Міністерство освіти і науки України
Одеський національний технологічний університет
Кафедра комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему *Дослідження стану застосувань у системах контейнеризації*
(назва кваліфікаційної роботи згідно наказу ОНТУ)

Здобувача *Редькіна О.О.*
(прізвище, ініціали)

2 курсу *561а* групи

Керівники: *ст. викл. Сіренко О. І.*
(посада, прізвище та ініціали)
д. т. н., проф. Артеменко С. В.
(посада, прізвище та ініціали)

Консультанти: *д.е.н., проф. Басюркіна Н.Й.*
(посада, прізвище та ініціали)

(посада, прізвище та ініціали)

Кваліфікаційна робота допускається до захисту

Рішення кафедри від *30.11* 2023 р., протокол № *3*

Завідувач кафедри комп. інженерії _____ Сергій АРТЕМЕНКО
(назва кафедри) (підпис) (Ім'я ПРІЗВИЩЕ)

Одеса – 2023 рік

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерної інженерії, програмування та кіберзахисту*
Кафедра *комп'ютерної інженерії*
Ступінь вищої освіти *магістр*
Спеціальність *123 «Комп'ютерна інженерія»*
Освітня програма *Комп'ютерні системи та мережі*

ЗАТВЕРДЖУЮ
Зав. кафедри *комп'ютерної інженерії*
Сергій АРТЕМЕНКО
« *30* » листопада 2022 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Редькіна Олександра Олексійовича

1. Тема роботи *Дослідження стану застосувань у системах контейнеризації*

Затверджена наказом університету від « *30* » *листопада* 2022 р., наказ № *884-03*

2 Термін здачі здобувачем закінченої роботи *28 листопада 2023 р.*

3. Вихідні дані роботи

1. Концепція 2. Встановлення Docker 3. Встановлення Prometheus 4. Встановлення Grafana 5. Встановлення KPI 6. Налаштування панелей систем моніторингу 7. Аналіз результату

4. Перелік питань, які потрібно розробити

1. Вступ. 2. Теоретичні основи систем контейнеризації. *3. Проектування системи. 4. Розробка системи. 5. Загальні висновки. 6. Економічні розрахунки. 7. Охорона праці.*

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

Слайд 1. Титульний слайд. Слайд 2. Вступ. Слайд 3. Основи контейнеризації.
Слайд 4. Docker як приклад контейнеризації. Слайд 5. Архітектура системи моніторингу .
Слайд 6. Системи моніторингу Prometheus. Слайд 7. Системи моніторингу Grafana.
Слайд 8. Готові панелі із Дашбордів Графани. Слайд 9. Реалізація та тестування.
Слайд 10. Економіка проекту Слайд 11. Висновки та перспективи. Слайд 12. Подяки

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Економіка	*д.е.н., проф. Басюркіна Н.Й.*		
Охорона праці	*д. т. н., проф.Артеменко С. В.*		
Нормоконтроль	*ст. викл. Сіренко О.І.*		

7. Дата видачі завдання *30.11.2022*

Керівники *Олександр СІРЕНКО*

Сергій АРТЕМЕНКО

Завдання прийняв до виконання *Олександр РЕДЬКІН*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Дослідженні предметної області*	*16.01.2023*	
2.	*Дослідження існуючих аналогів*	*30.01.2023*	
3.	*Дослідження основ контейнеризації*	*24.02.2023*	
4.	*Вивчення Docker, Prometheus, Grafana*	*12.02.2023*	
5.	*Проектування*	*17.08.2023*	
6.	*Розробка демонстраційної версії ПЗ*	*25.10.2023*	
7.	*Підготовка техніко-економічної частини*	*14.11.2023*	
8.	*Підготовка розділу охорони праці*	*16.11.2023*	
9.	*Оформлення пояснювальної записки*	*22.11.2023*	
10.	*Оформлення графічної частини та лістингу*	*27.11.2023*	

Здобувач-дипломник _____ *Олександр РЕДЬКІН*

Керівники роботи _____ *Олександр СІРЕНКО*

_____ *Сергій АРТЕМЕНКО*

Несу відповідальність за ідентичність електронного та друкованого варіантів кваліфікаційної роботи, даю згоду на обробку персональних даних та не заперечую проти розміщення кваліфікаційної роботи на офіційних web-ресурсах ОНТУ.

Підтверджую, що в кваліфікаційній роботі відсутні порушення норм академічної доброчесності.

Здобувач-дипломник *Олександр РЕДЬКІН* _____

АНОТАЦІЯ

Ця дипломна робота присвячена дослідженню стану застосувань у системах контейнеризації, моніторингу систем контейнеризації, з особливим акцентом на Docker та використання *Prometheus* і *Grafana* для моніторингу ресурсів. Робота складається з п'яти основних розділів, які охоплюють теоретичні основи контейнеризації, аналіз існуючих систем, а також практичну реалізацію системи моніторингу.

У першому розділу роботи подано теоретичні основи моніторингу систем контейнеризації. Описано концепції та принципи контейнеризації, її історичний розвиток та сучасний стан. Особливо увага приділяється аналізу систем контейнеризації, з детальним поглядом на *Docker* та *Kubernetes*.

Другий розділ присвячений проектуванню системи моніторингу. У цьому розділі розглядаються вимоги до системи моніторингу, вибір архітектурного стилю, а також обзору існуючих рішень з моніторингу.

Третій розділ зосереджений на розробці системи моніторингу, включаючи її тестування та оптимізацію. Четвертий розділ включає економічний аналіз проекту, а п'ятий розділ присвячений питанням охорони праці.

Результатом роботи є розгорнута на локальному комп'ютері система моніторингу *KPI* метрик контейнерів та відображення параметрів у зручному для нас вигляді, з можливістю зберігання показань у БД для їх подальшого аналізу.

ключові слова: контейнеризація, система моніторингу, моніторинг ресурсів, Docker, Kubernetes, Prometheus, Grafana.

ABSTRACT

This thesis is dedicated to the study of application states in containerization systems and the monitoring of containerization systems, with a special focus on Docker and the use of Prometheus and Grafana for resource monitoring. The work consists of five main sections, covering the theoretical foundations of containerization, analysis of existing systems, and the practical implementation of a monitoring system.

In the first section, the theoretical foundations of monitoring containerization systems are presented. The concepts and principles of containerization, its historical development, and current state are described. Particular attention is paid to the analysis of containerization systems, with a detailed overview of Docker and Kubernetes.

The second section is devoted to the design of the monitoring system. This section discusses the requirements for the monitoring system, the choice of architectural style, and an overview of existing monitoring solutions.

The third section is focused on the development of the monitoring system, including its testing and optimization. The fourth section includes an economic analysis of the project, while the fifth section is devoted to occupational safety issues.

The result of the work is a monitoring system for KPI metrics of containers deployed on a local computer, displaying parameters in a convenient format for us, with the ability to store readings in a database for further analysis.

***Keywords**: containerization, monitoring system, resource monitoring, Docker, Kubernetes, Prometheus, Grafana.*

ЗМІСТ

ВСТУП

У сучасному світі комп’ютерні системи стають дедалі складнішими та масштабнішими. Це пов’язано зі зростанням вимог до продуктивності, надійності та безпеки. Розробка та розгортання таких систем вимагає нових підходів, які дозволять удосконалити ці характеристики.

