибір щодо способу оркестрування, необхідно сформувати та поставити задачу на реалізацію кластера.

## Постановка задачі та планування цільового кластера

Враховуючи важливість поставленого завдання, при плануванні інфраструктури, яка повинна розгортати та підтримувати сервіси телемедичних досліджень, необхідно визначитися із таким стеком технологій, який би забезпечував як надійність зберігання даних, так і доступність цих даних на високому рівні для пацієнтів та медичного персоналу.

Проаналізувавши існуючі підходи до кластеризації, було прийняте рішення взяти за мету реалізацію та налаштування HA кластера. Для імплементації такої архітектури було вирішено запроваджувати підхід з розгортання високодоступного кластера з гарячим резервом, який був детально описаний у розділі 1.3.3. За результатами порівняння двох сучасних ПЗ для оркестрування контейнерів у розділі 1.9 було обрано Kubernetes.

Під час планування кластера перед розробниками постає проблема зберігання інформації, яка полягає у фіксації результатів. Так, кожне дослідження або консультація потребує зберігання та уніфікації форматів даних. Враховуючи велику кількість пацієнтів та учасників досліджень, необхідно інтегрувати таке сховище даних, яке дозволяло б накопичувати відповідні обсяги даних та обробляти їх у досить стислі терміни.

Вкрай важливим питанням є інтеграція Container Networking Interface (CNI), спеціалізованого ПЗ, яке дозволить налаштувати зв’язок між вузлами кластера і зовнішньою мережею. Без цього компоненту медичний персонал не отримає доступу до всього сервісу і не зможе заносити результати досліджень у базу даних: передача даних буде реалізована тільки на рівні Мережевого проксі, який відповідає за передачу даних всередині кластера.

Крім того, за відсутністю вбудованого у K8s балансувальника навантаження, необхідно обрати найбільш сумісний із CNI та налаштувати його на розподілення трафіка між вузлами. Найбільш популярним балансувальником навантаження є MetalLB, який при своєму широкому функціоналі розповсюджується вільно та має довготривалу підтримку від розробників. MetalLB спеціалізується на наданні LoadBalancer-сервісів для Kubernetes кластеру. Він може автоматично назначати зовнішні IP-адреси сервісам Kubernetes, які вимагають зовнішнього доступу. [23] Крім того, цей балансувальник не потребує великої кількості ресурсів для його роботи, що є одною з ключових переваг над існуючими аналогами, такими як Nginx Controller, які до того ж не мають такої комплексної підтримки від розробників. Таким чином, було прийняте рішення в цій кваліфікаційній роботі використовувати саме MetalLB для підтримки розгортання сервісів телемедичних досліджень.

Так як метою цієї кваліфікаційної роботи є перш за все напрацювання практичного досвіду із розгортання, налаштування та адміністрування кластера під певну задачу, то використання хмарних провайдерів для ознайомлення та вивчення кластерів не є ефективним через індивідуальні (і частіше за все не сумісні) екосистеми сервісів у кожного провайдера, а також неминучі експлуатаційні витрати у майбутньому.

Таким чином, було прийняте рішення створити та налаштувати групу віртуальних машин, яка б імітувала існуюче обладнання, і вже потім на цій основі розгортати кластер.

Так як операційна система Linux дистрибутива Ubuntu варіації Server є однією із найбільш ефективних та сучасних рішень з дотриманням усіх методів безпеки для використання на серверах, то на розгорнутій інфраструктурі необхідно встановити саме цю ОС.

Також необхідно розгортнути сервіси моніторингу стану кластера і сповіщень про наявні проблеми в ньому. Для економного використання ресурсів необхідно реалізувати принцип доступу до зібраних метрик процесів кластера високої доступності тільки через встановлений зазделегідь зовнішній балансувальник навантаження.

Крім того, вкрай важливо оновлювати версію розгорнутих сервісів та програм через запуск пакетного менеджера Helm, який реалізовує підхід CI/CD, який буде відповідати за безперервне автоматичне розгортання оновлених додатків

Таким чином, підбиваючи підсумки, можна сформувати наступні вимоги до цільового проекту:

* Створення та налаштування віртуальних машин з встановленою на них ОС Ubuntu Server.
* Відкриття необхідних для роботи портів.
* Розгортання HA кластера виду активний-активний із залученням ПЗ Kubernetes, використовуючи наявні VM.
* Інтеграція CNI в роботу кластера.
* Під’єднання об’єктного сховища даних.
* Розгортання зовнішнього Load Balancer MetalLB для сервісів
* Налаштування системи моніторингу та сповіщень
* Запуск пакетного менеджера Helm
* Реалізація підходу CI/CD

В результаті можна навести діаграму роботи майбутньої системи.

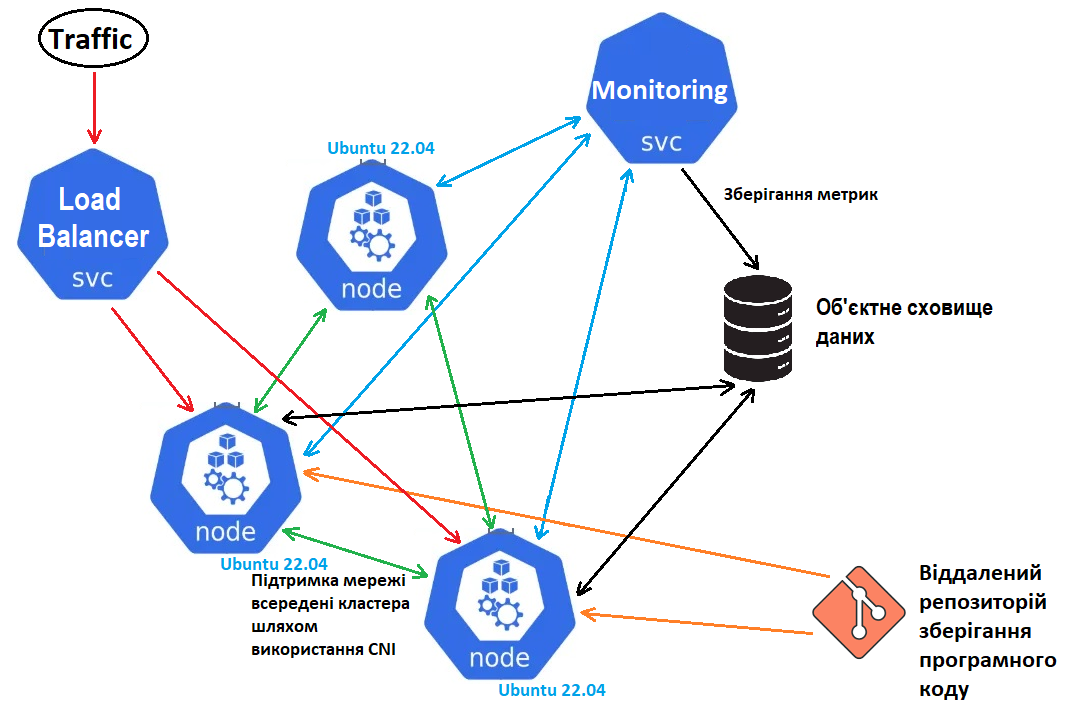
****

Рисунок 1.4 – Діаграма роботи кластерної системи

## Висновки до першого розділу

В першій частині було проведено аналіз технології побудови розподіленої архітектури, за результатами якого було виокремлено види кластерних архітектур. Крім того, проаналізувавши різницю між принципами побудови кластерної архітектури та grid-систем, можна прийти до висновку, що ключовою відмінністю між двома технологіями є забезпечення висока доступність. Так, при схожій побудові, на відміну від всіх видів кластерних системи, grid-системи не забезпечують високу доступність. [12] Таким чином, спростовується твердження про винесення grid-систем до підвидів кластерів.

За результатами аналізу технологій контейнеризації та оркестрування встановлено, що ці технології, маючи велику популярність, пропонують широкий спектр інструментів для виконання поставлених задач, а також можуть використовуватися для підтримки роботи розподіленої інфраструктури.

Після опрацювання теоретичного матеріалу було сформовано та поставлене завдання на реалізацію. На цьому етапі було вказані основні вимоги до розгортання такої інфраструктури, яка б дозволила запускати на ній сервіси для виконання телемедичних досліджень.

# ПРОЕКТУВАННЯ РОЗПОДІЛЕНОЇ ІНФРАСТРУКТУРИ СЕРВІСІВ ДЛЯ ЗАДАЧ ТЕЛЕМЕДИЧНИХ МОНІТОРИНГОВИХ ДОСЛІДЖЕНЬ

## Огляд архітектури Kubernetes

Як було зазначено раніше, Kubernetes має комплексну архітектуру, яка складається із окремих компонентів, які необхідно встановити. До основних компонентів можна віднести Площину керування кластером, Pod`и, Etcd сховище, Kubernetes API Server, Планувальник, Менеджер контролерів кластера, Kubelet, Мережевий проксі, а також Середовище роботи контейнерів. [9] Крім того, хмарні рішення повинні включати також Хмарний менеджер контролерів, який віповідає за зв’язок між API хмарного провайдера. Зв’язок між всіма компонентами в кластері K8s наведено на рисунку 2.1.

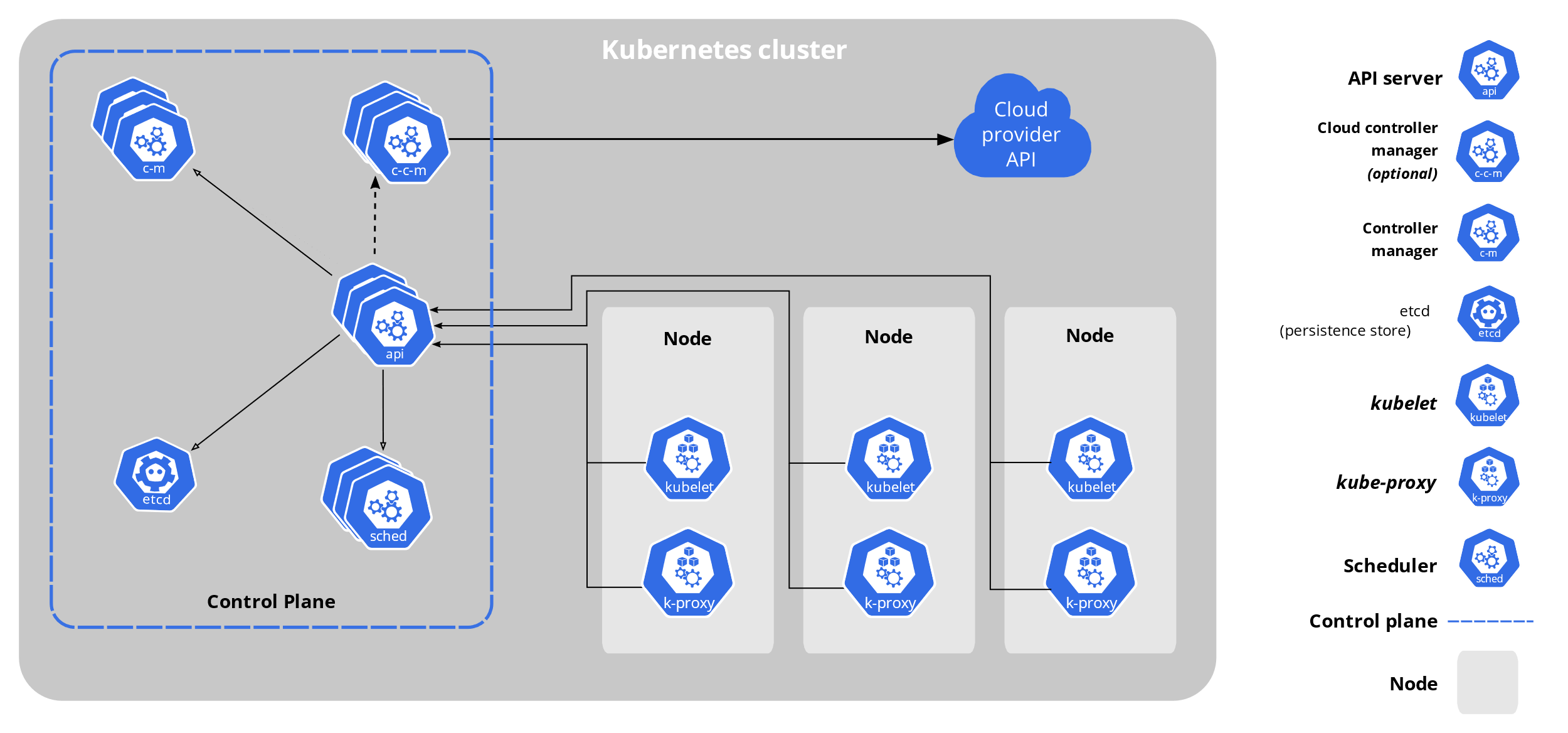


Рисунок 2.1 – Архітектура взаємодії компонентів кластера Kubernetes

## Площина керування та Pod`и

Площина керування кластера Kubernetes складається із одного, або групи вузлів, які відповідають за керування усіма процесами всередині кластера.

Pod`и – це спеціальні ресурси кластера, які містять у собі один або декілька контейнерів, які представляють одну програму. Pod`и представляють собою так звану «надбудову» над контейнерами, і мають задачу зайняти проміжну ланку між Середовищем роботи контейнерів та вузлом. [10] Вся взаємодія ізольованих контейнерів із зовнішнім середовищем відбувається саме через ці Pod`и.

## Компоненти вузла керування

### Etcd сховище

Назва цього сховища походить від назви директорії /etc, де зберігаються всі конфігураційні файли в UNIX системах, а “d” – від слова distributed, тобто розподілене. Це сховище, як випливає із вищесказаного, створене для того, щоб зберігати критично важливі дані у форматі значення-ключ. [10] До критично важливої інформації для кластера можна віднести конфігураційні значення, метадані, дані про стан вузлів, Pod`ів, розгортань, тощо.

Дане сховище розгортається на вузлі керування, але доступ до etcd має кожен робочий вузол у повному обсязі.

### Kube-apiserver

API сервер - це компонент вузла керування кластера Kubernetes, який є frontend частиною кластера. [9] Головна задача цього сервера полягає у наданні інтерфейсів, використовуючи які інші компоненти кластера отримують можливість зв’язуватись один із одним. Крім того, АРІ сервер обробляє вхідні REST-запити (концепція взаємодії компонентів, яка використовує протокол HTTP) [21] і змінює вміст etcd сховища, що дозволяє користувачеві системи створювати задачі для виконання, а також розгортати та оновлювати контейнери, сервіси, та інші ресурси кластера. Одночасно із цим, АРІ сервер здійснює моніторинг компонентів шляхом відправки в певний проміжок часу запитів до одного із службових сервісів робочого вузла із командою надати поточний стан вузлів і контейнерів всередині цих вузлів. Також, АРІ сервер надає функцію масштабування вже запущених сервісів та контейнерів. [9]

### Планувальник

Даний агент розміщений на вузлі-мастері задля перевірки новостворених Pod`ів на прив’язку до вузла. Ще не прив’язані Pod`и такий Планувальник розподіляє до наймеш завантаженого по ресурсам вузла. У разі випадку, коли існує однакова кількість однаково навантажених вузла, агент починає порівнювати такі фактори, як наявність певних спеціальних обмежень і правил, встановлених користувачем, або ж час до закінчення роботи вузла, якщо такий вказано [11].

### Менеджер контролерів кластера

Компонент площини керування, задача якого запустити всі необхідні для роботи процеси-контролери. Незважаючи на те, що кожен такий контролер вважається окремим процесом, задля спрощення і так складної архітектури, розробниками було прийнято рішення помістити всіх їх у один двійковий файл та разом виконувати. [9] Таким чином, всі ці контролери виконуються в одному процесі вузла керування. Серед найважливіших контролерів можна виділити наступні:

* Контролер вузлів. Головне завдання цього процесу полягає у отриманні інформації від API серверу про стан кожного із вузлів, і у разі виявленні вузлів, які припинили роботу, цей контролер ставить мітку, і намагається вирішити проблему.
* Контролер задач – відслідковує наявність у системі об’єктів типу Job, які представляють одноразові завдання на виконання кластером, після чого виділяє ресурси для виконання поставленої задачі.
* Контролер EndpointSlice – підтримує роботу об’єктів типу EndpointSlice, які відповідають за забезпечення зв’язку між службами та модулями. [10]
* ServiceAccount – створює облікові записи однойменного типу для нових просторів імен.

## Компоненти робочого вузла кластера

### Kubelet

Одним із головних компонентів робочого вузла є Kubelet. Під час процесу під’єднання вузла до кластера цей агент розгортається та налаштовується на новій для кластера машині в першу чергу. Основна задача Kubelet`у полягає у здійсненні операцій створення та оновлення Pod`ів та контейнерів в них, у випадку, коли надходить така команда від вузла керування. [11] Крім того, цей агент відповідає за перевірку запущених контейнерів на справність, а також наявність прив’язки до Pod`ів. У разі відсутності такої прив’язки, або ж для новоствореного контейнера, Kubelet створює додатковий Pod, в який переміщає цей контенейнер. Під час своєї роботи агент використовує набір PodSpec`ів – спеціальних ресурсів, в яких описаний кожен контейнер, який необхідно розгорнути або перевірити. Варто зазначити, що цей агент повноцінно не керує контейнерами, спектр його дій в цьому плані обмежений тільки вищеописаними функціями.

### Мережевий проксі

Ще одним із найважливіших агентів, який запускається на робочому вузлі є Kube-proxy, або ж Мережевий проксі. Задача цієї служби – налагодити та підтримувати мережу між Pod`ами та сервісами всередині кластера. Мережевий проксі також підтримує мережеві правила, які встановлені Kubernetes`ом або ж задані користувачем, на всіх наявних вузлах.

### Середовище роботи контейнерів

Задачами середовища роботи контейнерів (з англ. container runtime) є запуск і підтримка робочого стану всіх контейнерів всередині кластера. Дане середовище не встановлюється ні під час ініціалізації кластера, ні під час додавання нових вузлів, тому користувачеві необхідно обов’язково самостійно на кожному майбутньому вузлі інсталювати це ПЗ. Під роботу у кластері Kubernetes створена велика кількість Container Runtime`ів. Найбільш популярними у цьому сегменті ПЗ є containerd, CRI-O, Docker Engine та Mirantis CR.

## Вибір необхідних компонентів для роботи кластера

### Вибір середовища роботи контейнерів

Для використання у цій кваліфікаційній роботі будуть розглядатися три основні середовища роботи контейнерів: Docker Engine, Containerd та CRI-O. Незважаючи на те, що всі вони призначені для управління контейнерами, вони мають свій підхід до роботи, а також інші відмінності у роботі.

Docker Engine – це середовище роботи контейнерів було розроблено компанією Docker. Це ПЗ надає повний набір функцій для управління контейнерами, включаючи можливість створення, запуску та керування контейнерами. Docker Engine є простий у управлінні, а також включає вбудований Docker Daemon, який виконує всі поставлені задачі. Серед переваг над іншими можна назвати наявність таких функцій, як підтримка мережі Docker, томів та глибоке налаштування та управління образами.

Containerd спочатку був розроблений компанією Docker як закритий внутрішній проект, але пізніше став поширюватися вільно із відкритим вихідним кодом під егідою Cloud Native Computing Foundation (CNCF). [22] Containerd є спрощеною версією Docker Engine, яка фокусується на базових функціях управління контейнерами. Containerd включає в собі всі основні можливості, які надає Docker Engine, а саме: розгортання, запуск, зупинка та видалення контейнерів, а також мінімальні функції із керування наявними образами. Containerd був розроблений з метою створення стабільної та простої у використанні платформи для управління контейнерами.

CRI-O. Цей Container runtime був розроблений спеціально для використання в кластері Kubernetes. CRI-O також є проектом, який розроблявся у CNCF і має спрощений набір функцій у порівнянні із Docker Engine. Основна задача робота CRI-O полягає в наданні ізольованого та високодоступного середовища для виконання контейнерів у Pod`ах кластера Kubernetes. [23] При зменшенії кількості функціоналу, CRI-O перейняв у Docker Engine тільки найважливіші інструменти, такі як технології ізоляції SELinux, AppArmor і Seccomp, які допомагають запобігати можливим атакам і забезпечити вищий рівень безпеки в середовищі виконання контейнерів та інші. [24] Проста архітектура та легкий об’єм дозволяє швидко встановити та налаштувати дане середовище.

Всі три середовища роботи контейнерів можуть вільно використовуватись в кластері Kubernetes для управління контейнерами, але їх вибір залежить від конкретних потреб і вимог середовища. Docker Engine надає повний функціонал Docker та вищий рівень абстракції, тоді як Containerd та CRI-O фокусуються на базових функціях і мають більш просту архітектуру [23], що робить ці два ПЗ для роботи контейнерів легкими без зменшення ефективності роботи.

Так як для поставленої задачі розгортання розподіленої інфраструктури потрібна швидкодія, простота в управлінні, та не потрібен широкий функціонал (так як це ПЗ буде працювати у зв’язці із іншими вбудованими компонентами кластера), то для роботи в кластері на рівні середовища управління контейнерів прекрасно підійде CRI-O, який має ці всі переваги, і до того ж спеціально розроблений та адаптований під вимоги кластера Kubernetes.

### Вибір Контейнерного мережевого інтерфейсу

Як зазначалось раніше, Контейнерний мережевий інтерфейс (КМІ), або ж CNI, – це служба кластера K8s, задача якої підтримувати зв’язок між всіма вузлами, а також доповнювати функціонал Мережевого проксі. Оскільки кожен вузол та Pod в кластері має свій IP, який необхідно маршрутизувати для повного функціоналу роботи розгорнутих в ньому додатків, то без інтеграції КМІ неможливо реалізувати доналашутвання розгорнутої на етапі ініціалізації мережі Pod`ів. CNI використовує свої інструменти для підтримки роботи такої мережі.

Завдяки підтримці сторонніх КМІ існує велика варіативність у виборі такої служби. Найбільш популярними в цій сфері є такі CNI, як Flannel та Calico.

Flannel, який створений компанією CoreOS, є одним із найпопулярніших CNI плагінів для маршрутизації в кластері. Flannel створює мережу 3 рівня OSI для кожного вузла шляхом запуску спеціального, невеликого за об’ємом, агенту flanneld. [23] Задача цього сервісу полягає у виділені окремої підмережі для кожного хоста із великої, раніше створеної мережі кластера, а також маршрутизації між підмережами. Трафік в рамках одного вузла передається за допомогою протоколів другого рівня OSI (L2). В той же час для передачі даних між вузлами, або ж від контейнера до зовнішньої мережі, відбувається інкапсулювання інформації у UDP пакети третього рівня (L3) для передачі до відповідного місця призначення. [23] Так як весь процес встановлення Flannel полягає у декларативному запуску одного агенту, то можна зробити висновок, що процес встановлення Flannel`у не вимагає багато часу та зусиль на налаштування. Крім того, Flannel має свої власні інструменти маршрутизації трафіку, які вимагають меншого втручання адміністратора, ніж аналогічні технології, наявні у Calico.

Calico, будучи також одним із найпопулярніших рішень в сфері КМІ, відомий своєю продуктивністю, гнучкістю, та більш комплексним у порівнянні з Flannel функціоналом маршрутизації всього трафіку. [22] Так, задачами Calico є не тільки забезпечення з’єднання між хостами, але і підтримка безпеки та надійності передачі даних. [23] Як і з Calico, Flannel можна встановити шляхом декларативного розгортання одного файлу. Flannel пропонує підхід до маршрутизації даних між хостами, при якому в створеній мережі 3 рівня використовується протокол BGP, що має у своїй концепції віртуальній зовнішній роутер. [24] При цьому, для роботи мережі не потрібно додатково загортати дані в пакети вищого рівня, що призводить до більшої швидкодії.

Ключовою різницею між двома Контейнерними мережевими інтерфейсами є їх принцип роботи. Так як раніше обраний MetalLB працює із протоколом маршрутизації BGP тільки у Beta режимі, то це може привести до неочікуваних помилок та нестабільної роботи під час фільтрування вхідного трафіку для вузлів, якщо використовувати ці дві служби в одній зв’язці. З цього випливає той факт, що інтеграція КМІ Calico для роботи мережі із MetalLB не зможе гарантувати стабільність роботи мережі в одній розподіленій інфраструктурі. На відміну від Calico, Flannel, який не поступається популярності та рівню підтримки від розробників, має повну підтримку у MetalLB.

Таким чином доцільніше використовувати для цільового кластера саме КМІ Flannel через повну сумісність із вищеобраним Балансувальником навантаження.

### Планування системи моніторингу та сповіщень

Зазвичай, для розгортання системи моніторингу та сповіщень у кластері K8s використовується стек двох сервісів: служба моніторингу, а також сервіс відображення інформації.

Серед сервісів моніторингу найбільш популярним та продуктивним є Prometheus. Prometheus збирає та зберігає у заздалегідь налаштованому місці метрик стану як компонентів Kubernetes, так і всіх інших процесів і користувацьких додатків всередині кластера. Дане ПЗ зарекомендувало себе простотою використання та стабільністю в своїй роботі протягом років. Крім того, Prometheus підтримує інтеграції великої кількості сторонніх сервісів для візуалізації стану системи у графіках та таблицях.

Серед сервісів візуалізації метрик можна навести таких головних гравців на ринку, як Grafana і Kibana.

Grafana – це інструмент візуалізації метрик, який може бути інтегрованим із будь якою існуючою системою моніторингу та допомагає відображати зібрані метрики у вигляді зрозумілих графіків продуктивності, витрат ресурсів кожним компонентом за певний проміжок часу, тощо. Крім того, Grafana має в своєму спектрі функцій також сервіс відображення сповіщень про критичні помилки у системі, що дозволяє вчасно помітити проблему та запобігти виходу із строю додатку, або ж інколи навіть всього кластера.

Kibana є таким же інструментом відображення даних з таким же функціоналом, але спеціалізується більше на роботі із журналами (логами) працюючих сервісів і надає спрощене уявлення про продуктивність кластера та використання ресурсів для його роботи.

Загалом, проаналізувавши два стеки, можна прийти до висновку, що стек Prometheus/Grafana зазвичай використовується для моніторингу та візуалізації метрик, тоді як використання Prometheus/Kibana в системі призведе до отримання візуалізації спрощених метрик та в більшій мірі спеціалізується на журналі процесів.

Таким чином, після аналізу роботи та спеціалізації двох стеків моніторингу стану кластера було прийняте рішення обрати стек із сервісом моніторингу та збирання метрик Prometheus і службою відображення інформації та сповіщень Grafana, так як при такій комбінації адміністратор системи зможе отримати детальну інформацію як про стан системи, так і матиме змогу отримувати детальні повідомлення у випадку виходу із ладу того чи іншого компоненту кластера чи служби.

### Вибір об'єктного сховища даних

Основна особливість таких сховищ – це те, що для роботи із даними використовується S3-сумісний інтерфейс, що дозволяє зберігати об'єкти та даних з сумісними з S3 додатками і сервісами. Існує велика кількість об'єктних сховищ даних, які можуть бути використані в кластері Kubernetes, серед головних гравців у цій сфері можна виділити наступні:

* Amazon S3 - є одним з найбільш популярних об'єктних сховищ даних. Він є фактичним стандартом для зберігання об'єктів в хмарі AWS.
* Google Cloud Storage – є об'єктним сховищем даних, яке надається в хмарному середовищі Google Cloud.
* OpenStack Swift – це відкрите об'єктне сховище даних, яке може бути використане в приватних хмарах або локальних кластерах.
* MinIO – проект, який відомий своєю простою і швидкою установкою, S3-сумісним інтерфейс та можливістю виконувати роль сховища даних як в хмарному середовищі, так і у локально розгорнутій інфраструктурі.

Кожна з цих платформ має свої особливості та інтеграцію зі своїми власними сервісами та інструментами. Вибір конкретного об'єктного сховища залежить від поставленої задачі та інфраструктури, яку використовує користувач. Так, Amazon S3 та Google Cloud Storage можна обрати у тому випадку, коли інфраструктура кластера розгорнута у хмарі. На противагу цьому, OpenStack Swift та MinIO можна використовувати для будь якої архітектури розгортання кластера.

Ключовим фактором порівняння OpenStack Swift та MinIO є швидкість інсталяції. Так, при однаковому функціоналі, MinIO пропонує швидке встановлення а також не вимагає значних витрат у часі на налаштування,

Таким чином, було прийняте рішення використовувати MinIO у розгорнутому кластері для зберігання результатів телемедичних досліджень.

## Створення та підготовка віртуальних машин до розгортання кластера

Для виконання поставленої задачі було створено три віртуальні машини у середовищі VirtualBox, які моделюватимуть наявність реальної інфраструктури. Налаштування об’єму пам’яті та інших характеристик віртуальної материнської плати наведено нижче на рисунку 2.2.

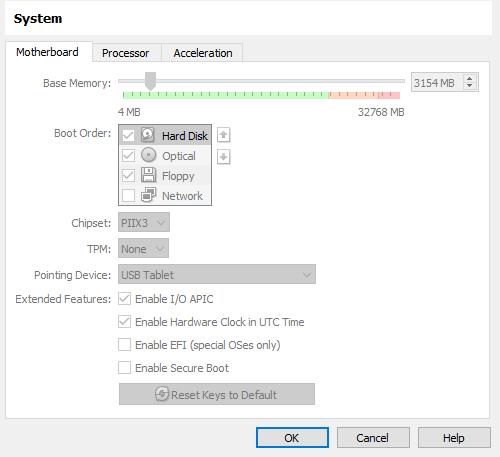


Рисунок 2.2 – Налаштування об’єму пам’яті

Крім того, для кожної віртуальної машини було виділено по 3 ядра із ресурсів персонального комп’ютера.

Результат створення трьох віртуальних машин наведено нижче на рисунку 2.3.

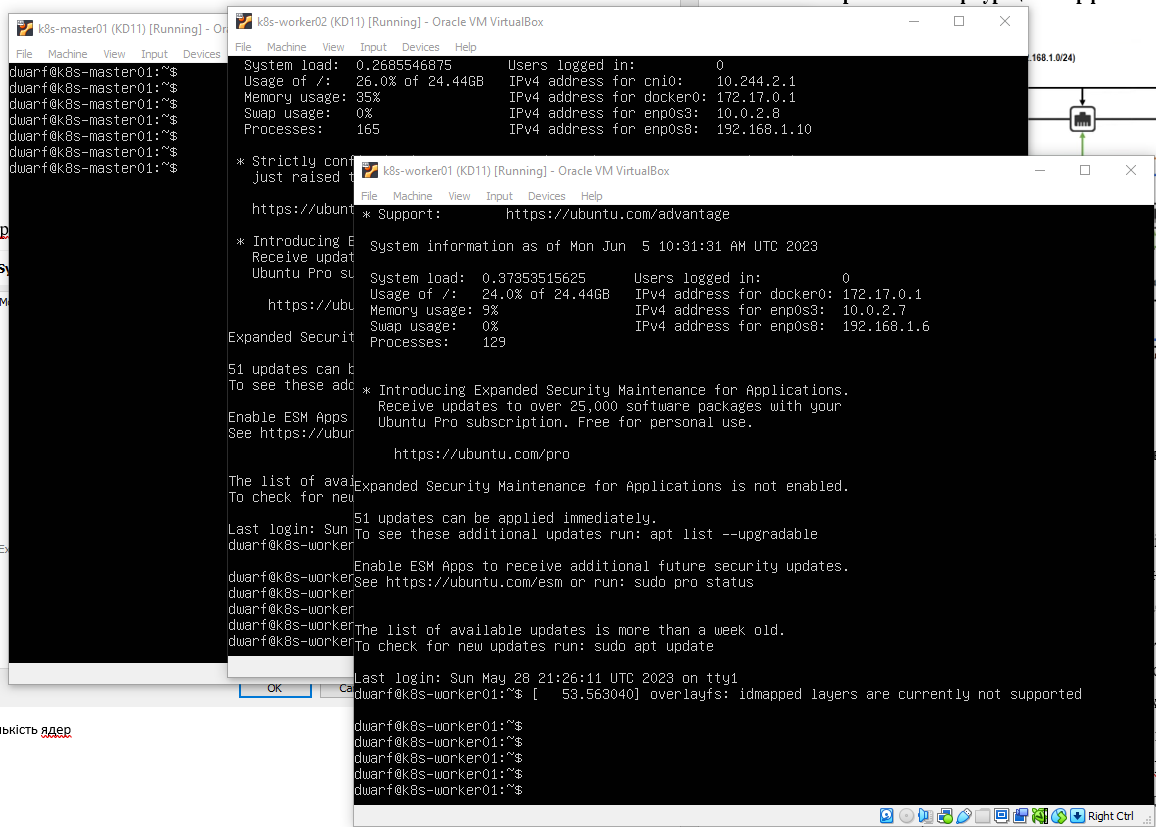


Рисунок 2.3 – Результат створення віртуальних машин

Підбиваючи підсумки, можна навести таблицю, яка містить роль, hostname та характеристики кожної створеної ВМ.