Контейнеризація та мікросервісна архітектура є двома такими підходами, які набирають все більшої популярності.. Технологія контейнеризації , що забезпечує ізоляцію процесів та сервісів в контейнерах, та технології мікросервісної архітектури радикально змінила підходи до розгортання та масштабування додатків, пропонуючи небачені раніше можливості для ефективності та гнучкості.

Зазначимо мету, об’єкт, предмет та задачі, які необхідно вирішити в ході кваліфікаційної роботи.

Мета роботи: Розробка та аналіз системи моніторингу для контейнеризованих додатків з використанням інструментів *Docker*, *Prometheus* та *Grafana*.

Об'єкт дослідження: Системи моніторингу в контексті контейнеризації.

Предмет дослідження: Методи та інструменти моніторингу ресурсів контейнеризованих додатків.

Задачі роботи:

1. Дослідити теоретичні основи та поточний стан технологій контейнеризації.
2. Аналізувати існуючі підходи до моніторингу в контейнеризованих середовищах.
3. Розглянути особливості використання *Docker* для створення та управління контейнерами.
4. Дослідити можливості *Prometheus* та *Grafana* як інструментів для моніторингу та візуалізації даних.

5. Розробити прототип системи моніторингу, що інтегрує *Docker*, *Prometheus*, та *Grafana*.
6. Протестувати та оцінити ефективність запропонованої системи.
7. Зробити висновки щодо можливостей оптимізації та масштабування системи моніторингу.

Наукова новизна кваліфікаційної роботи полягає в удосконаленні існуючих методів розробки та розгортання систем моніторингу, а також в оптимізації використаних ресурсів.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ СИСТЕМ КОНТЕЙНЕРІЗАЦІЇ ТА МОНІТОРИНГУ

1.1 Основи контейнеризації

Контейнеризація - це сучасний підхід у сфері розробки та розгортання програмного забезпечення, який забезпечує легкість, швидкість та ефективність у доставці додатків. Основна ідея контейнеризації полягає у відокремленні додатку та його залежностей від основної операційної системи, що дозволяє легко переносити додатки між різними середовищами (розробка, тестування, виробництво) без потреби переконфігурації або модифікації. На (рис.1.1) представлено схему влаштування контейнерів.

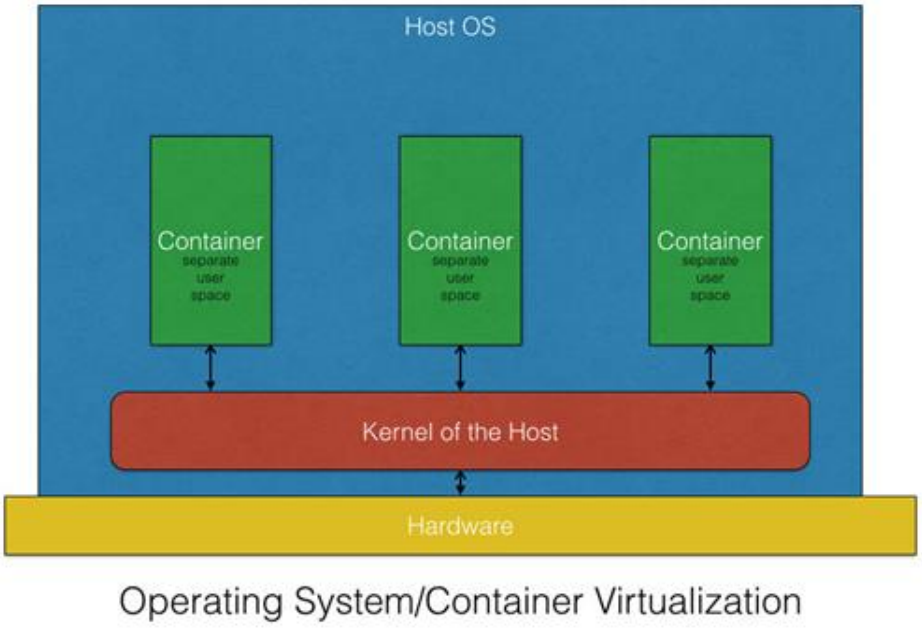

Рис 1.1 - Схема влаштування контейнерів

Апаратне забезпечення (Hardware) - це фізичні компоненти комп'ютерної системи, на яких виконується операційна система та всі програми.

Ядро хоста (*Kernel of the Host*) - це основна частина операційної системи, яка управляє апаратним забезпеченням та розподіляє ресурси між програмами. Ядро забезпечує низькорівневі системні сервіси, необхідні для виконання програмного забезпечення.

Операційна система хоста (*Host OS*) - це комплексне програмне забезпечення, що включає в себе ядро, системні бібліотеки, утиліти та інші основні компоненти, які разом забезпечують інтерфейс між апаратним забезпеченням та кінцевим користувачем або додатками.

Контейнер (*Container*) - це ізольоване середовище виконання для додатків, яке включає додаток та його залежності. Всі контейнери на хості ділять ядро операційної системи хоста, але кожен з них має власний окремий простір користувача (*Separate User Space*), що забезпечує ізоляцію та безпеку.

Операційна система/Віртуалізація контейнерів (*Operating System/Container Virtualization*) - це загальна назва для технологій, що дозволяють віртуалізацію за допомогою контейнерів на рівні операційної системи. Відмінною особливістю є те, що вона дозволяє ефективно використовувати системні ресурси, так як контейнери спільно використовують ядро хоста та менш вимогливі до ресурсів порівняно з традиційною віртуалізацією на основі віртуальних машин.

Ключові аспекти концепції контейнеризації включають:

1. Ізоляція та незалежність. Контейнери ізолюють додатки та їх залежності від зовнішнього середовища, забезпечуючи незалежність від інфраструктури.
2. Легкість та портативність. Використання контейнерів значно спрощує процес розгортання додатків, оскільки вони містять усе необхідне для їхнього функціонування.
3. Ефективність ресурсів. Контейнери використовують менше ресурсів порівняно з традиційними віртуальними машинами, оскільки вони не потребують окремої операційної системи.
4. Однорідність виконавчого середовища. Незалежно від того, де запускається контейнер, він працюватиме однаково, що забезпечує однорідність між різними середовищами.
5. Швидкість створення та розгортання. Контейнери можуть бути створені та запущені за лічені секунди, що є ключовим фактором для неперервної інтеграції та розгортання (*CI/CD*).

Початки контейнеризації можна відслідкувати до введення механізму chroot у Unix в 1979 році, який вперше дозволив ізолювати файлові системи. Але саме концепція, схожа на сучасні контейнери, з'явилася з появою *FreeBSD Jails* у 2000 році та *Linux VServer* у 2001 році. Далі були *Solaris Containers* та *LXC* (*Linux Containers*), які продовжили розвиток ізоляції та надали основу для того, що ми зараз знаємо як контейнери.

Але саме великий стрибок вперед стався з появою *Docker* у 2013 році. *Docker* зробив процес створення, розгортання та управління контейнерами значно простішим і зручнішим, ніж це було раніше, завдяки своєму легкому та ефективному рішенню.