Таблиця 2.1. – IP адреса машин-вузлів кластера

|  |  |  |  |
| --- | --- | --- | --- |
| Hostname | Роль машини | Кількість ядер | Пам’ять машини |
| k8s-master-01 | Вузол керування | 3 | 3154 MB |
| k8s-worker-01 | Робочий вузол | 3 | 3154 MB |
| k8s-worker-02 | Робочий вузол | 3 | 3154 MB |

### Планування принципу взаємодії між хостами та зв'язку кластера із мережею та конфігурація інтерфейсів згідно із планом

Для повноцінної роботи кожного майбутнього вузла було створено два віртуальних адаптери, а також третій був створений програмно під час інсталяції КМІ.

Перший інтерфейс-адаптер називається Nat Network Adapter, та відповідає за зв’язок між всіма вузлами. Всі вузли поєднані у одну мережу 10.10.0.0/24, яка відповідає за передачу запитів між вузлами.

Другий адаптер має тип Мостового, та відповідає за зв’язок кожного вузла із Персональним комп’ютером (під’єднання через ssh-клієнт), на якому розгорнуті ВМ, а також із зовнішньою мережею. Такий інтерфейс створює віртуальний інтерфейс, який використовує наявний фізично існуючий адаптер комп’ютера. В такому випадку спеціальний агент VirtualBox займається фільтрацією трафіку до необхідної точки призначення. Під поняттям «зовнішня мережа» мається на увазі домашня локальна мережа. Таким чином, Мостовий інтерфейс кожного вузла отримує від роутера, який маршрутизує домашню мережу 192.168.1.0/24 , свою власну IP адресу для зв’язку.

Третій інтерфейс створюється та конфігурується CNI, та відповідає за передачу даних у мережі 10.244.0.0/24. [9]

Таким чином, можна навести діаграму роботи інтерфейсів в кластері, яка наведена на рисунку 2.4.

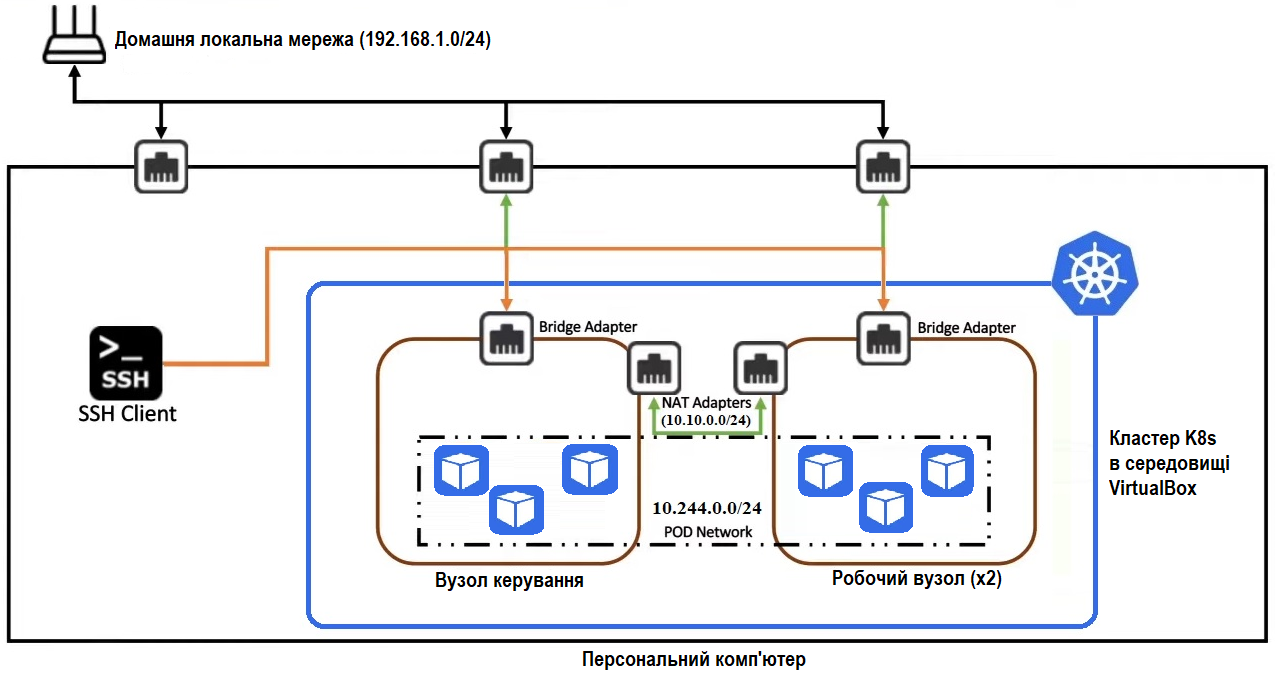


Рисунок 2.4 – Діаграма роботи інтерфейсів в кластері

Середовище VirtualBox дозволяє створити віртуальну мережу для вузлів, а також адаптери дня неї. Результат створення та конфігурацію такої мережі наведено нижче на рисунку 2.5.

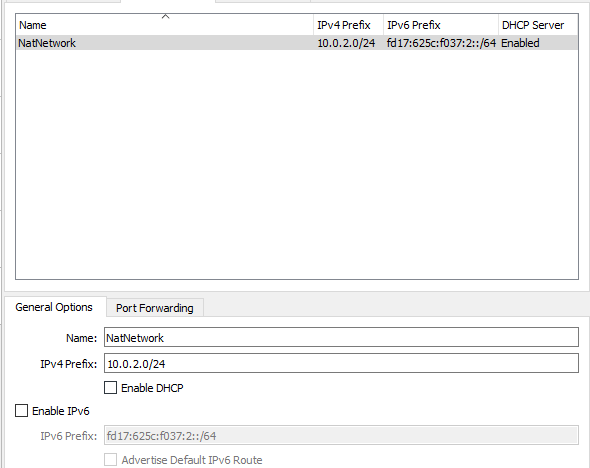


Рисунок 2.5 – Конфігурація створеної NAT мережі

Після створення цієї мережі необхідно вручну в налаштуваннях кожної із машини обрати два інтерфейси. Першим встановлюється мережевий інтерфейс NAT, конфігурація якого наведена нижче на рисунку 2.6.

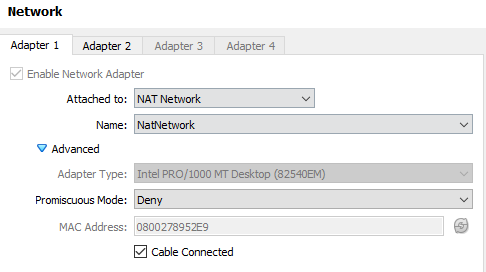


Рисунок 2.6 – Призначення та конфігурація NAT інтерфейсу машини

Другим інтерфейсом налаштовується Мостовий адаптер, або ж Bridged. При виборі параметра Name відбувається вибір реального інтерфейсу Персонального комп’ютера, на якому розгорнуті ВМ. Налаштування Мостового адаптеру наведено нижче на рисунку 2.7.

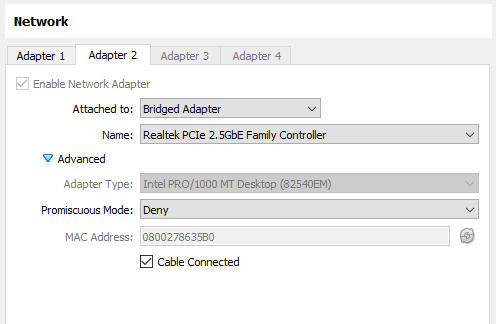


Рисунок 2.7 – Налаштування Мостового адаптеру

Налаштування інтерфейсів можна перевірити шляхом виконання команди ifconfig. Конфігурація наявних інтерфейсів всіх трьох створених машин наведено на рисунках 2.8, 2.9 та 2.10.

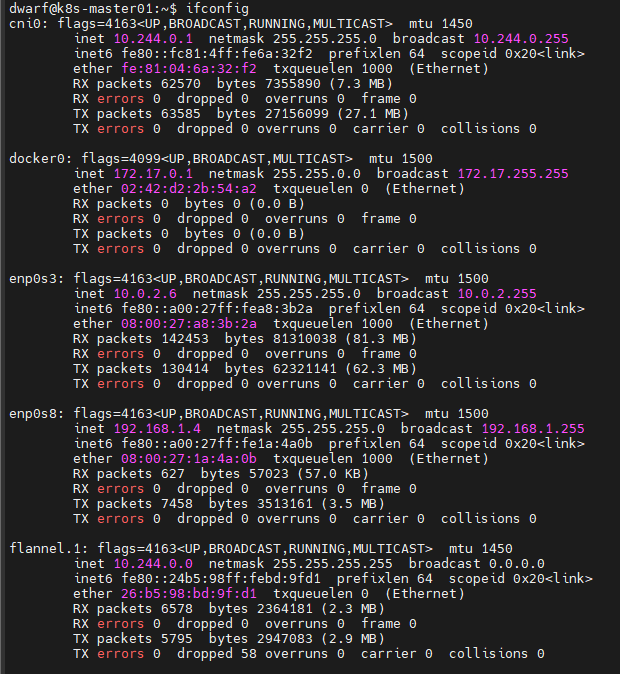


Рисунок 2.8 – Конфігурація інтерфейсів ВМ k8s-master-01

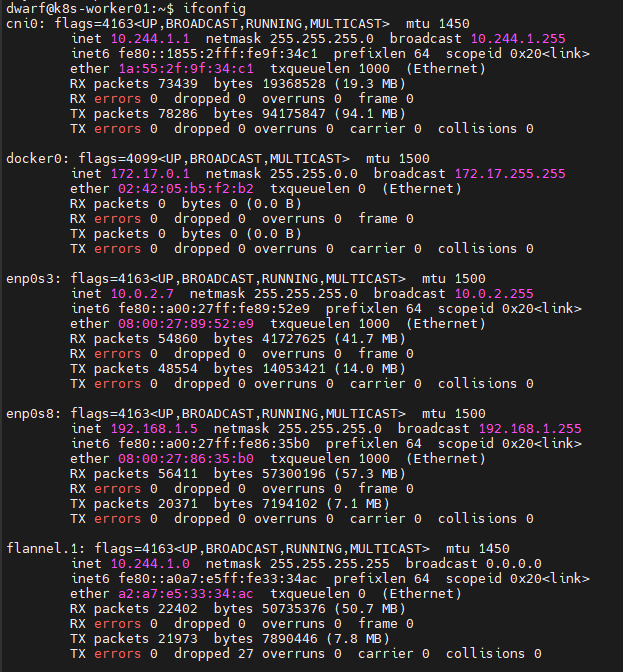


Рисунок 2.9 – Налаштування інтерфейсів ВМ k8s-worker-01



Рисунок 2.10 – Налаштування інтерфейсів ВМ k8s-worker-01

Із наведених скріншотів можна побачити, що кожен із хостів має кілька встановлених інтерфейсів, серед яких інтерфейси enp0s3 та enp0s8 це описані раніше NAT Network та Мостовий адаптери відповідно. Так, майбутньому вузлу керування була виділена IP адреса 10.0.2.6 для зв’язку всередині майбутнього кластера, а також 192.168.1.4 для зв’язку із Персональним комп’ютером, на якому запущені мережі, а також із зовнішньою мережею (Локальна домашня мережа). Наступним двом машинам були призначені IP адреси 10.0.2.7 та 10.0.2.8 для першого адаптера, а також 192.168.1.5 та 192.168.1.6 для другого відповідно. Крім того, так як цей скріншот зроблений вже після всіх необхідних налаштувань, то тут присутні такі інтерфейси як CNI і Flannel, які встановлюються у розділі 3.2.

Підбиваючи підсумки, можна навести таблицю із адаптерами та виділеними для них IP адресами для кожного майбутнього вузла кластера.

Таблиця 2.2. – IP адреса машин-вузлів кластера

|  |  |  |
| --- | --- | --- |
| Hostname машини | IP адреса enp0s3 | IP адреса enp0s8 |
| k8s-master-01 | 10.0.2.6 | 192.168.1.4 |
| k8s-worker-01 | 10.0.2.7 | 192.168.1.5 |
| k8s-worker-02 | 10.0.2.8 | 192.168.1.6 |

На всіх машинах була здійснена перевірка доступу до мережі Інтернет шляхом введення команди ping google.com, результати такої перевірки на вузлі k8s-master-01 наведено на рисунку 2.11.

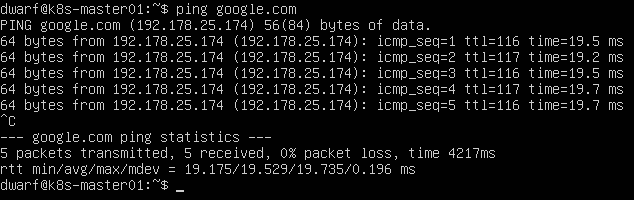


Рисунок 2.11 – Результати команди ping, введеної для k8s-master-01

### Відкриття необхідних для роботи кластера портів

Кластер, запущений використовуючи локальну інфраструктуру, вимагає налаштування файрволу для забезпечення підключень за певними портами. Так, для повноцінної роботи вузла керування необхідно відкрити порти 6443 для Kubernetes API серверу, 2379-2380 для взаємодії із etcd сховищем, 10250 для Kubelet API вузла керування, 10257 для Менеджера контролерів, а також 10259 для Планувальника. Для робочого вузла необхідно відкрити порт 10250 для роботи його Kubelet API, порти 2379-2380 для роботи із Etcd, а також порти 30000-32767 для сервісів користувача. Варто зазначити, що порти 30000-32767 є стандартними портами для роботи сервісів, але користувач може змінити ці порти на будь-які доступні у конфігурації кластера.

Для відкриття вищеописаного переліку портів необхідно налаштувати стандартну утиліту ufw – файрвол в системі Linux, який фільтрує весь трафік. Конфігурація відбувається шляхом вписання наступної команди: ufw allow <номер порту>/<протокол передачі даних>. [25]

Крім того, так як для роботи з вузлами використовується ssh-клієнт для можливості передачі скриптів .bash, то під час формування файлу було додано відкриття порту для такого зв’язку. Під портом для ssh мається на увазі порт 22, який стандартно використовується для протоколу віддалених підключень. Вміст файлу із налаштуванням портів для вузла керування наведено нижче на рисунку 2.12.

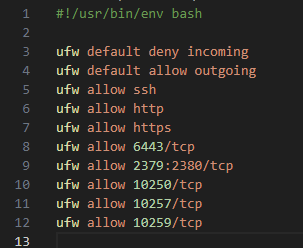


Рисунок 2.12 – Вміст скриптового файлу .bash налаштування портів для вузла керування

Як видно із рисунка, до вмісту цього файлу входять також відкриття портів http (80) та https (443), які використовуються для пересилання запитів та відповідей на ці запити.

Результат виконання команд наведено на рисунку 2.13.

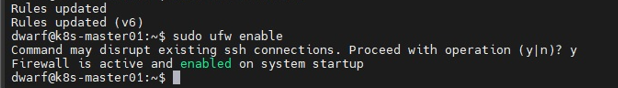


Рисунок 2.13 – Результат налаштування портів

Після цього можна під’єднатися до створених віртуальних машин за допомогою ssh-клієнту MobaXterm. Для підключення було використано Мостовий адаптер, якому назначається IP адреса в домашній мережі. Як було зазначено раніше, вузол керування k8s-master-01 має IP адресу 192.168.1.4, яку можна використати для підключення.

Після успішного підключення, рисунок 2.5, можна вводити команди із буфера обміну, а також передавати скриптові файли налаштування, про які зазначалось раніше, у файлову систему ВМ, яку можна побачити зліва від терміналу.

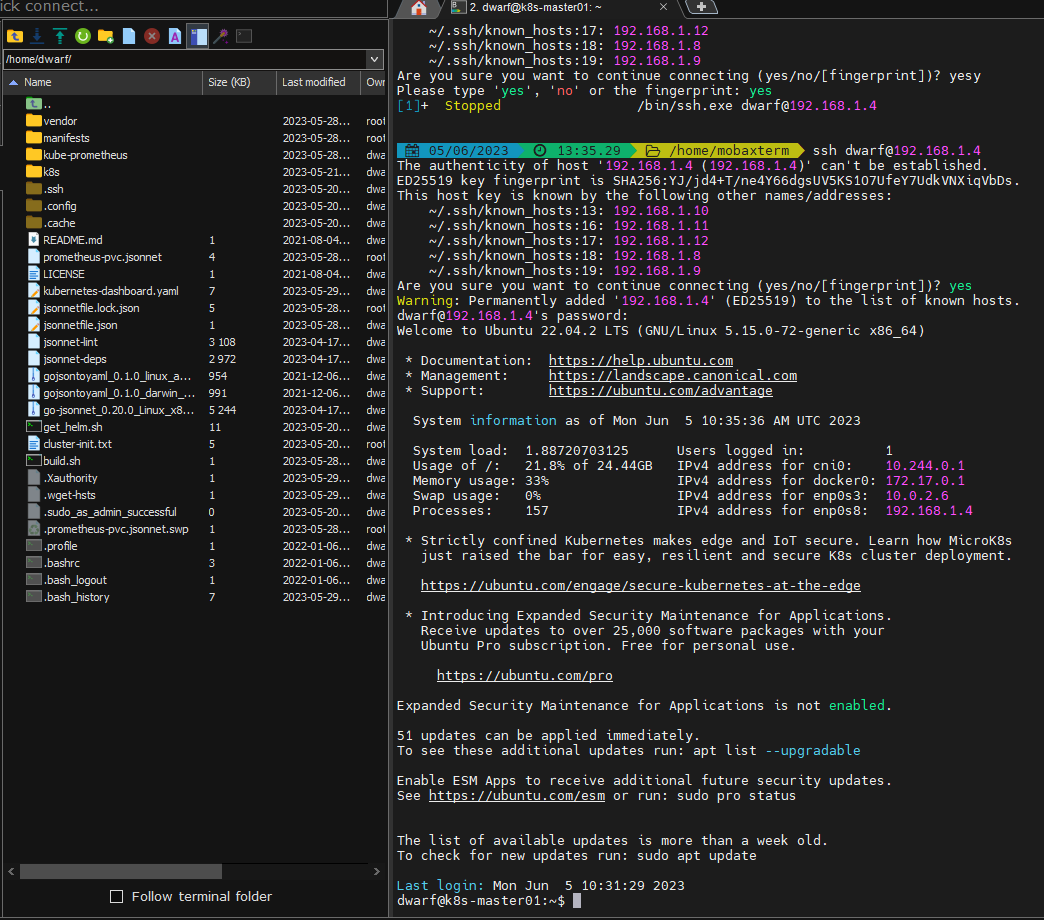


Рисунок 2.14 – Успішне підключення до k8s-master-01

### Вимкнення Swap

Swap file, або ж файл підкачки, відповідає за збереження кешу всіх процесів. Причому відбувається автоматичне збереження як кеш чистих сторінок пам’яті (Clean cache page - сторінка, в якій зберігаються закешовані дані файлу, який не змінювався) так і кеш грязних сторінок (Dirty cache page – сторінка, в якій зберігаються дані файлу, вміст якого вже змінився). [26] Таким чином, даний файл із часом накопичуватиме велику кількість непотрібної інформації для роботи кластера, що позначиться на продуктивності роботи всього вузла. Розробники Kubernetes знають про цю проблему, тому на етапі ініціалізації вузла керування відбувається перевірка на наявність увімкненої підкачки, і у разі її виявлення – весь процес створення кластера завершується помилкою.

Для вирішення задачі вимкнення Swap`у було першочергово видалено сам файл підкачки swap.img, введено команду sudo swapoff -a, а також відредаговано конфігураційний файл /etc/fstab, який відповідає за файлову систему машини. На рисунку 2.15 наведено цей файл, для відключення Swap було закоментовано рядок файлу із згадкою про видалений файл swap.img.

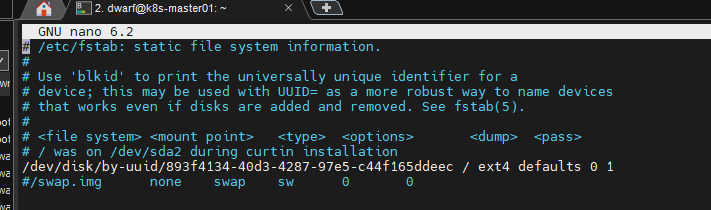


Рисунок 2.15 – Конфігураційний файл /etc/fstab в текстовому редакторі nano

### Встановлення компонентів Kubernetes та Docker

Встановлення компонентів K8s та Docker було реалізовано за допомогою скриптових файлів inst-k8s.sh та inst-docker.sh, тексти кодів яких наведені у Додатку А даної кваліфікаційної роботи. Варто зазначити, що для встановлення було обрано версію 1.26.2 всіх компонентів. Дана версія не є найновішою, але вважається найстабільнішою серед останніх, а так як задача розгортання розподіленої інфраструктури для телемедичних сервісів вимагає стабільності та гарантування високої доступності всієї системи, то було прийняте рішення обрати саме цю версію.

Для перевірки працездатності можна ввести запит на виведення інформації про встановлені компоненти. Результати виконання наведені на рисунку 2.16.

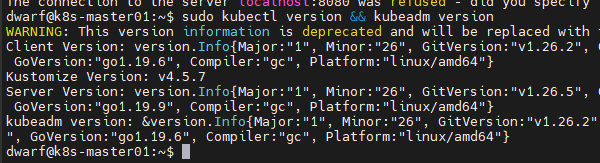


Рисунок 2.16 – Запит із надання інформації про компоненти

Із рисунку можна побачити, що і Kubectl і Kubeadm були успішно встановлені та мають однакову версію, яка була зазначена у скриптовому файлі раніше – 1.26.2.

### Встановлення середовища керування контейнерами

На етапі планування було обрано середовище CRI-O, яке вимагає додаткового налаштування системних параметрів (sysctl) Kubernetes перед встановленням. Так, для вирішення цієї задачі було створено автоматизований скриптовий файл iptables.sh, який поєднує у собі всі необхідні для налаштування команди. Вміст даного файлу наведено нижче на рисунку 2.17.

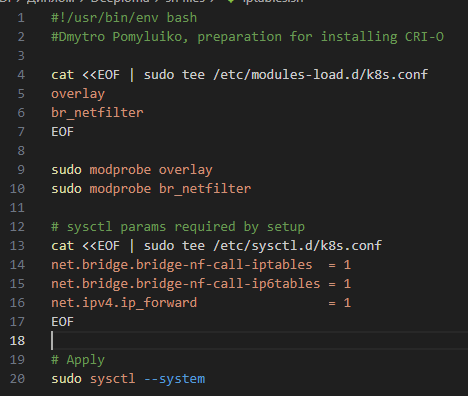


Рисунок 2.17. – Вміст файлу налаштування

Використовуючи підхід pipe, були відредаговані конфігураційні файли завантаження модулів K8s на роботу у постійному режимі, а також дозволена робота із таблицями IPv4 та IPv6.

Після цього можна приступати до завантаження та запуску CRI-O. Для цієї задачі було використано файл inst-CRI-O.sh, який містить стандартні команди завантаження apt-get, apt-install та apt-update. Текст цього скриптового файлу наведений у Додатку А.

## Висновки по другому розділу

Під час написання цього розділу було розглянуто архітектуру Kubernetes, за результатами аналізу було підтверджено той факт, що K8s має комплексну архітектуру із великою кількість взаємопов’язаних компонентів. Після огляду архітектури кластера був зроблений огляд існуючих сервісів для підтримки роботи кластера. За результатами такого дослідження, був обраний кінцевий список технологій, які будуть використовуватися у розгорнутому кластері.

В кінці була проведена робота із створення та налаштування ВМ. Під час виконання цього процесу було проаналізовано принцип взаємодії трьох машин у кластері як між собою, так і з зовнішньою мережею. В результаті цього дослідження був обраний підхід з використанням двох інтерфейсів, які дозволять кожній ВМ одночасно мати зв’язок як із зовнішнім середовищем, так і з іншими машинами у кластері. Завантаження середовища виконання контейнерів CRI-O було останнім кроком у підготовці віртуальних машин до об’єднання у один кластер. Після цього можна перейти до розгортання та налаштування розподіленої інфраструктури для поставленої задачі телемедичного моніторингу.

# РОЗГОРТАННЯ ТА НАЛАШТУВАННЯ КЛАСТЕРА СЕРВЕРІВ ДЛЯ ЗАДАЧ ТЕЛЕМЕДИЧНИХ МОНІТОРИНГОВИХ ДОСЛІДЖЕНЬ

## Розгортання кластера

Після налаштування всіх віртуальних машин було ініціалізовано перший вузол керування майбутнього кластера, використовуючи скриптовий файл, вміст якого наведений нижче на рисунку 3.1.

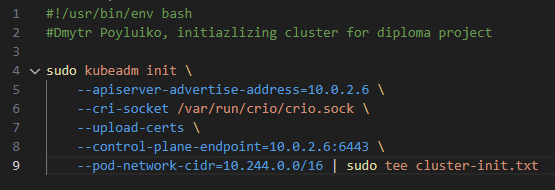


Рисунок 3.1 – Вміст скриптового файлу ініціалізації вузла керування

Було додано спеціальні параметри, що задають необхідні значення для роботи кластера.

Так, --apiserver-advertise-address=10.0.2.6 відповідає за налаштування IP адреси, а з нею і інтерфейсу, який буде використовуватись API сервером для прослуховування запитів. [9] Даний параметр необхідно визначити в тому випадку, коли необхідний інтерфейс не є першим в списку (тобто стандартним). Як раніше описувалось, під час налаштування віртуальних машин було створено два інтерфейси, при цьому інтерфейс для зв’язку між майбутніми вузлами, який також необхідний для роботи Kube-Apiserver`у, був визначений другим. Якщо не дописати цей параметр у команду ініціалізації кластера, то за умовчуванням буде використовуватись перший наявний адаптер, а це Мостовий адаптер, який не підходить під такі задачі, тому це може привести до численних помилок під час роботи кластера.

Другим параметром було вказано шлях до контейнерного двигуна, який буде використовуватись у основі кластера.

Під третім параметром було вказано завантажити та використати сертифікати безпеки, що вкрай необхідні для передачі даних в зовнішню мережу.

Параметр --control-plane-endpoint=10.0.2.6:6443 відповідає за встановлення єдиної спільної точки входу трафіку для всіх вузлів керування. [10] Якщо цей параметр не вказати, то буде неможливо в майбутньому додавати інші вузли керування для підвищення рівня доступності площини керування.

Останній параметр задає мережу між Pod`ами, в даному випадку було обрано 10.244.0.0/16 так як цю мережу використовує Мережевий Контейнерний Інтерфейс Flannel [23], що дозволить зменшити витрату часу на додаткове переналаштування.

В кінці використовується так званого «конвеєр» (від англ. pipe), що дозволить перенаправити всі log’и та результат виконання команди у створений файл cluster-init.txt. Цей файл використовується для зручного перегляду всієї інформації про перебіг процесу створення вузла керування, а також виокремлення спеціального токену, який дозволить приєднати робочі вузли для створеного кластера.

На рисунку 3.2 наведений вміст файлу cluster-init.txt, повна версія якого наведена у Додатку А кваліфікаційної роботи.

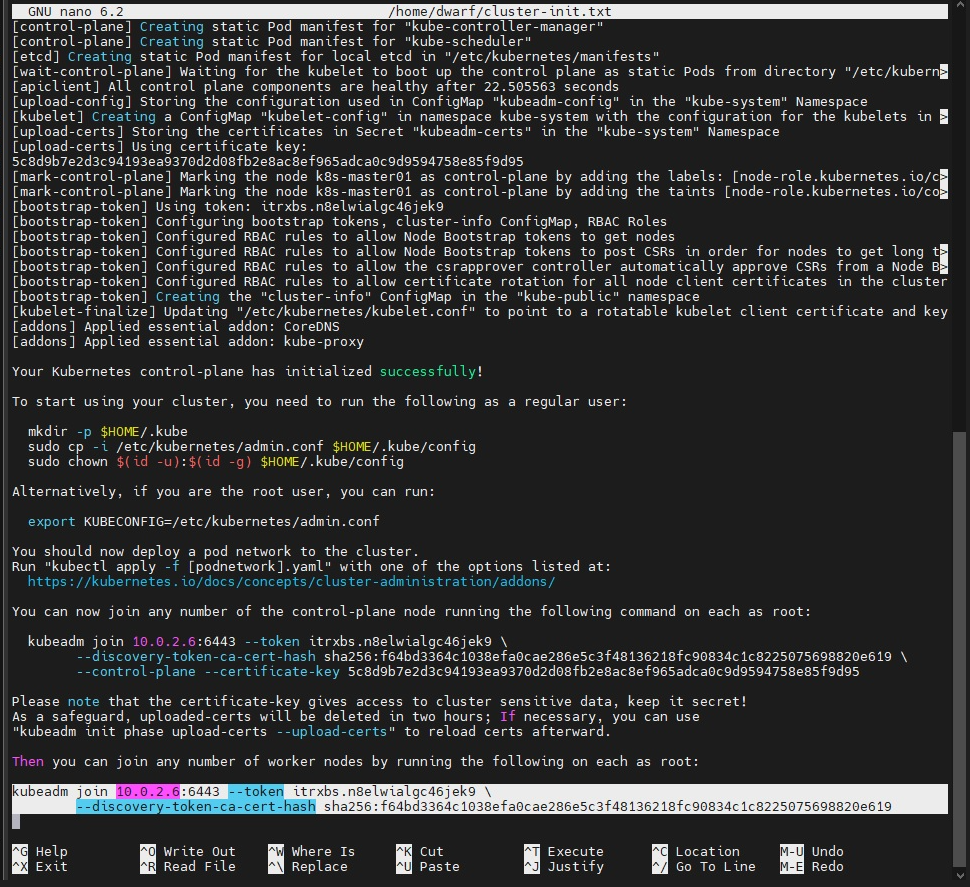


Рисунок 3.2. – Вміст файлу cluster-init.txt

Із вмісту файлу видно, що вузол керування був успішно ініціалізований. В кінці файлу можна побачити дві команди: першу можна використати для приєднання додаткових вузлів керування, а другу, яка виділена, – для робочих вузлів. Скопіювавши цю команду можна перейти до двох віртуальних машин, які планувались як робочі, і запустити процес приєднання. Результат виконання цього процесу наведений на рисунку 3.3.

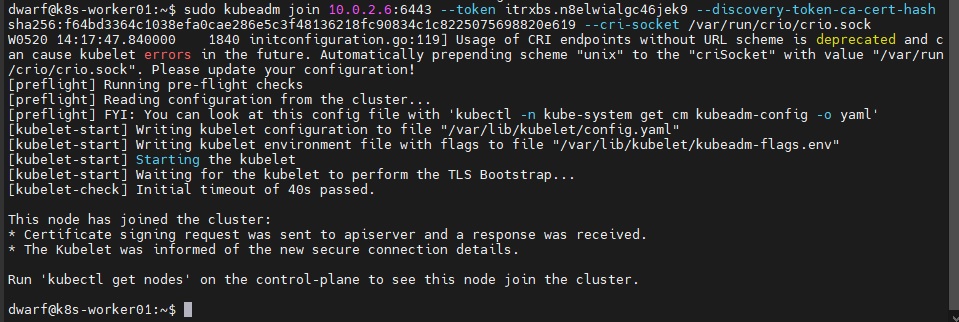


Рисунок 3.3 – Результат приєднання одного із робочих вузлів

Із виводу результату на консоль видно, що робочий вузол успішно приєднаний до кластера. Після цього, повторивши аналогічні дії із приєднання останньої ВМ, процес розгортання кластера можна вважати завершеним, і тепер необхідно перейти до налаштування сервісів для забезпечення повноцінної роботи кластера.