Docker дозволив розробникам "упаковувати" свої додатки разом із всіма необхідними залежностями в стандартизовані блоки, які можна легко переміщувати між різними середовищами, від локальних машин до виробничих серверів. Це вирішило багато головних болів, пов'язаних з розробкою програмного забезпечення, таких як проблеми "воно працює у мене на комп'ютері".

Сьогодні контейнеризація є ключовою складовою багатьох *DevOps* практик та хмарних платформ. *Kubernetes*, що був створений *Google* і став open source, є системою оркестрації контейнерів, яка дозволяє автоматично розгортати, масштабувати та управляти контейнеризованими додатками. Ця екосистема надає неймовірну гнучкість та ефективність, що є невід'ємним елементом сучасної розробки програмного забезпечення.

Завдяки цим технологіям, ми маємо можливість швидко реагувати на зміни вимог, оптимізувати використання ресурсів та забезпечити неперервну доставку та інтеграцію наших додатків. *Docker* і *Kubernetes* разом відкрили нову еру у світі програмування, де розробники можуть зосередитись на коді, не переймаючись нюансами інфраструктури.

1.2 Аналіз систем контейнеризації

Docker, з'явившись на *IT*-арені у 2013 році, вніс революційні зміни в спосіб, яким розробники та системні адміністратори працюють із програмним забезпеченням. Ця технологія не тільки спростила процес розгортання та управління додатками, а й пропонувала новий, більш ефективний спосіб ізоляції та доставки додатків.

Контейнер - це стандартна одиниця програмного забезпечення, яка упаковує код і всі його залежності, щоб застосунок міг швидко і надійно працювати з одного обчислювального середовища в інше. Образ контейнера *Docker* - це легкий автономний виконуваний пакет програмного забезпечення, який містить усе необхідне для запуску програми: код, середовище виконання, системні інструменти, системні бібліотеки та налаштування.

Образи контейнерів стають контейнерами під час виконання, а у випадку контейнерів *Docker* - образи стають контейнерами, коли їх запускають на *Docker Engine* .

Docker Engine підтримує мільйони застосунків по всьому світу, надаючи стандартизований формат пакування для різних застосунків.

Docker Engine поклав початок руху за контейнеризацію

Docker Engine - це де-факто галузеве середовище виконання контейнерів, що працює в різних операційних системах *Linux (CentOS, Debian, Fedora, RHEL та Ubuntu) і Windows Server* . *Docker* створює прості інструменти та універсальний підхід до пакування, який об'єднує всі залежності програми всередині контейнера, який потім запускається на *Docker Engine*. *Docker Engine* дає змогу контейнерним додаткам працювати в будь-якому місці та в будь-якій інфраструктурі, усуваючи "пекло залежностей" для розробників і операторів, а також усуваючи фразу "це працює на моєму ноутбуці!" проблему.

Розглянемо основні переваги використання *Docker*:

1. За допомогою *Docker* розробники можуть легко "упакувати" додаток та його залежності в один контейнер, який може бути запущений на будь-якій

системі, що підтримує *Docker*. Це забезпечує ізольованість середовища від локальної машини розробника до продакшн-сервера, вирішуючи проблему "воно працює у мене на комп'ютері".

2. Стандарт: *Docker* створив галузевий стандарт для контейнерів, тому їх можна переносити куди завгодно. *Docker* дозволяє запускати додатки в ізольованих контейнерах, які можна легко розгорнути на будь-якій машині, де встановлено *Docker*. Це спрощує процес розгортання додатків у різних середовищах, таких як локальні машини, хмара або гібридні середовища. *Docker*-контейнери всюди: *Linux*, *Windows*, центри обробки даних, хмари, безсерверні системи тощо (рис.1.2).

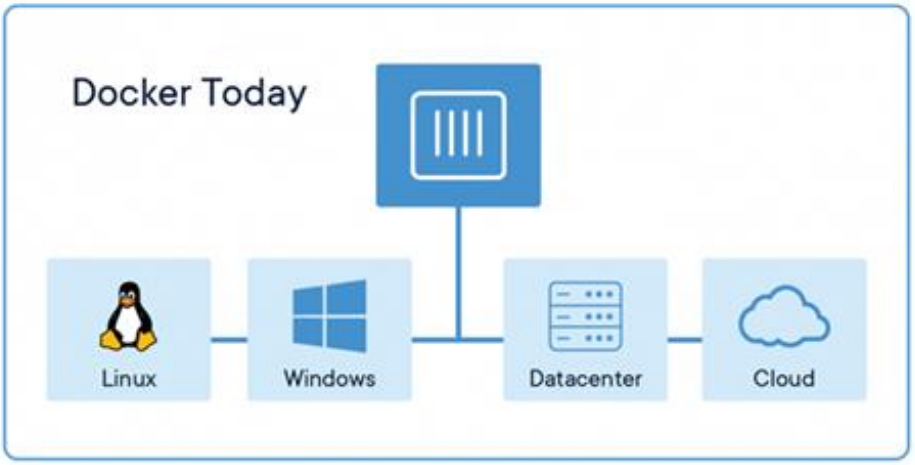

Рис.1.2 - Docker Today

3. Покращує масштабованість. *Docker* дозволяє масштабувати додатки, просто додаючи або видаляючи контейнери. Це робить додатки більш масштабованими та адаптивними до змінних потреб.

4. Легкість: контейнери використовують ядро ОС комп'ютера і, отже, не потребують ОС для кожного додатка, що підвищує ефективність сервера і знижує витрати на сервер і ліцензування.

5. Безпека: додатки безпечніші в контейнерах, а *Docker* забезпечує найнадійніші в галузі можливості ізоляції за замовчуванням.

6. Ефективність використання ресурсів. *Docker* використовує лише ресурси, необхідні для роботи додатку. Це допомагає заощадити ресурси та знизити витрати.

Переваги використання *Docker* для розробників:

1. Дозволяє розробникам зосередитися на розробці, а не на налаштуванні інфраструктури.

2. Полегшує тестування та розгортання додатків.

3. Допомагає розробникам створювати більш надійні та безпечні додатки.

Переваги використання *Docker* для системних адміністраторів:

1. Дозволяє легко керувати додатками та інфраструктурою.

2. Полегшує масштабування додатків.

3. Допомагає системним адміністраторам створювати більш безпечні та надійні системи.

Однією з ключових особливостей *Docker* є його *Dockerfile* - простий текстовий файл, що містить інструкції для створення образу контейнера. проста схема створення контейнера (рис 1.3). Це дозволяє розробникам легко реплікувати середовища та автоматизувати процеси розгортання. Іншою важливою характеристикою є *Docker Hub* - централізоване сховище для зберігання та обміну образами контейнерів (рис.1.3).

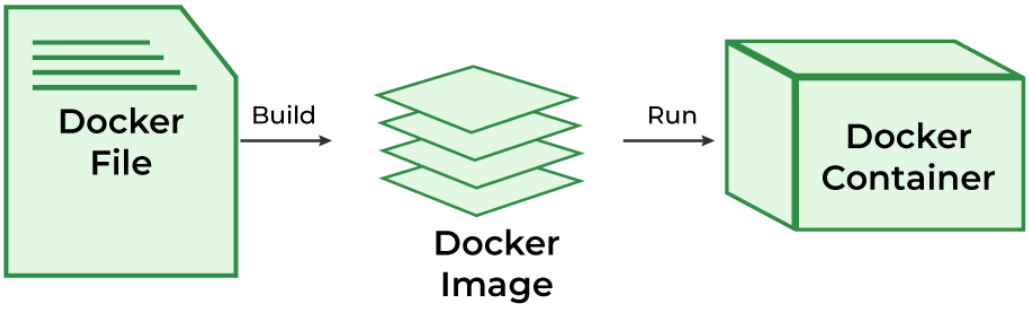

Рис 1.3 - Схема створення контейнера

Dockerfile: Це початковий файл з набором інструкцій для створення *Docker* образу (*Docker Image*). Він містить вказівки на базовий образ, залежності, необхідні файли, команди встановлення та конфігурації.

Build (Побудова): Цей етап включає виконання інструкцій, вказаних у *Dockerfile*, для створення *Docker* образу. В процесі побудови кожна інструкція в *Dockerfile* створює новий шар у *Docker* образі.

Docker Image (*Docker* образ): Після завершення побудови ми отримуємо *Docker* образ, який є "замороженим" станом додатка з усіма його залежностями. Образ може бути збережений локально або завантажений у *Docker* реєстр для подальшого використання.

Run (Запуск): На цьому етапі *Docker* образ використовується для створення запущеного контейнера. Контейнер — це виконавче середовище для образу, що дозволяє додатку працювати.

Docker Container (*Docker* контейнер): Це запущений екземпляр *Docker* образу, який працює як окремий процес у системі. Контейнер ізолює додаток та його виконання від інших процесів та надає потрібні ресурси.

Давайте як приклад створимо простий *Docker file,* який використовує образ *Ubuntu* та виводить текст "*Hello, Docker*! *Oleksandr*" під час запуску:

1. Створимо папку *My_container*

2. У текстовому редакторі створюємо файл як показано в лістингу 1.1 :

Лістинг 1.1. Фрагмент коду *Docker file*

```
# Використання базового образу Ubuntu
FROM ubuntu:latest

# Оновлення списку пакетів
RUN apt-get update

# Встановлення необхідних пакетів (наприклад, у цьому випадку можна
залишити цей крок порожнім)

# Команда, що виконується при запуску контейнера
CMD echo "Hello, Docker! Oleksandr"
```

Ось короткий опис кожного кроку:

1. *FROM ubuntu: latest* - цей рядок визначає базовий образ, в даному випадку останню версію Ubuntu.

2. *RUN apt-get update* - оновлює список доступних пакетів. Це корисно, якщо вам потрібно встановити додаткові пакети у ваш контейнер.

3. *CMD echo "Hello, Docker!"* - це команда, яка виконуватиметься за замовчуванням при запуску контейнера. У цьому випадку вона просто виводить текстове повідомлення.

4. У *Windows* 10 запустити *Windows PowerShell* від адміністратора

5. Перейдіть до відповідної директорії і побудуйте Docker-образ як показано в лістингу 1.2:

Лістинг 1.2. Команди

```
cd c:\my_container
docker build -f My_hello_docker_2023.txt -t my-simple-container
```

У цій команді:

1. *docker build* - це команда для створення *Docker*-образу.
2. *-f My_hello_docker_2023.txt* вказує *Docker* на те, що він повинен використовувати My_hello_docker_2023.txt як *Dockerfile*.
3. *-t my-simple-container* - це тег, який ви надаєте створюваному образу.
4. . (крапка) вказує на поточну директорію як контекст побудови.

Після виконання цих кроків *Docker* почне процес створення образу на основі інструкцій, вказаних у *My_hello_docker_2023.txt.*

6. Тепер, коли у нас є образ, ми можемо створити та запустити контейнер з цього образу, використовуючи наступну команду як показано в лістингу 1.3:

Лістинг 1.3. Команда запустити контейнер

```
docker build -f My_hello_docker_2023.txt -t my-simple-container .
```

У цій команді:

1. docker build - це команда для створення *Docker*-образу.

2. -f *My_hello_docker_2023.txt* вказує *Docker* на те, що він повинен використовувати *My_hello_docker_2023.txt* як *Dockerfile*.

3. -t *my-simple-container* - це тег, який ви надаєте створюваному образу.

4. . (крапка) вказує на поточну директорію як контекст побудови.

Після виконання цих кроків *Docker* почне процес створення образу на основі інструкцій, вказаних у *My_hello_docker_2023.txt.*

7. Щоб переглянути список всіх *Docker*-образів, які ми створили або завантажили на вашу машину, використовуємо команду docker images. Ця команда надає інформацію про всі доступні на вашій машині образи, включаючи їх назву (repository), тег (tag), ID образу, час створення та розмір.

8. Запустім свій *Docker*-контейнер: Тепер, коли у нас є образ, ми можемо створити та запустити контейнер з цього образу, використовуючи наступну команди як показано в лістингу 1.4:

Лістинг 1.4. Команди запуска контейнеру

```
docker run ID (контейнера).
docker run 8c5f606a3c93
```

Приклад працюючої програми (рис 1.4).

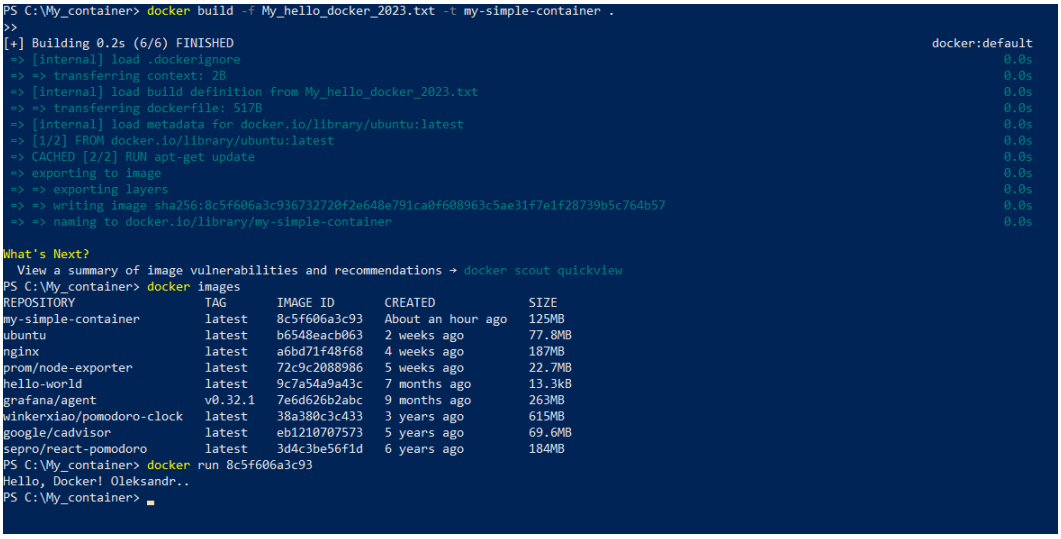

Рис 1.4 - Приклад запуску контейнера

Разом із контейнерами Docer часто застосовують технологію Kubernetes. Kubernetes - це відкрита платформа для управління контейнеризованими додатками та послугами, відома своєю здатністю працювати на різних платформах, своєю гнучкістю та використанням декларативних методів конфігурації та автоматизації. Ця платформа розвивається завдяки своїй великій та активній екосистемі.