## Встановлення CNI

Як було раніше зазначено, CNI відіграє значну роль у роботі кластера. На підтвердження цього факту можна навести рисунок 3.4, який показує стан компонентів системи без встановленого КМІ.

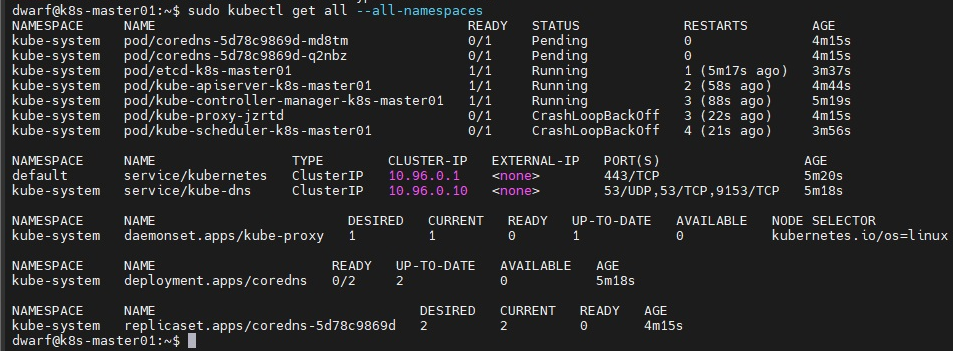


Рисунок 3.4 – Стан системи без встановленого КМІ

Як бачимо, без інтеграції КМІ такі ключові служби як Мережевий проксі або Планувальник знаходяться у стані виходу із ладу. Таким чином, для продовження роботи встановлюється CNI Flannel, який був обраний за результатами порівняння у другій частині.

Встановлення відбувається за допомогою використання файлу inst-CRI-O.sh, частковий вміст якого наведений на рисунку 3.5.

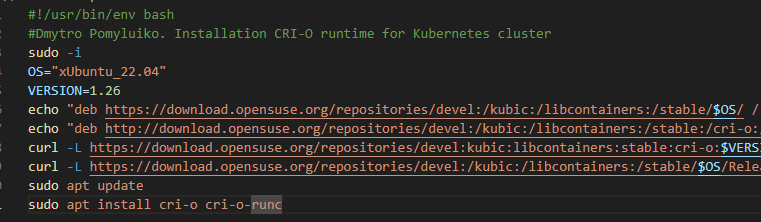


Рисунок 3.5 – Файл inst-CRI-O.sh

Запуск CRI-O реалізований за допомогою скриптового файлу services-CRI-O.sh, вміст якого наведений на рисунку 3.6.

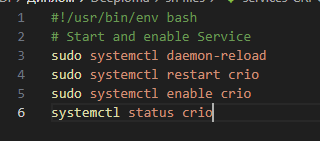


Рисунок 3.6 – Файл services-CRI-O.sh

Повні тексти цих файлів наведені у Додатку А кваліфікаційної роботи.

Після успішного встановлення Flannel на кожному вузлі розгортає свій Pod, який відповідає за маршрутизацію трафіку. Перелік наявних Pod`ів Flannel у системі наведений нижче на рисунку 3.7.

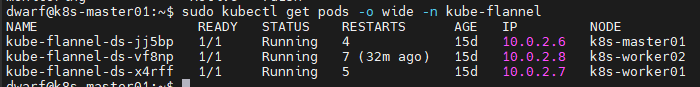


Рисунок 3.7 – Перелік Pod`ів Flannel

Як видно із рисунку, всі три Pod`и успішно запустились. Після цього можна перевірити стан вузлів, ввівши команду kubectl get nodes -o wide, рисунок 3.8.

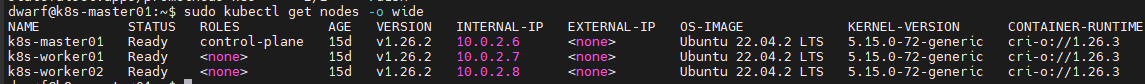


Рисунок 3.8 – Стан вузлів після встановлення CNI

Даний рисунок показує, що всі наявні вузли у системі мають статус Ready, і готові до запуску на них сервісів. Крім того, варто звернути увагу на CONTAINER-RUNTIME, який підтверджує успішне встановлення середовища роботи контейнерів CRI-O на всіх вузлах кластера.

Після цього можна перейти до інтеграції та налаштування сервісів всередині кластера.

## Встановлення пакетного менеджера Helm

Першим таким сервісом в кластері буде Helm. Helm – це пакетний менеджер для керування сервісами та пов’язаними службами в кластері Kubernetes. [26] Це ПЗ дозволяє легко встановлювати та оновлювати сервісні служби, компоненти та користувацькі додатки, розгорнуті в кластері. Використовуючи Helm можна налагодити принцип CI/CD, що дозволяє зберігати та керувати версіями конфігурацій і додатків, а також автоматично оновлювати раніше розгорнуті сервіси в кластері. [26]

Основна концепція Helm - це "чарти" (charts). Чарти – це репозиторій, який містить всю інформацію для налагодження автоматичного розгортання додатків в Kubernetes. Чарти містять в собі файли конфігурації, шаблони ресурсів Kubernetes, значення важливих параметрів та всі необхідні залежності. [26] Для оновлення сервісів для телемедичних задач можна використовувати власний чарт, та завантажувати звідти оновлення програми, розгорнутої в кластері.

Для встановлення пакетного менеджеру Helm було передано на мастер-вузол скриптовий файл, inst-helm.sh, вміст якого наведений у Додатку А. Результат виконання даного файлу наведений нижче на рисунку 3.9.

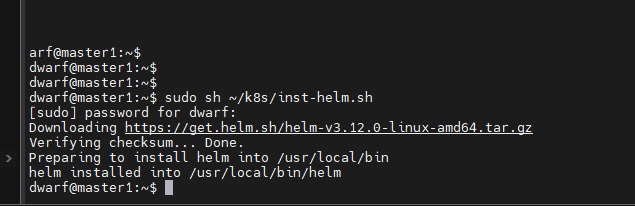


Рисунок 3.9 – Встановлення Helm

## Встановлення об’єктного сховища MinIO

Перед встановленням необхідно створити сертифікат TLS, який буде використовувати об’єктне сховище. Для цього створюєтсья спеціальний об’єкт Secret, який зберігає конфігураційні файли, доступ до яких має тільки адміністратор системи. Вміст цього Secret наведений нижче на рисунку 3.10.

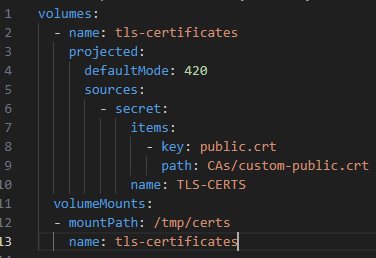


Рисунок 3.10 - Вміст файлу із сертифікатом TLS

Після цього можна перейти до процедури встановлення MinIO. Для цього використовується файл inst-minio.sh, вміст якого наведений на рисунку 3.11.

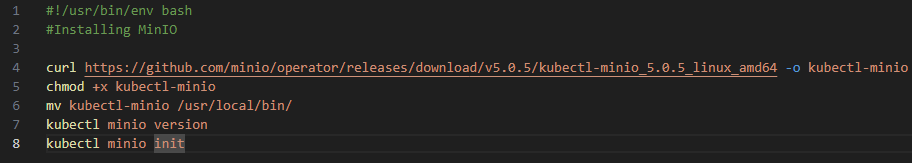


Рисунок 3.11 – Файл inst-minio.sh

Після цього необхідно в конфігурації MinIO дізнатися порти для підключеня, рисунок 3.12.

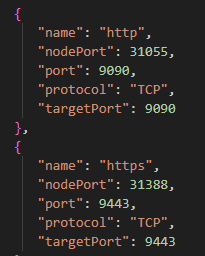


Рисунок 3.12 –Інформація про порти MinIO

Таким чином, із рисунка витікає те, що для підключення до панелі керування базою даних можна використати порти 9090 та 9443 для підключеня по http та https відповідно.

Так як MinIo був розгорнутий на вузлі керування, то до панелі MinIO можна підключитися ввівши IP адресу вузла керування та один із вищезазначених портів.

В даному випадку було використано 192.168.1.4:9090. Результат наведений на рисунку 3.13.

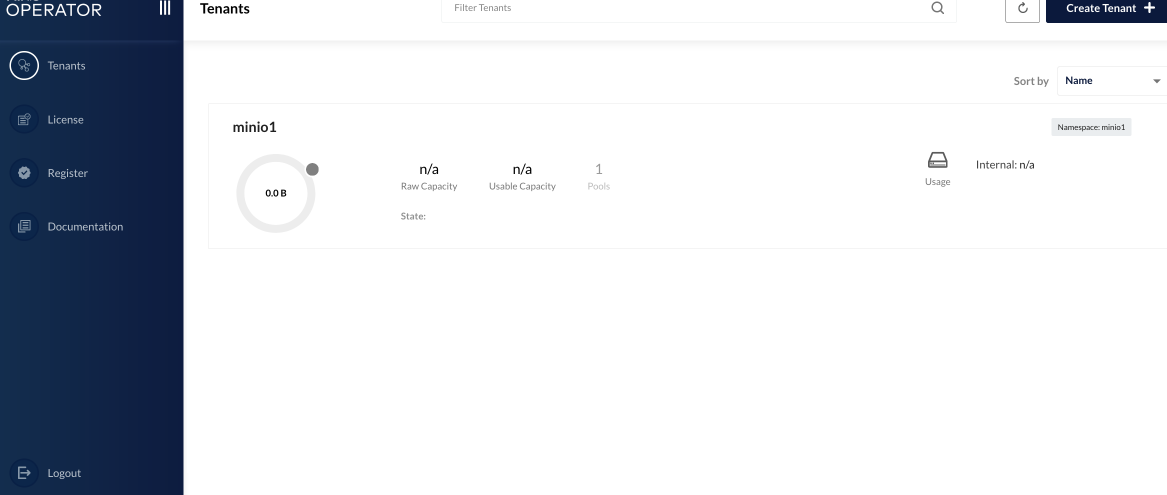


Рисунок 3.13 – Головне меню MinIO Dashboard

## Встановлення зовнішнього балансувальника MetalLB

Встановлення MetalLB відбувається шляхом виконання скриптового файлу, який використвоує пакетний менеджер Helm для завантаження та розгортання. Вміст скриптового файлу із командами наведений нижче на рисунку 3.14.

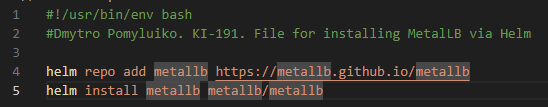


Рисунок 3.14 – Встановлення MetalLB

Під час інсталяції MetalLB створив свій простір імен. Результати перевірки наявності такого простору імен відображені на рисунку 3.15.



Рисунок 3.15 – Простір імен MetalLB в системі

Після інатляції балансувальник навантаження вимагає додаткового налаштування. Для налаштування був створений файл mlb-ip.yaml, вміст якого наведений на рисунку 3.16.

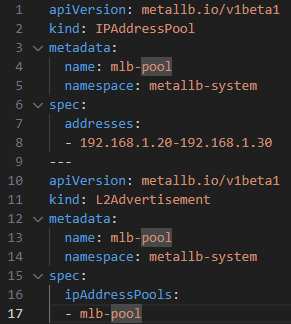


Рисунок 3.16. – Конфігураційний файл mlb-ip.yaml

В даному файлі визначається простір IP адрес, які MetalLB буде використовувати для своєї роботи. В даному випадку було прийняте рішення «зарезервувати» адреси 192.168.1.20-192.168.1.30, які будуть єдиними точками входу трафіку для розгорнутих сервсів. Крім того, нижче у файлі зазначається протокол, який буде використовуватись під час роботи. В даному випадку обрано протокол L2, для якого був створений об’єкт L2Advertisment, задача якого оповіщати сервіси та призначати їм IP адреси.

Результат прийняття цього файлу у систему наведений нижче на рисунку 3.17. Крім цього, також додатково було виведено список наявних Pod`ів у просторі імен metallb-system.

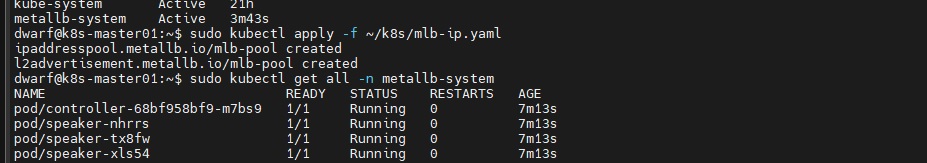


Рисунок 3.17 – Встановлення конфігураційного файлу та список Pod`ів MetalLB в системі.

Як видно із рисунку, і зазначений перелік адресів, і об’єкт L2Advertisment були успішно прийняті кластером, що призвело до створення трьох Pod`ів по один на кожен вузол. Ці Pod’и називаються Speakers і відповідають за передачу трафіку вузлам.

Крім того, було здійснений вивід всіх ресурсів, наявних у просторі імен metallb-system, рисунок 3.18.

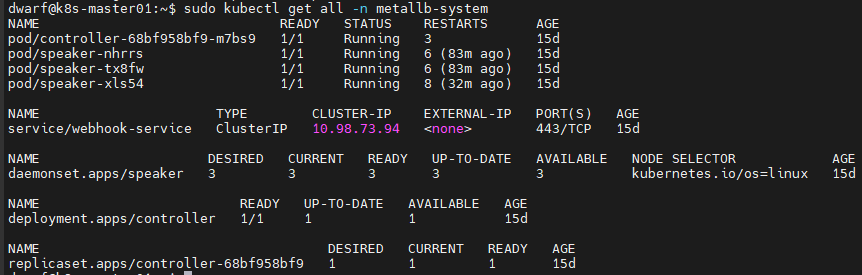


Рисунок 3.18 – Перелік всіх ресурсів у просторі імен metallb-system

## Встановлення системи моніторингу та сповіщень

Встановлення стеку Prometheus/Grafana відбувається шляхом клонування відповідного репозиторію на GitHub, а також прийняттям у кластер необхідних конфігураційних файлів із репозиторія.

На рисунку 3.19 відображено процес встановлення всіх компонентів як Prometheus, так і Grafana.

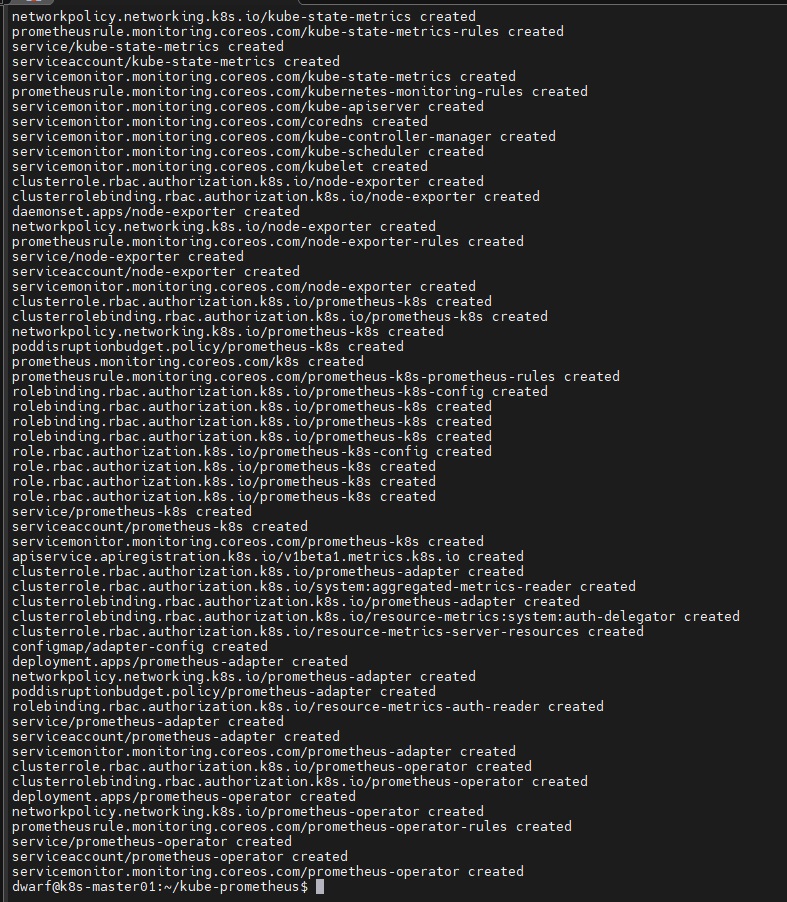


Рисунок 3.19 – Процес встановлення стеку моніторингу

Після процесу встановлення було створено простір імен для цього стеку, результат виведення інформації по простір monitoring наведений на рисунку 3.20.