Термін "Kubernetes" має грецьке коріння і означає "керманич" або "пілот". Google оприлюднив вихідний код Kubernetes у 2014 році. Платформа базується на п'ятнадцятирічному досвіді Google у управлінні великими робочими навантаженнями, об'єднуючи кращі ідеї та практики, запропоновані спільнотою. На зображенні показані три різні підходи до розгортання додатків: традиційне розгортання, віртуалізоване розгортання та розгортання в контейнерах (рис.1.5).

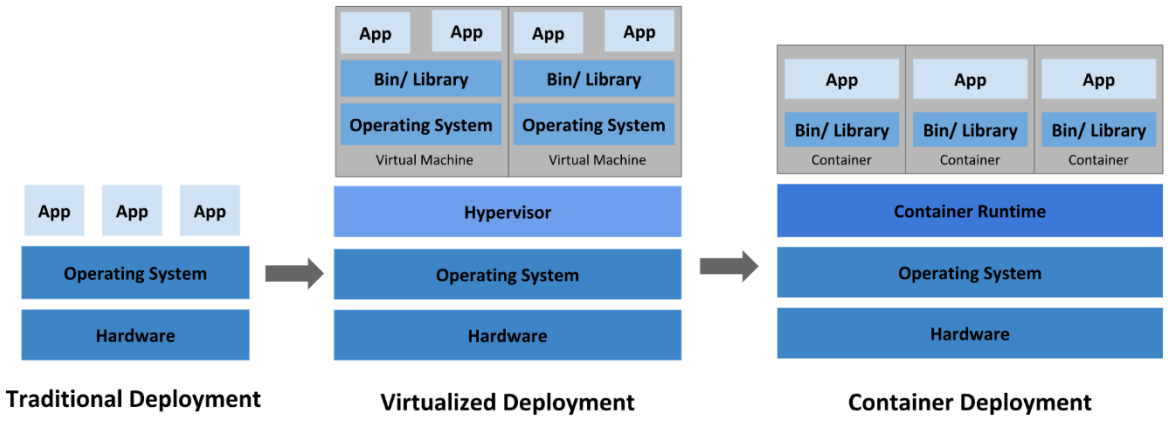

Рис. 1.5 - Три підходи до розгортання додатків

Традиційне розгортання (*Traditional Deployment*)**.** На початку організації запускали застосунки на фізичних серверах. Оскільки в такий спосіб не було можливості задати обмеження використання ресурсів, це спричиняло проблеми виділення та розподілення ресурсів на фізичних серверах. Наприклад: якщо багато застосунків було запущено на фізичному сервері, могли траплятись випадки, коли один застосунок забирав собі найбільше ресурсів, внаслідок чого інші програми просто не справлялись з обов'язками. Рішенням може бути запуск

кожного застосунку на окремому фізичному сервері. Але такий підхід погано масштабується, оскільки ресурси не повністю використовуються; на додачу, це дорого, оскільки організаціям потрібно опікуватись багатьма фізичними серверами.

Віртуалізоване розгортання (Virtualized Deployment). Як рішення - була представлена віртуалізація. Вона дозволяє запускати численні віртуальні машини (*Virtual Machines або VMs*) на одному фізичному ЦПУ сервера. Віртуалізація дозволила застосункам бути ізольованими у межах віртуальних машин та забезпечувала безпеку, оскільки інформація застосунку на одній VM не була доступна застосунку на іншій VM.

Віртуалізація забезпечує краще використання ресурсів на фізичному сервері та кращу масштабованість, оскільки дозволяє легко додавати та оновлювати застосунки, зменшує витрати на фізичне обладнання тощо. З віртуалізацією ви можете представити ресурси у вигляді одноразових віртуальних машин.

Кожна VM є повноцінною машиною з усіма компонентами, включно з власною операційною системою, що запущені поверх віртуалізованого апаратного забезпечення.

Розгортання контейнерів (Container Deployment). Контейнери схожі на VM, але мають спрощений варіант ізоляції і використовують спільну операційну систему для усіх застосунків. Саме тому контейнери вважаються "легкими", в порівнянні з віртуалками. Подібно до VM, контейнер має власну файлову систему, ЦПУ, пам'ять, простір процесів тощо. Оскільки контейнери вивільнені від підпорядкованої інфраструктури, їх можна легко переміщати між хмарними провайдерами чи дистрибутивами операційних систем.

Контейнеризація стала важливою частиною сучасної розробки програмного забезпечення, оскільки вона пропонує ряд переваг:

1. Гнучкість у створенні та впровадженні: Використання контейнерів значно полегшує процес розробки та розгортання програм, особливо в рамках

гнучких методологій розробки, надаючи переваги в ефективності порівняно з традиційними методами використання віртуальних машин.

2. Неперервність у процесах розробки: Забезпечується стабільність і легкість управління процесами розробки, інтеграції та розгортання, включаючи швидкі оновлення та зворотні відкати.

3. Розділення обов'язків: Контейнери дозволяють розділити відповідальності між командами розробників та операцій, полегшуючи управління інфраструктурою.

4. Детальне моніторинг та аналіз*:* Можливість стеження за різними аспектами системи, включаючи стан програм та інші важливі метрики.

5. Уніфікованість робочих середовищ: Контейнери забезпечують однакові умови праці у різних середовищах, від розробника до хмарних рішень.

6. Кросплатформеність: Вони працюють однаково добре в різних операційних системах і хмарних сервісах.

7. Орієнтація на застосунки: Контейнери зміщують фокус з управління операційною системою на більш високий рівень абстракції, зосереджуючись на застосунках.

8. Гнучка мікросервісна архітектура*:* Програми розділяються на малі, незалежні компоненти, що полегшує їх управління порівняно з монолітними системами.

9. Ізоляція ресурсів: Забезпечується стабільність продуктивності програм.

10. Ефективне використання ресурсів: Контейнери дозволяють досягти високої ефективності використання системних ресурсів.

Контейнери - це прекрасний спосіб упакувати та запустити ваші застосунки. У прод (робочої версії продукту) оточенні вам потрібно керувати контейнерами, в яких працюють застосунки, і стежити, щоб не було простою. Наприклад, якщо один контейнер припиняє роботу, інший має бути запущений йому на заміну. Система яка керує цими процесами називається Kubernetes.

Kubernetes надає вам каркас для еластичного запуску розподілених систем. Він опікується масштабуванням та аварійним відновленням вашого застосунку,

пропонує шаблони розгортань тощо. Наприклад, *Kubernetes* дозволяє легко створювати розгортання за стратегією canary у вашій системі. Стратегія canary полягає в тому, що нова версія додатку або сервісу спочатку розгортається лише на малій частині користувачів. Це дає можливість перевірити нову версію в продакшн -середовищі з обмеженим ризиком, перш ніж випустити оновлення для всіх користувачів.

Kubernetes надає вам:

1. Виявлення сервісів та балансування навантаження. *Kubernetes* може надавати доступ до контейнера, використовуючи *DNS*-ім'я або його власну *IP*-адресу. Якщо контейнер зазнає завеликого мережевого навантаження, Kubernetes здатний збалансувати та розподілити його таким чином, щоб якість обслуговування залишалась стабільною.