Рисунок 3.20 – Виведення інформації про простір імен monitoring

Крім того, варто зауважити, що для повноцінної роботи системи моніторингу і сповіщень необхідно видалити спеціальні мережеві правила, які встановлює Kubernetes для простору імен monitoring.

Результат видалення наведений на рисунку 3.21.

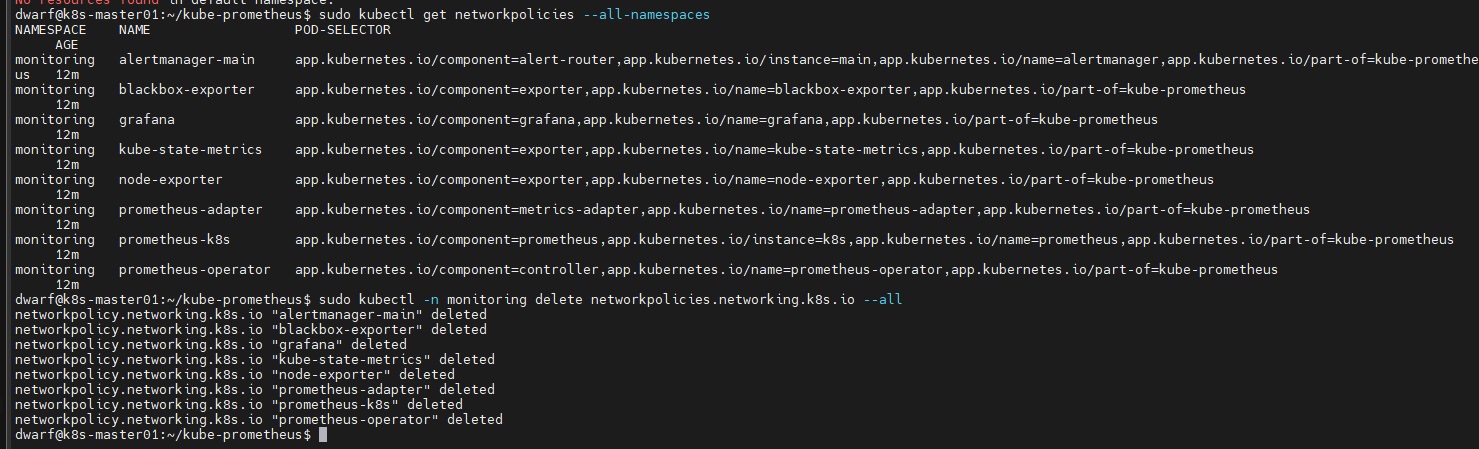


Рисунок 3.21 видалення мережевих правил для простору імен monitoring

В процесі інсталяції було створено декілька сервісів, які користувач зможе використовувати для робот із метриками. До таких сервісів відносяться Prometheus AlertManager, Prometheus-metrics та Grafana. Список всіх наявних сервісів, які були створені в просторі імен monitoring наведено на рисунку 3.22.

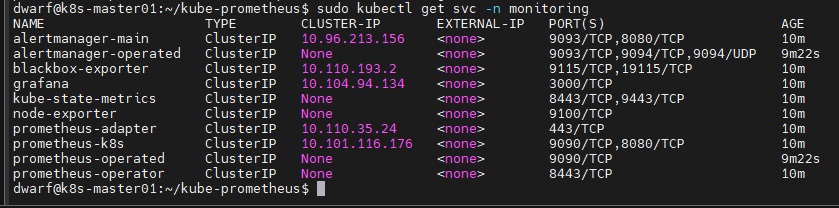


Рисунок 3.22 – Перелік створених сервісів для простору імен monitoring

Список всіх наявних ресурсів у просторі імен monitoring наведено нижче на рисунку 3.23.

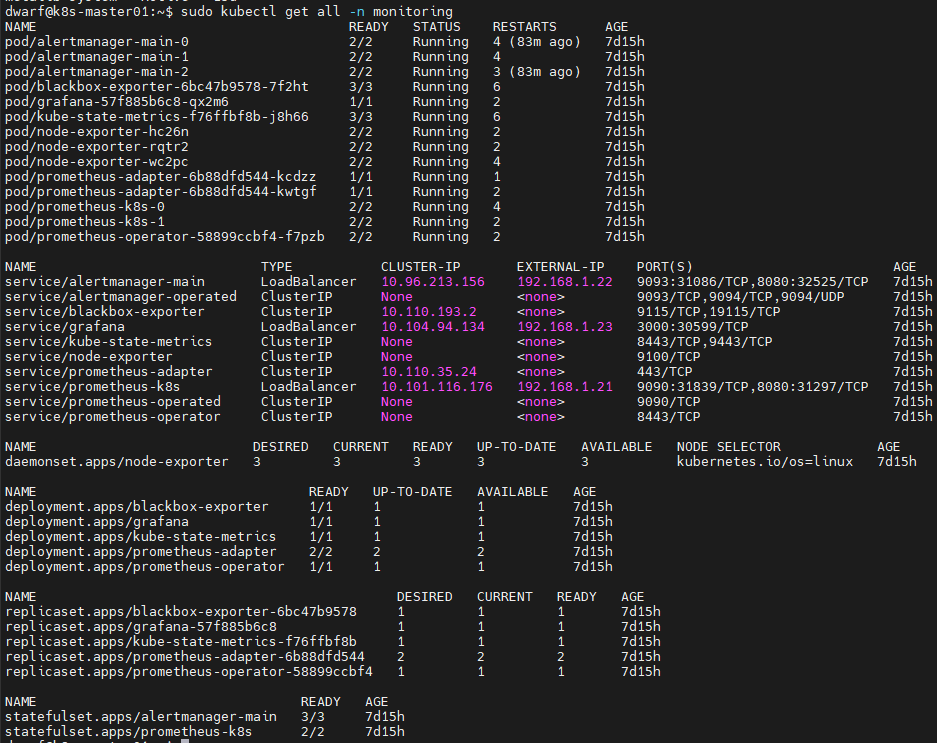


Рисунок 3.23 – Перелік всіх ресурсів у просторі імен monitoring

## Перевірка працездатності реалізованої системи

Для перевірки працездатності було створено Deployment тестового додатку. В Kubernetes Deployment – це назва колекції ресурсів, які виділені на підтримку визначеного користувачем додатку. Визначається Deployment декларативно, як і більшість сервісів, ресурсів та об’єктів кластера. В ролі тестового додатку буде виступати один із найпопулярніших веб-серверів – Nginx. Вміст файлу depl-nginx.yaml Для створення Development наведено на рисунку 3.24.

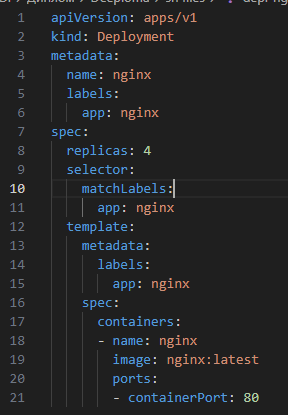


Рисунок 3.24 – Вміст файлу delp-nginx.yaml

Вищенаведений рисунок є прикладом стандартного Deployment у кластері, і містить мінімальний набір параметрів для конфігурації.

У рядку 4 під ключем name вказується назва Deployment, крім того, можна додати користувацькі мітки (англ. Labels), що дозволять при великій кількості одночасно запущених сервісів та Deployment`ів як орієнтуватися у списку ресурсів, так і фільтрувати його для пошуку необхідної інформації.

Варто звернути уваги на перший блок spec. Так як Kubernetes підтримує ReplicaSet, то користувач може вказати необхідну кількість реплік, і K8s буде підтримувати це значення на постійній основі. В тестових цілях була вказана цифра 4, наявність такої кількості Pod`ів буде перевірено пізніше. Крім того, головним параметром є matchLabels, що відповідає за вибір контейнера з нижче описаного шаблону. Так, даний Deployment візьме за основу аткий шаблон контейнера, який має мітку app: nginx.

В блоці template вказано шаблон контейнера, який необхідно запустити. Тут вказана метадата із назвою шаблону, а також у блоці spec наведений образ та конфігурація контейнера, який необхідно запустити. Так як було обрано образ контейнера nginx, то в полі image вказана даний образ, а також його версія. Крім того, створений Deployment буде взаємодіяти із запитами через відкритий на початку роботи порт 80, який зазвичай відповідає за передачу пакетів HTTP.

Після розгортання Deployment відкриємо список Pod`ів, рисунок 3.25.

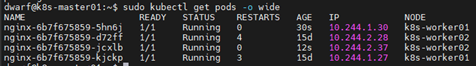


Рисунок 3.25 – список наявних Pod`ів в системі.

Як бачимо, Kubernetes створив таку ж кількість Pod`ів, яку було вказано у файлі. Якщо видалити перші два Pod`и, рисунок 3.26, то K8s максимально швидко створить нові, на заміну старим.

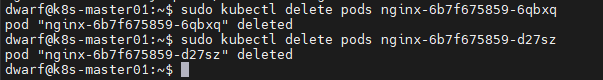


Рисунок 3.26 – видалення двох Pod`ів із Development

Перелік Pod`ів nginx після видалення двох реплік наведено на рисунку 3.27.

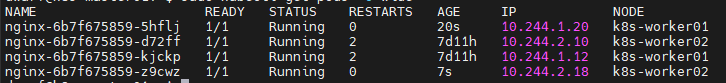


Рисунок 3.27 – Перелік Pod`ів nginx після видалення перших двох

Kubernetes після видалення двох Pod`ів помітив відсутність необхідної кількості репліку в системі, і створив два нових Pod`и на заміну видаленим, які поміщені на вузли k8s-worker-01 та k8s-worker-02 відповідно. Із часу роботи Pod`ів видно те, що в такому переліку перший і останній Pod`и нові.

Після цього було створено сервіс, який використовує балансувальник навантаження MetalLB для роботи із розгорнутим Deployment nginx. Конфігураційний файл такого сервісу наведений нижче на рисунку 3.28.



Рисунок 3.28 – Конфігураційний файл балансувальника навантаженя для додатку nginx

Даний сервіс має тип LoadBalancer, що дозволяє використовувати ресурси встановленого балансувальника навантаження під час фільтрації та перерозподілу трафіку. Крім того були призначені порти для http та https підключень: 80 та 443 відповідно.

Відкривши перелік створених сервісів, рисунок 3.29, можна побачити створений сервіс для nginx. Як видно, такий сервіс отримав зовнішню IP адресу, яку можна використовувати для зв’язку із тестовим додатком nginx.

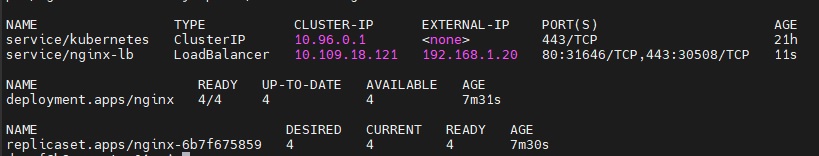


Рисунок 3.29 – Результат створення сервісу LoadBalancer

Для перевірки роботи цього сервісу можна перейти в браузер будь якого персонального комп’ютера або пристроя, який під’єднаний до локальної мережі, і ввести зовнішню IP адресу сервісу nginx-lb. В даному випадку балансувальник навантаження виділив для новоствореного сервісу першу адресу із заданого раніше переліку – 192.168.1.20. Результат тестування роботи наведений нижче на рисунку 3.30

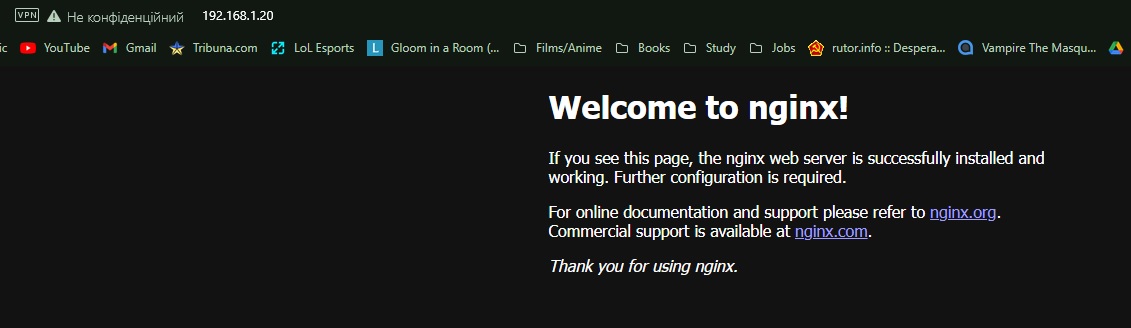


Рисунок 3.30 Перевірка доступу до сервісу nginx через балансувальник навантаження

Також доступність такого сервісу у локальній мережі було протестовано шляхом введення вищевказаного IP використовуючи смартфон. Результат перевірки наведений на рисунку 3.31.

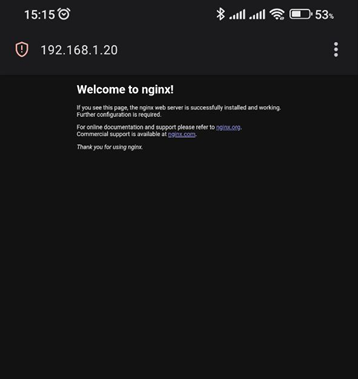


Рисунок 3.31 – Перевірка доступності сервісу використовуючи смартфон

Крім того, зробивши аналогічні кроки було створені сервіси із зовнішніми IP адресами для роботи раніше розгорнутої системи моніторингу і сповіщень.

Таким чином, адреси 192.168.1.21-192.168.1.23 використовуються для забезпечення доступу до метрик та графіків використаних кластером ресурсів. Результат виведення інформації про всі наявні в кластері сервіси наведений нижче на рисунку 3.32.

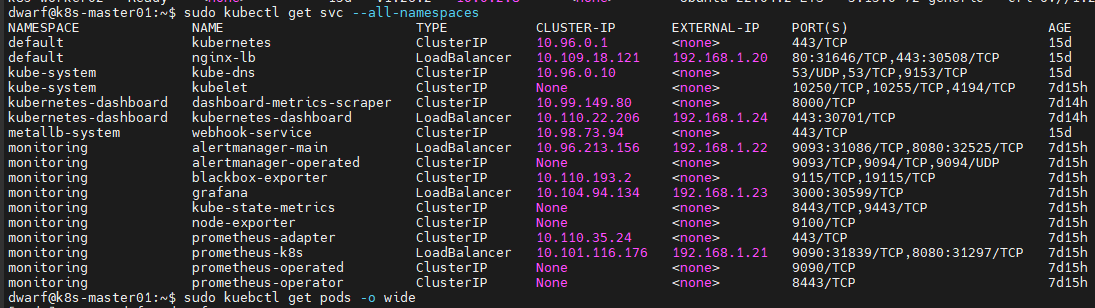


Рисунок 3.32 – Результат виведення інформації про всі сервіси кластера

Перевірка роботи сервісу Prometheus AlertManager, який виводить сповіщення про помилки у роботі наведена нижче на рисунку 3.33. Для доступу до цього сервісу необхідно ввести адресу 192.168.1.22 та порт 9093.