2. Оркестрація сховища інформації. *Kubernetes* дозволяє вам автоматично монтувати системи збереження інформації на ваш вибір: локальні сховища, рішення від хмарних провайдерів тощо.

3. Автоматичне розгортання та відкочування. За допомогою *Kubernetes* ви можете описати бажаний стан контейнерів, що розгортаються, і він регульовано простежить за виконанням цього стану. Наприклад, ви можете автоматизувати в Kubernetes процеси створення нових контейнерів для розгортання, видалення існуючих контейнерів і передачу їхніх ресурсів на новостворені контейнери.

4. Автоматичне розміщення задач. Ви надаєте *Kubernetes* кластер для запуску контейнеризованих задач і вказуєте, скільки ресурсів ЦПУ та пам'яті (*RAM*) необхідно для роботи кожного контейнера. *Kubernetes* розподіляє контейнери по вузлах кластера для максимально ефективного використання ресурсів.

5. Самозцілення. *Kubernetes* перезапускає контейнери, що відмовили; заміняє контейнери; зупиняє роботу контейнерів, що не відповідають на задану користувачем перевірку стану, і не повідомляє про них клієнтам, допоки ці контейнери не будуть у стані робочої готовності.

6. Управління секретами та конфігурацією. *Kubernetes* дозволяє вам зберігати та керувати чутливою інформацією, такою як паролі, *OAuth* токени та *SSH* ключі. Ви можете розгортати та оновлювати секрети та конфігурацію без переобирання образів ваших контейнерів, не розкриваючи секрети в конфігурацію стека.

Перед контейнеризацією відкриті нові горизонти. Інтеграція з технологіями штучного інтелекту та машинного навчання, розвиток *Edge Computing*, де контейнери можуть бути використані для ефективного управління даними на краю мережі, відкривають нові можливості. Однак, разом з новими можливостями приходять і нові завдання, такі як управління залежностями, безпека та інтеграція з різноманітними середовищами.

Ми розглянемо *Docker, Kubernetes, Amazon ECS (Elastic Container Service*), *Microsoft Azure, Salesforce.com, Google Cloud, Oracle* оцінюючи їх ключові характеристики, переваги та обмеження. *Docker* - фактичний стандарт.

Docker є однією з найпопулярніших платформ для контейнерізації. Він вирізняється своєю простотою використання, широкою підтримкою у спільноті, та сильною інтеграцією з різними хмарними сервісами.

Переваги:

- легкість управління, велике співтовариство, широкий набір інструментів.

Недоліки:

- обмеження в плані нативної оркестрації контейнерів.

Kubernetes, створений *Google*, є провідною системою оркестрації контейнерів. Він забезпечує більшу гнучкість та масштабування порівняно з *Docker Swarm* та іншими рішеннями.

Переваги:

- висока масштабованість, гнучкість управління, сильна підтримка спільноти.

Недоліки:

- висока складність, крута крива навчання.

Amazon ECS (Elastic Container Service) є хмарною службою, яка дозволяє легко запускати додатки в контейнерах на інфраструктурі *AWS*.

Переваги:

- тісна інтеграція з *AWS*, простота використання.

Недоліки:

- залежність від *AWS*, обмеження у використанні іншої хмарної інфраструктури.

Microsoft Azure пропонує широкий спектр послуг контейнерізації, включаючи *Azure Kubernetes Service (AKS), Azure Container Instances (ACI*) та інші інтегровані хмарні рішення.

Переваги:

- тісна інтеграція з іншими продуктами Microsoft, висока надійність, підтримка enterprise рівня.

Недоліки:

- може бути складніше у використанні для новачків, вищі витрати порівняно з деякими іншими хмарними платформами.

При порівнянні цих платформ важливо враховувати декілька ключових аспектів: легкість використання, масштабованість, підтримка спільноти, інтеграцію з іншими сервісами та витрати. Наприклад, хоча *Kubernetes* пропонує великі можливості масштабування та гнучкості, *Docker* може бути простішим у використанні для менших проектів або для команд з обмеженим досвідом в оркестрації контейнерів.

Історії успіху використання цих платформ можна знайти у широкому спектрі індустрій. Великі корпорації часто віддають перевагу *Kubernetes* або *Azure* через їх високу масштабованість та інтеграцію з корпоративними системами. З іншого боку, стартапи та малі компанії можуть вибрати *Docker* або *Amazon ECS* для більш простого розгортання та управління.

1.3 Моніторинг у контейнерних системах

Моніторинг у контейнерних системах є ключовим елементом для забезпечення їх ефективної та безперебійної роботи. Основні компоненти моніторингу - це збір метрик та логів, які надають важливу інформацію про стан системи, продуктивність та потенційні проблеми.

Метрики - це кількісні дані, які можуть включати інформацію про використання ресурсів (ЦП, пам'ять, дисковий простір), час відповіді, кількість запитів тощо. Вони важливі для розуміння загального стану системи та її продуктивності.

Логи - це записи подій або дій, що відбуваються у системі. Вони містять інформацію про помилки, попередження, інформаційні повідомлення та інші важливі події. Аналіз логів дозволяє виявити та вирішити проблеми, а також оптимізувати роботу системи.

Одним з основних завданнь у моніторингу контейнерних систем є їх складність та динамічність. Контейнери можуть швидко створюватися та знищуватися, що ускладнює постійний збір даних та аналіз стану системи.

Необхідність інтеграції моніторингу з різними інструментами та платформами також є важливим завданням. Ефективний моніторинг вимагає збору даних з різних джерел та їх агрегації у централізованій системі.

Автоматизація процесів моніторингу є ключовою для підтримки високої продуктивності та стабільності в динамічних середовищах контейнерізації. Це включає автоматичне виявлення проблем та оповіщення адміністраторів.

1.4 Інструменти для моніторингу

Інструменти для моніторингу контейнерних систем грають важливу роль у підтримці стабільності та продуктивності IT-інфраструктури. У цьому розділі ми розглянемо два ключових інструменти - *Prometheus* та *Grafana* - та аналізуємо їх можливості та обмеження.

Prometheus є відкритим інструментом для моніторингу та оповіщення, який широко використовується у спільноті контейнерізації. Його головні переваги полягають у можливості збирати та зберігати метрики в реальному часі, гнучкій системі оповіщень та підтримці різноманітних джерел даних. Архітектура *Prometheus* відображена на (рис.1.6).