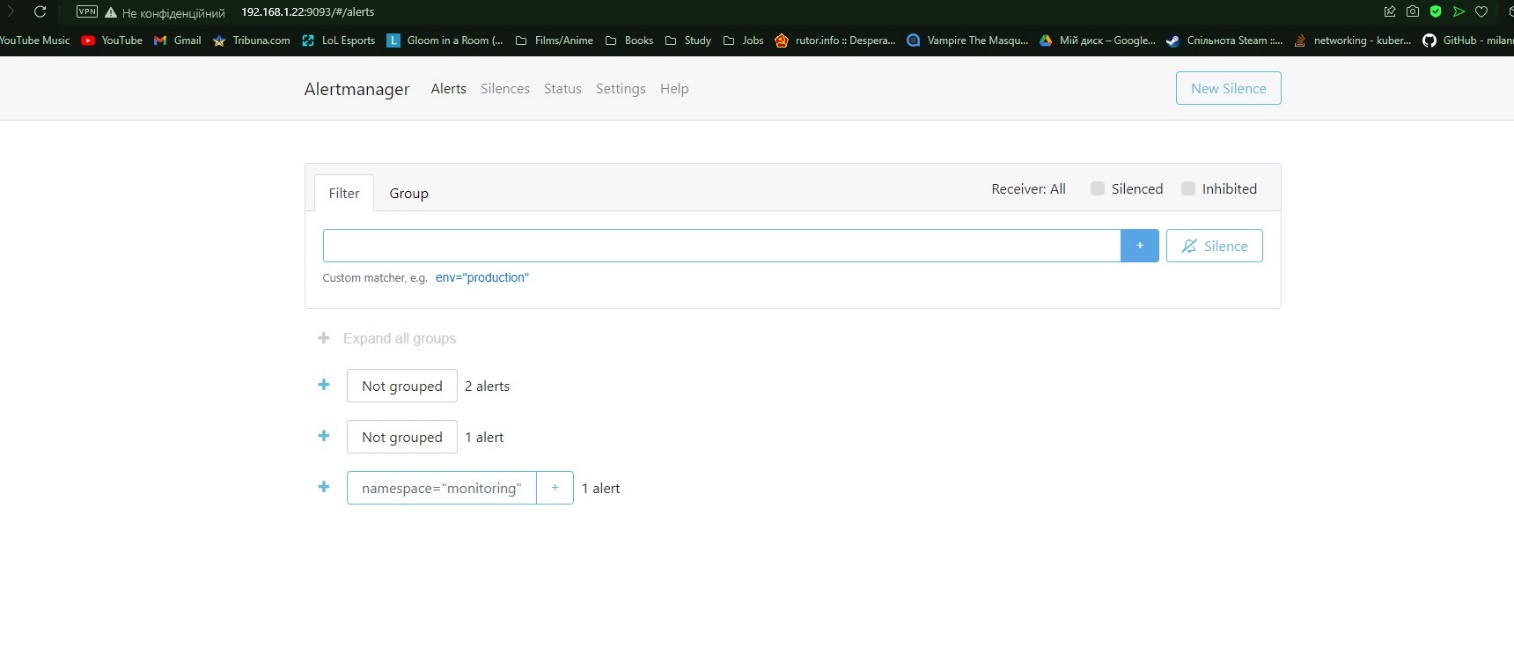


Рисунок 3.33 - Перевірка роботи сервісу сповіщень

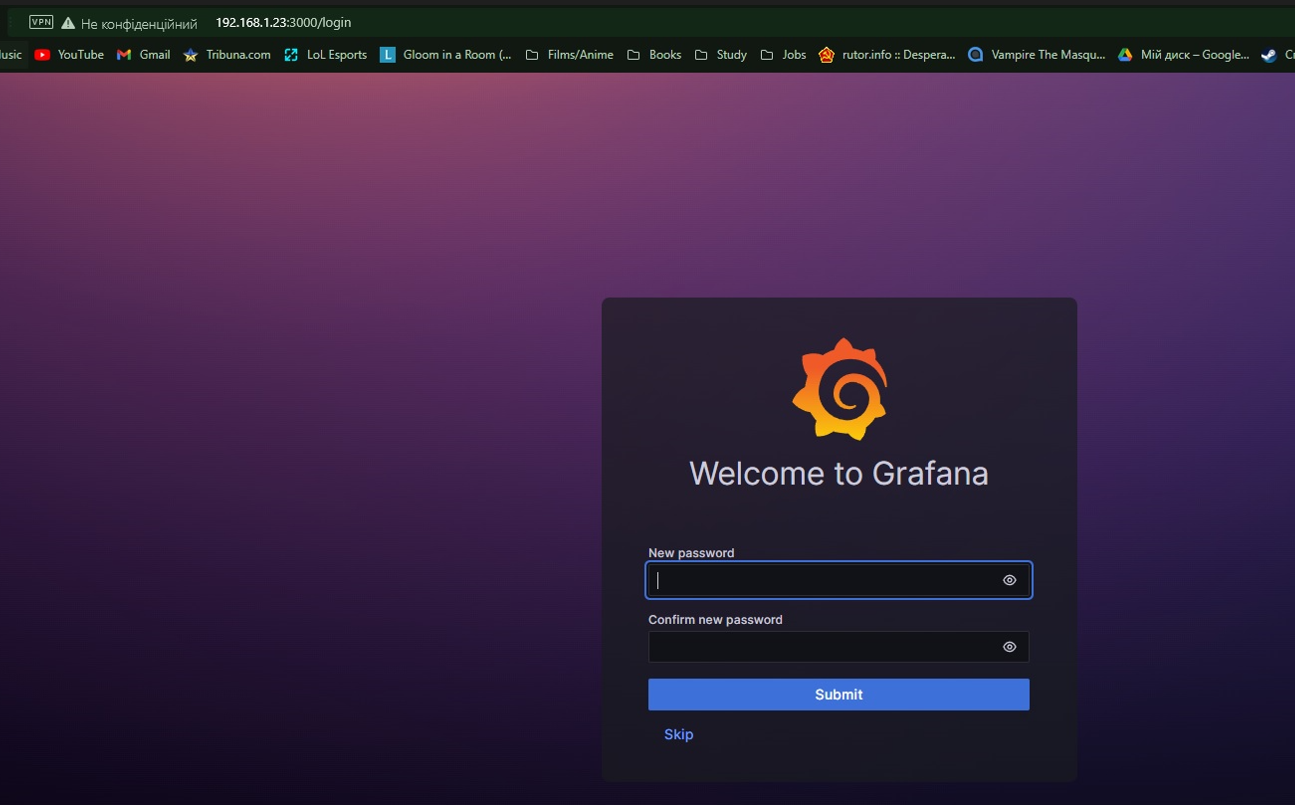
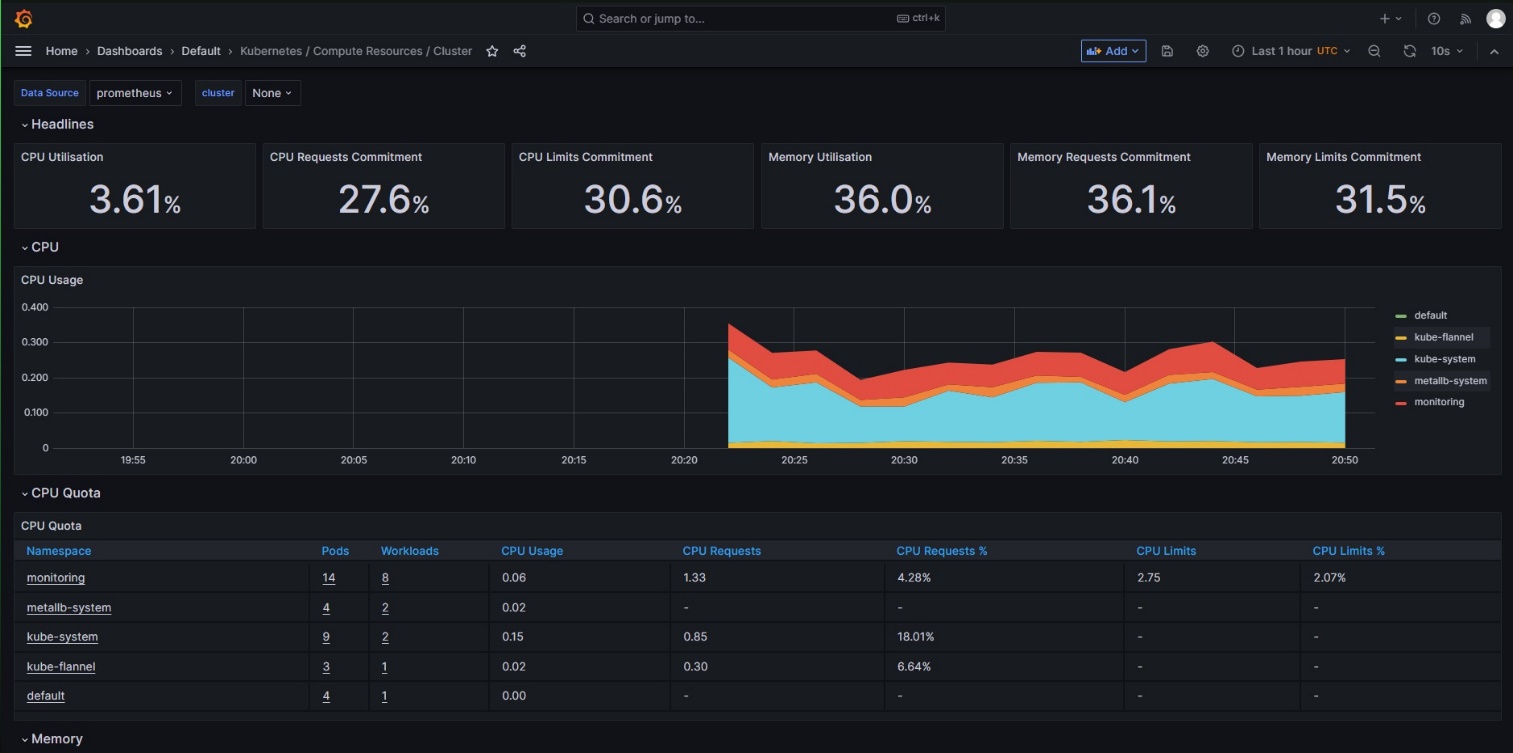
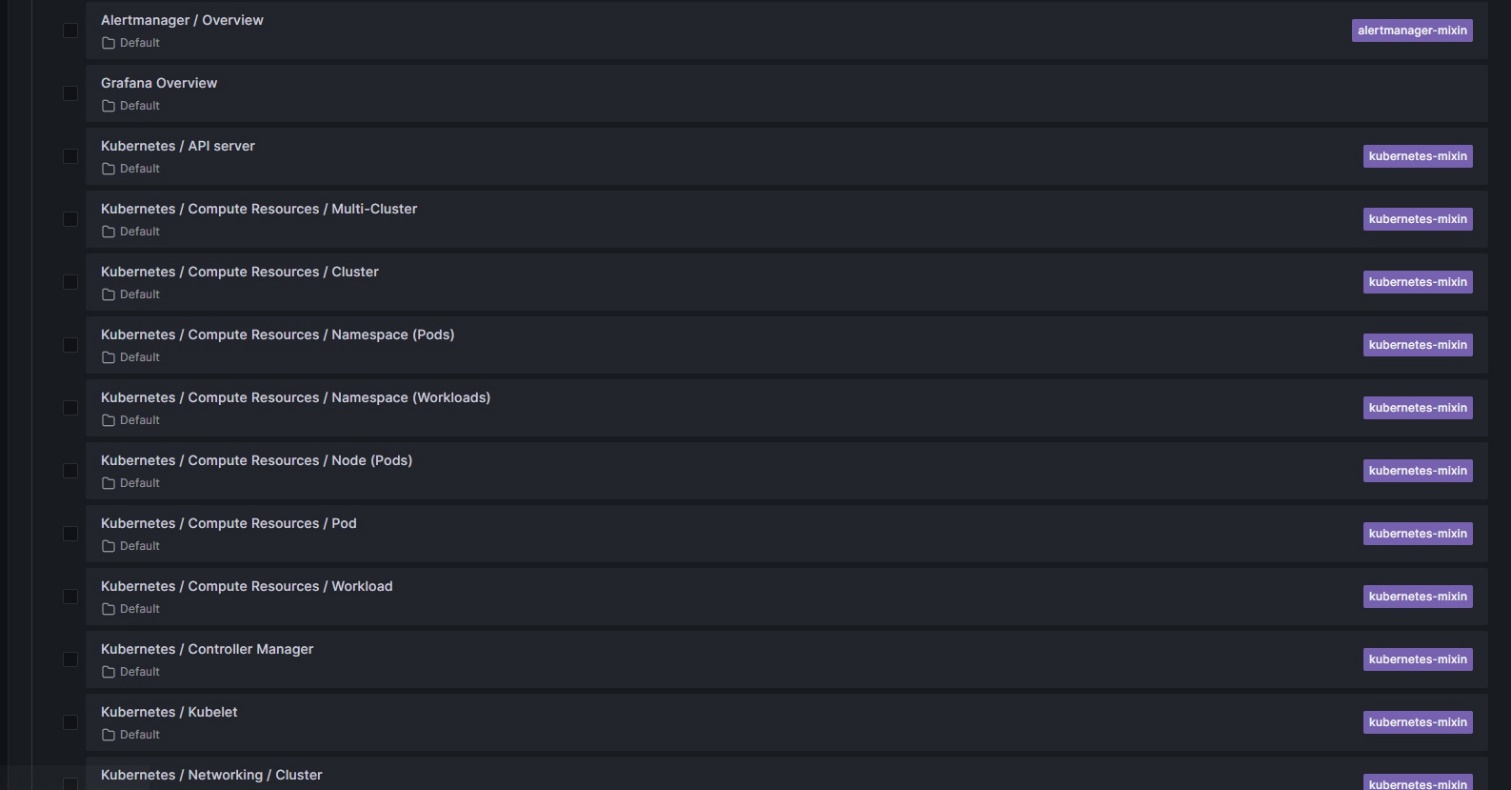
За адресою 192.168.1.23:3000 доступний сервіс Grafana, рисунок 3.34.

Рисунок 3.34 – Вікно входу до сервісу Grafana

Після авторизації, можна побачити інфографіки використаних кластером або ж по простору імен ресурсів, рисунок 3.35, а також перелік всіх процесів всередині кластера, рисунок 3.36.

Рисунок 3.35 – Інформація про витрату ресурсів системоюРисунок 3.36 – Список наявних процесів всередині кластера

За адресою 192.168.1.21:9090 доступний сервіс, який зберігає жунали (логи) всіх наявних процесів у кластері, рисунок 3.37.

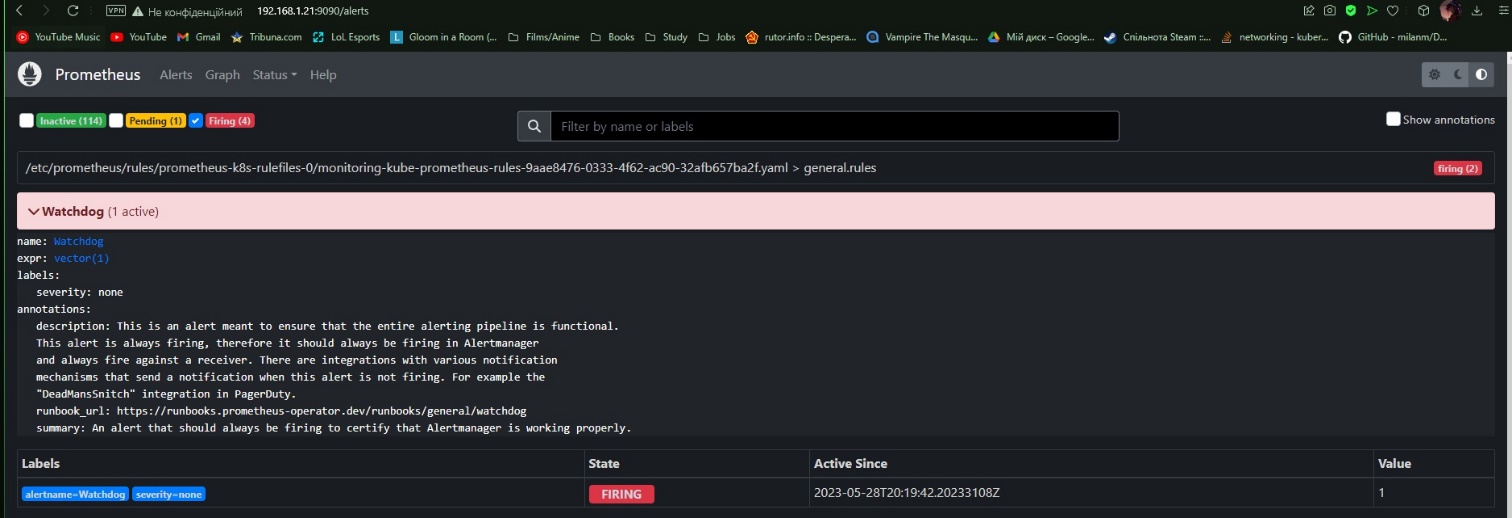


Рисунок 3.37 – Сервіс логування всіх процесів в кластері

## Висновки по третій частині

В цій частині була проведена робота з ініціалізації та налаштування кластера. Були встановлені всі необхідні сервіси для повноцінної роботи додатків телемедичного спрямування а також моніторингу стану всіх процесів всередині кластера. Крім того, було здійснено налаштування доступу сервісам та тестовому додатку до зовнішньої мережі шляхом налагодження проходження трафіку через балансувальник навантаження, який виділяє зовнішню IP адресу для всіх сервісів типу LoadBalancer.

# ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було проведено аналіз технології побудови розподіленої архітектури, за результатами якого було виокремлено види кластерних архітектур. Крім того, було проаналізовано відмінності кластерної архітектури від схожої за будовою архітектурою grid-систем. За результатами аналізу технологій контейнеризації та оркестрування встановлено, що ці технології, маючи велику популярність, пропонують широкий спектр інструментів для виконання поставлених задач, а також можуть використовуватися для підтримки роботи розподіленої інфраструктури.

Крім того було розглянуто архітектуру Kubernetes а також зроблений огляд існуючих сервісів для підтримки роботи кластера.

Згідно висунутим вимогам було розгорнута розподілена інфраструктури, яка забезпечує повноцінну роботу сервісів телемедичних моніторингових досліджень.

Підбиваючи підсумки можна визначити наступний перелік виконаних дій:

* створення та налаштування ВМ для роботи у кластері;
* створення внутрішньої мережі для вузлів кластера;
* поєднання всіх хостів у один кластер за допомогою ПЗ Kubernetes;
* встановлення КМІ для зв’язку Pod`ів із вузлами;
* додавання зовнішнього балансувальника трафіку;
* інсталяція об’єктного сховища даних MinIO
* налагодження системи моніторингу і сповіщень;
* налагодження CI/CD конвеєру за допомогою Helm Charts
* тестування роботи системи шляхом створення тестовго Deployment із образом nginx;

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке телемедицина? - [Електронний ресурс]. - Режим доступу: URL: https://ehealth.gov.ua/2021/11/15/shho-take-telemedytsyna/

2. Телемедицина в Україні - [Електронний ресурс]. - Режим доступу: URL: https://telemed24.ua/articles/telemedecina-v-ukraini

3. Владзимирський А. В., Стадник О. М., Карліньска М. Перше застосування телемедицини в Україні: Мар'ян Франке та Вітольд Ліпіньскі // Укр.ж.телемед.мед.телемат.- 2012.- Т.10,№ 1. - С.18-26.

4. Вишневський В.В., Романенко Т.М., Луговський Ю.О., Борецький О.Ф. Сервіси підготовки даних для автентифікації людини за її електрокардіограмою. Математичні машини і системи. 2022. № 3. С. 58–69.

5. Архітектура кластерів - [Електронний ресурс]. - Режим доступу: URL: https://server-shop.ua/ua/understanding-clustering-for-servers.html

6. Case studies: Sporotfy - [Електронний ресурс]. - Режим доступу: URL: https://kubernetes.io/case-studies/spotify/

7. Ком’ютерний кластер - [Електронний ресурс]. - Режим доступу: URL: https://uk.wikipedia.org/wiki/Комп%27ютерний\_кластер

8. Кластер серверів - [Електронний ресурс]. - Режим доступу: URL: https://uh.ua/ua/clusters/klaster-serverov.html

9. Документація з Kubernetes. Kubernetes.io. - [Електронний ресурс]. - Режим доступу: URL: https://kubernetes.io/docs/home/

10. Kubernetes cluster architecture - [Електронний ресурс]. - Режим доступу: URL:https://www.vmware.com/topics/glossary/content/kubernetes-cluster.html

11. Кластерна архітектура - [Електронний ресурс]. - Режим доступу: URL: https://studfile.net/preview/6368040/page:4/

12. What is containerization - [Електронний ресурс]. - Режим доступу: URL: https://aws.amazon.com/what-is/containerization/

13. Контейнеризація ресурсів - [Електронний ресурс]. - Режим доступу: URL: https://www.ibm.com/topics/containerization

14. AWS EC2 - [Електронний ресурс]. - Режим доступу: URL: https://aws.amazon.com/ru/ec2/

15. Контейнеризована архітектура - [Електронний ресурс]. - Режим доступу: URL: https://www.aquasec.com/cloud-native-academy/container-security/containerized-architecture/

16. Статистика використання контейнерів і оркестрування - [Електронний ресурс]. - Режим доступу: URL: https://www.statista.com/topics/8299/container-technology/#topicOverview

17. Принципи оркестрування контейнерів - [Електронний ресурс]. - Режим доступу: URL: https://devopscube.com/docker-container-clustering-tools/

18. Оркестрування - [Електронний ресурс]. - Режим доступу: URL: https://avinetworks.com/glossary/container-orchestration/

19. ПЗ Docker Swarm - [Електронний ресурс]. - Режим доступу: URL: https://docs.docker.com/engine/swarm/

20. Плюси та мінус Docker Swarm - [Електронний ресурс]. - Режим доступу: URL: https://circleci.com/blog/docker-swarm-vs-kubernetes/

21. REST-запити - [Електронний ресурс]. - Режим доступу: URL: https://uk.wikipedia.org/wiki/REST

22. Контейнерні двигуни - [Електронний ресурс]. - Режим доступу: URL: https://habr.com/ru/companies/domclick/articles/566224/

23. CNI - [Електронний ресурс]. - Режим доступу: URL: https://habr.com/ru/companies/flant/articles/329830/https://habr.com/ru/companies/flant/articles/329830/

24. Принципи роботи протоколу BGP - [Електронний ресурс]. - Режим доступу: URL: https://habr.com/ru/articles/450814/

25. Команди налаштування файрволу - [Електронний ресурс]. - Режим доступу: URL: https://www.baeldung.com/linux/uncomplicated-firewall

26. Принцип CI/CD в кластері https://komodor.com/blog/ci-cd-pipelines-for-kubernetes-best-practices-and-tools/

# ДОДАТКИ

# ДОДАТОК А

# A.1 ФАЙЛИ, ЗАДІЯНІ ПІД ЧАС РОЗГОРТАННЯ ТА НАЛАШТУВАННЯ КЛАСТЕРА ДЛЯ ТЕЛЕМЕДИЧНИХ ЗАДАЧ

A.1.1 Файл ports-master.sh для налаштування портів вузла-мастера

#!/usr/bin/env bash

#Dmytro Pomyluiko. KI-191. File for conf ports in master node

ufw default deny incoming

ufw default allow outgoing

ufw allow ssh

ufw allow http

ufw allow https

ufw allow 6443/tcp

ufw allow 2379:2380/tcp

ufw allow 10250/tcp

ufw allow 10257/tcp

ufw allow 10259/tcp

A.1.2 Файл ports-worker.sh для налаштування портів робочого вузла

#!/usr/bin/env bash

#Dmytro Pomyluiko. KI-191. File for conf ports in worker node

ufw default deny incoming

ufw default allow outgoing

ufw allow ssh

ufw allow http

ufw allow https

ufw allow 10250/tcp

ufw enable

A.1.3 Файл inst-docker.sh для налаштування портів робочого вузла

#!/usr/bin/env bash

#Dmytro Pomyluiko. KI-191. File for installation docker for cluster

sudo apt-get update

sudo apt-get install -y \

apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

sudo apt-get update

sudo apt-get install -y docker-ce docker-ce-cli containerd.io

A.1.4 Файл iptables.sh для налаштування портів робочого вузла

#!/usr/bin/env bash

#Dmytro Pomyluiko, preparation for installing CRI-O

cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf

overlay

br\_netfilter

EOF

sudo modprobe overlay

sudo modprobe br\_netfilter

# sysctl params required by setup

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-iptables = 1

net.bridge.bridge-nf-call-ip6tables = 1

net.ipv4.ip\_forward = 1

EOF

# Apply

sudo sysctl --system

A.1.5 Файл inst-CRI-O.sh для встановлення

#!/usr/bin/env bash

#Dmytro Pomyluiko. Installation CRI-O runtime for Kubernetes cluster

sudo -i

OS="xUbuntu\_22.04"

VERSION=1.26

echo "deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/ /" > /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list

echo "deb http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/$VERSION/$OS/ /" > /etc/apt/sources.list.d/devel:kubic:libcontainers:stable:cri-o:$VERSION.list

curl -L https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable:cri-o:$VERSION/$OS/Release.key | apt-key add -

curl -L https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/Release.key | apt-key add -

sudo apt update

sudo apt install cri-o cri-o-runc*Sta*

sudo systemctl daemon-reload

sudo systemctl restart crio

sudo systemctl enable crio

systemctl status crio

A.1.6 Файл kubeadm-init.sh для ініціалізації кластера

#!/usr/bin/env bash

#Dmytro Poyluiko, initiazlizing cluster for diploma project

sudo kubeadm init \

--apiserver-advertise-address=10.0.2.6 \

--cri-socket /var/run/crio/crio.sock \

--upload-certs \

--control-plane-endpoint=10.0.2.6:6443 \

--pod-network-cidr=10.244.0.0/16 | sudo tee cluster-init.txt

A.1.7 Файл kubeadm-join.sh для приєднання робочих вузлів у кластер

#!/usr/bin/env bash

#Dmytro Pomyluiko. KI-191. Joining new worker nodes to the initialized cluster

sudo kubeadm join 10.0.2.6:6443 --token itrxbs.n8elwialgc46jek9 \

--discovery-token-ca-cert-hash sha256:f64bd3364c1038efa0cae286e5c3f48136218fc90834c1c8225075698820e619 \

--cri-socket /var/run/crio/crio.sock

A.1.8 Файл start-interract.sh для увімкнення запитів у кластер

#!/usr/bin/env bash

#Dmytro Pomyluiko. KI-191. Bash file for enable the requests to the cluster

sudo mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

A.1.9 Файл inst-helm.sh для встановлення пакетного менеджеру Helm

#!/usr/bin/env bash

sudo curl -fsSL -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

sudo chmod 700 get\_helm.sh

sudo ./get\_helm.sh

A.1.10 Файл pod-adm.sh для налаштування правил Pod`ів для успішного встановлення MetalLB

#!/usr/bin/env bash

#Dmytro Pomyluiko. KI-191. File for editing Pod security for MetalLB

sudo kubectl label --overwrite ns default \

pod-security.kubernetes.io/enforce=privileged \

pod-security.kubernetes.io/enforce-version=v1.27 \

pod-security.kubernetes.io/audit=privileged \

pod-security.kubernetes.io/audit-version=v1.27 \

pod-security.kubernetes.io/warn=privileged \

pod-security.kubernetes.io/warn-version=v1.27

A.1.10 Файл helm-metallb.sh для встановлення MetalLB

#!/usr/bin/env bash

#Dmytro Pomyluiko. KI-191. File for installing MetalLB via Helm

helm repo add metallb https://metallb.github.io/metallb

helm install metallb metallb/metallb

A.1.11 Файл mlb-ip.yaml для встановлення переліку IP, адрес, які може розподіляти MetalLB

apiVersion: metallb.io/v1beta1

kind: IPAddressPool

metadata:

name: mlb-pool

namespace: metallb-system

spec:

addresses:

- 192.168.1.20-192.168.1.30

---

apiVersion: metallb.io/v1beta1

kind: L2Advertisement

metadata:

name: mlb-pool

namespace: metallb-system

spec:

ipAddressPools:

- mlb-pool

A.1.10 Конфігураційний файл kube-flannel.yaml для встановлення CNI Flannel

---

kind: Namespace

apiVersion: v1

metadata:

name: kube-flannel

labels:

k8s-app: flannel

pod-security.kubernetes.io/enforce: privileged

---

kind: ClusterRole

apiVersion: rbac.authorization.k8s.io/v1

metadata:

labels:

k8s-app: flannel

name: flannel

rules:

- apiGroups:

- ""

resources:

- pods

verbs:

- get

- apiGroups:

- ""

resources:

- nodes

verbs:

- get

- list

- watch

- apiGroups:

- ""

resources:

- nodes/status

verbs:

- patch

- apiGroups:

- networking.k8s.io

resources:

- clustercidrs

verbs:

- list

- watch

---

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

labels:

k8s-app: flannel

name: flannel

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: flannel

subjects:

- kind: ServiceAccount

name: flannel

namespace: kube-flannel

---

apiVersion: v1

kind: ServiceAccount

metadata:

labels:

k8s-app: flannel

name: flannel

namespace: kube-flannel

---

kind: ConfigMap

apiVersion: v1

metadata:

name: kube-flannel-cfg

namespace: kube-flannel

labels:

tier: node

k8s-app: flannel

app: flannel

data:

cni-conf.json: |

{

"name": "cbr0",

"cniVersion": "0.3.1",

"plugins": [

{

"type": "flannel",

"delegate": {

"hairpinMode": true,

"isDefaultGateway": true

}

},

{

"type": "portmap",

"capabilities": {

"portMappings": true

}

}

]

}

net-conf.json: |

{

"Network": "10.10.0.0/16",

"Backend": {

"Type": "vxlan"

}

}

---

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: kube-flannel-ds

namespace: kube-flannel

labels:

tier: node

app: flannel

k8s-app: flannel

spec:

selector:

matchLabels:

app: flannel

template:

metadata:

labels:

tier: node

app: flannel

spec:

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.io/os

operator: In

values:

- linux

hostNetwork: true

priorityClassName: system-node-critical

tolerations:

- operator: Exists

effect: NoSchedule

serviceAccountName: flannel

initContainers:

- name: install-cni-plugin

image: docker.io/flannel/flannel-cni-plugin:v1.1.2

#image: docker.io/rancher/mirrored-flannelcni-flannel-cni-plugin:v1.1.2

command:

- cp

args:

- -f

- /flannel

- /opt/cni/bin/flannel

volumeMounts:

- name: cni-plugin

mountPath: /opt/cni/bin

- name: install-cni

image: docker.io/flannel/flannel:v0.21.4

#image: docker.io/rancher/mirrored-flannelcni-flannel:v0.21.4

command:

- cp

args:

- -f

- /etc/kube-flannel/cni-conf.json

- /etc/cni/net.d/10-flannel.conflist

volumeMounts:

- name: cni

mountPath: /etc/cni/net.d

- name: flannel-cfg

mountPath: /etc/kube-flannel/

containers:

- name: kube-flannel

image: docker.io/flannel/flannel:v0.21.4

#image: docker.io/rancher/mirrored-flannelcni-flannel:v0.21.4

command:

- /opt/bin/flanneld

args:

- --ip-masq

- --kube-subnet-mgr

resources:

requests:

cpu: "100m"

memory: "50Mi"

securityContext:

privileged: false

capabilities:

add: ["NET\_ADMIN", "NET\_RAW"]

env:

- name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name

- name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace

- name: EVENT\_QUEUE\_DEPTH

value: "5000"

volumeMounts:

- name: run

mountPath: /run/flannel

- name: flannel-cfg

mountPath: /etc/kube-flannel/

- name: xtables-lock

mountPath: /run/xtables.lock

volumes:

- name: run

hostPath:

path: /run/flannel

- name: cni-plugin

hostPath:

path: /opt/cni/bin

- name: cni

hostPath:

path: /etc/cni/net.d

- name: flannel-cfg

configMap:

name: kube-flannel-cfg

- name: xtables-lock

hostPath:

path: /run/xtables.lock

type: FileOrCreate

A.1.13 Файл depl-nginx.yaml для розгортання Deployment nginx

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx

labels:

app: nginx

spec:

replicas: 4

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

A.1.13 Конфігураційний файл nginx-lb.yaml для розгортання сервісу типу LoadBalancer для роботи Deployment nginx

apiVersion: v1

kind: Service

metadata:

name: nginx-lb

labels:

app: nginx

spec:

externalTrafficPolicy: Cluster

type: LoadBalancer

selector:

app: nginx

ports:

- name: http

port: 80

targetPort: 80

protocol: TCP

- name: https

port: 443

targetPort: 443

protocol: TCP