На зображенні ви бачите схему моніторингової системи, яка використовує *Prometheus* в екосистемі *Kubernetes*. Ось її ключові компоненти та їхні функції:

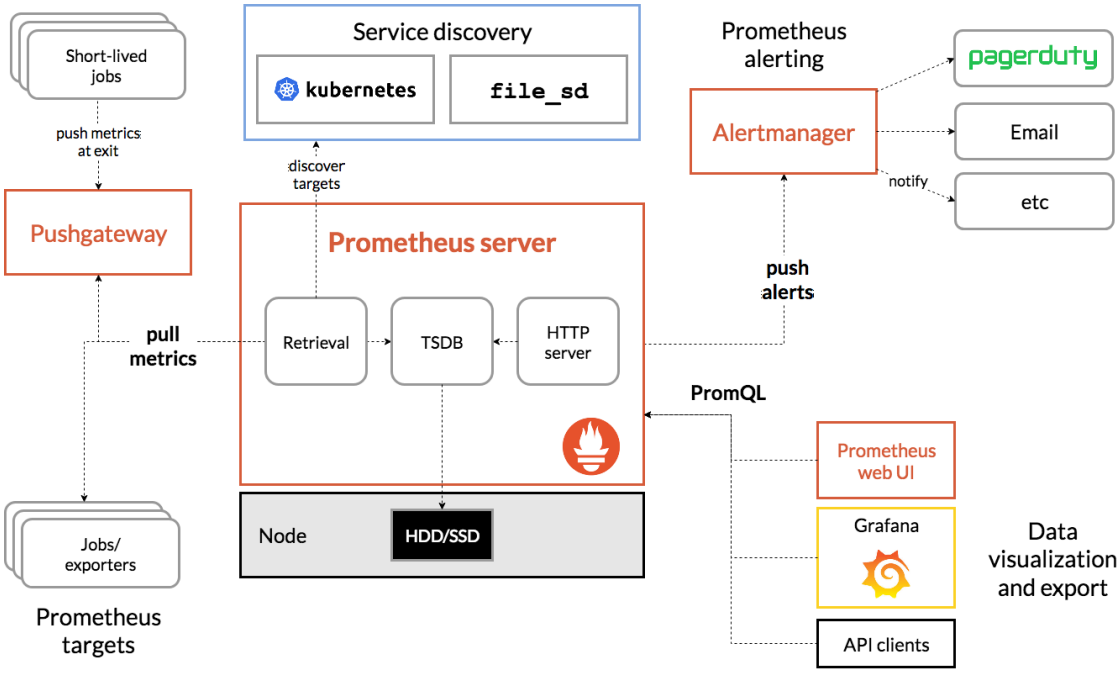

Рис. 1.6 – Архітектура *Prometheus*

1. *Service Discovery*: Використовується для автоматичного виявлення сервісів та робочих вузлів в Kubernetes, які потрібно моніторити. Це може бути налаштовано через file_sd для динамічного оновлення цілей моніторингу.

2. *Prometheus Server*: Основний компонент, який виконує збір метрик (Retrieval), зберігає їх в часовій базі даних (TSDB), та надає доступ до даних через HTTP server. Метрики збираються з вузлів (Nodes) та експортерів (Jobs/Exporters).

3. Pushgateway: Використовується для збору метрик від короткочасних задач, які не можуть постійно публікувати метрики, тому що вони швидко завершують свою роботу.

*4. Alertmanager***:** Компонент *Prometheus*, який управляє сповіщеннями. Alertmanager отримує сповіщення від *Prometheus Server* і відправляє їх до різних сервісів, таких як Email, PagerDuty, та інші.

*5. PromQL***:** Мова запитів *Prometheus*, яка дозволяє виконувати складні запити до даних, зібраних *Prometheus*, і використовується для створення дашбордів та алертів.

6. Grafana: Система для візуалізації даних, яка інтегрується з *Prometheus* для створення інформативних дашбордів, що дозволяють легко інтерпретувати зібрані метрики.

7. Prometheus Web UI ma API Clients: Веб-інтерфейс *Prometheus*, який дозволяє користувачам переглядати метрики та виконувати запити через *PromQL*, а також програми, що використовують *API Prometheus* для автоматизації роботи з даними:

Переваги:

- гнучкість, підтримка великої кількості експортерів, висока масштабованість.

Недоліки:

- може бути складним у налаштуванні, обмежена підтримка довгострокового зберігання даних.

Більшість компонентів Prometheus написані на Go, що спрощує їх складання та розгортання у вигляді статичних двійкових файлів.

Prometheus збирає метрики з інструментованих завдань або безпосередньо, або через проміжний шлюз для короткочасних завдань. Він зберігає всі очищені зразки локально і застосовує правила до цих даних, щоб агрегувати і записувати нові часові ряди на основі існуючих даних, або генерувати оповіщення. Grafana або інші споживачі API можна використовувати для візуалізації зібраних даних.

Grafana є популярним інструментом для візуалізації метрик. Він дозволяє створювати інтерактивні дашборди, що показують ключові показники продуктивності та стану системи. Grafana може інтегруватися з різними джерелами даних, включаючи Prometheus (рис 1.7).

Grafana: Графічна панель управління, що показує різноманітні графіки та візуалізації. Grafana — це інструмент візуалізації, який часто використовується для моніторингу інфраструктури та аналізу даних метрик.

Metrics: Метрики, які представляють кількісні дані, зазвичай щодо використання ресурсів, продуктивності системи чи додатків.

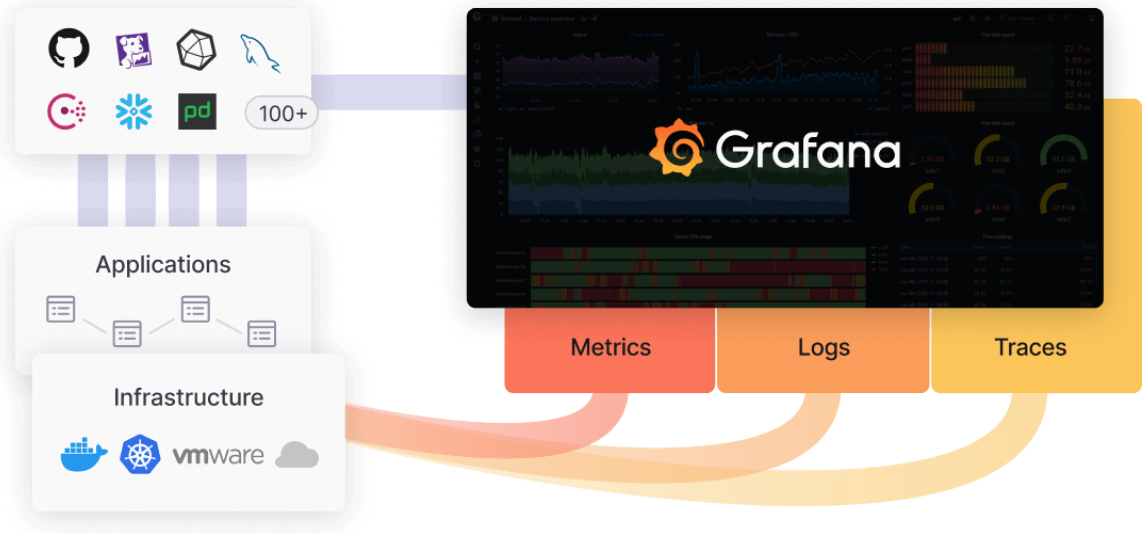

Рис.1.7 - Архітектура Grafana

Logs: Логи або журнали подій, які містять більш деталізовану інформацію про події в системі, помилки, запити тощо.

Traces: Сліди, які використовуються для відстеження та візуалізації взаємодій всередині додатків, зокрема для виявлення проблем з продуктивністю та відладки.

Applications: Додатки, які моніторяться або аналізуються.

Infrastructure: Інфраструктура, яка може включати сервери, віртуальні машини, контейнери тощо.

VMware: VMware — це компанія та платформа для віртуалізації, яка може інтегруватися з Grafana для моніторингу віртуальної інфраструктури.

100+: Позначає, що Grafana підтримує понад 100 різних плагінів та інтеграцій.

Іконки: Представляють різні плагіни або сервіси, що можуть бути інтегровані з *Grafana* для збору даних або управління моніторингом. Кожна іконка символізує певний інструмент або платформу:

Переваги:

- інтуїтивно зрозумілий інтерфейс, гнучкість налаштувань, підтримка широкого спектру джерел даних.

Недоліки:

- вимагає попередніх знань для ефективного використання, обмежена функціональність у плані оповіщень.

Prometheus є потужним інструментом для збору метрик, який особливо популярний у середовищах, які використовують *Kubernetes*. Він дозволяє збирати дані з різних джерел, включаючи хости, мікросервіси та інші компоненти системи. Його модель даних та запитів дає можливість глибокого аналізу продуктивності системи.

Prometheus відіграє важливу роль у моніторингу комплексних систем, особливо коли мова йде про великі об'єми даних та динамічні середовища. Важливим є розуміння його моделі даних та запитів для належного аналізу метрик. *Prometheus* може бути налаштований для збору даних з різноманітних джерел, включаючи *Kubernetes*, системні метрики та мікросервіси.

Оптимізація та налаштування: Незважаючи на складність налаштувань, належна конфігурація *Prometheus* може значно підвищити ефективність моніторингу.

Вивчення можливостей *Grafana* дозволяє адміністраторам та розробникам легко інтерпретувати масиви даних, зібраних *Prometheus* та іншими інструментами. Вона надає інтуїтивно зрозумілий інтерфейс для створення дашбордів, що можуть включати різноманітні типи візуалізацій - від простих графіків до складних інтерактивних діаграм..

Інтеграція з *Prometheus*: Однією з ключових особливостей *Grafana* є її здатність інтегруватися з *Prometheus*, що дозволяє візуалізувати зібрані метрики в різноманітних форматах.

Кастомізація та розширення: *Grafana* надає великі можливості для кастомізації дашбордів та віджетів, що дозволяє точно налаштувати відображення даних під конкретні потреби користувача.

Інтеграція *Prometheus* та *Grafana* є ключовою для створення ефективної системи моніторингу. *Prometheus* збирає та агрегує метрики, тоді як *Grafana* використовує ці дані для створення динамічних дашбордів. Така інтеграція дозволяє швидко виявляти та реагувати на проблеми, забезпечуючи оптимальну продуктивність та стабільність системи.

Завершуючи аналіз, важливо підкреслити, що вибір інструментів для моніторингу залежить від конкретних потреб проекту. *Prometheus* та *Grafana* часто є оптимальним вибором для систем, що використовують контейнери та мікросервіси, в той час як *ELK Stack* та інші інструменти можуть бути кращим вибором для систем з великими об'ємами логів або специфічних вимог до збору даних.

1.5 Стратегії моніторингу

Ефективний моніторинг у контейнерних системах стикається з завданням масштабування та управління. Цей розділ розгляне стратегії та практики, які допомагають ефективно управляти великими та складними середовищами контейнерізації.

Одним з основних завданням масштабування є забезпечення стабільності та продуктивності системи при збільшенні кількості контейнерів. Це включає збір та аналіз великої кількості метрик, що може створювати навантаження на систему моніторингу.

Швидке виявлення та реагування на інциденти є ключовими для підтримки високої доступності та продуктивності контейнерних систем. Це включає моніторинг за допомогою візуальних дашбордів, налаштування оповіщень та використання автоматизованих інструментів для швидкого реагування.

Розробка чіткого плану реагування на інциденти, включаючи процедури виявлення, оцінки та вирішення проблем, є важливою складовою ефективного моніторингу. Також необхідно передбачити можливість швидкого відновлення системи після збоїв.

Використання автоматизованих інструментів та технік може значно полегшити процес моніторингу великих систем. Автоматизація включає налаштування правил та процесів, які дозволяють системі моніторингу самостійно реагувати на зміни стану контейнерів та інших компонентів системи.

Побудова масштабної системи моніторингу, яка може ефективно обробляти великі об'єми даних та адаптуватися до змін у середовищі, є ключовим аспектом для великих контейнерних систем. Це може включати використання розподілених систем збору даних, оптимізацію зберігання та аналізу метрик.

Ефективний моніторинг вимагає інтегрованого підходу, який включає автоматизацію, масштабування, та гнучке реагування на інциденти. Оптимізація моніторингу в контейнерних середовищах дозволяє не тільки підтримувати стабільність та продуктивність, але й покращує загальну безпеку системи.

Сучасні технології, такі як штучний інтелект та машинне навчання, відкривають нові можливості для інтелектуального моніторингу. Використання передових алгоритмів дозволяє автоматично аналізувати тенденції, прогнозувати потенційні проблеми та оптимізувати ресурси системи.

Ефективний моніторинг в контейнерних системах часто вимагає використання кількох інструментів разом. Наприклад, інтеграція *Prometheus* з *Grafana* для візуалізації метрик, або використання *ELK Stack* разом з інструментами автоматизації для розширеного аналізу логів.

Важливою стратегією є адаптація системи моніторингу до змінних умов та нових вимог. Це може включати регулярне оновлення інструментів, налаштувань та підходів до моніторингу відповідно до розвитку технологій та змін у бізнес-процесах.

Висновки до першого розділу

У результаті аналізу предметної області встановлено, що контейнеризація надає значні переваги для розробки та розгортання додатків, зокрема завдяки ізоляції ресурсів, легкості управління та масштабування. Також було виявлено,

що для підтримки стабільності та високої доступності сервісів необхідний ретельний моніторинг систем, що базуються на контейнерах.

Розглянуто та порівняно ключові платформи контейнерізації, такі як *Docker* та *Kubernetes*, які є лідерами на ринку. Досліджено їх основні можливості, інтеграцію з інструментами моніторингу, такими як *Prometheus* та *Grafana*, та встановлено критерії для оцінювання ефективності їх використання в різноманітних середовищах розробки.

Визначено основні напрямки для подальшого дослідження, включаючи розробку оптимізованих стратегій моніторингу для контейнеризованих систем, аналіз завдань, пов'язаних з масштабуванням та управлінням ресурсів, та розробку плану для реагування на інциденти, що забезпечить підвищену готовність системи до відмов.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ СИСТЕМ КОНТЕЙНЕРІЗАЦІЇ

2.1 Аналіз вимог

У сучасному світі IT, де швидкість та гнучкість стають ключовими для бізнес-успіху, системи контейнерізації відіграють вирішальну роль. Контейнери – це легковагові та портативні середовища виконання, які дозволяють розробникам пакувати та розповсюджувати додатки незалежно від оточення. Це спрощує процеси розробки, тестування та розгортання, але також створює завдання для моніторингу та управління.

Ключові показники ефективності (*KPI*) та метрики в моніторингу систем контейнерізації відіграють важливу роль у забезпеченні ефективності та надійності цих систем. *KPIs* – це кількісні показники, які допомагають оцінити успішність системи в досягненні зазначених бізнес-цілей. Наприклад, *KPI* може включати параметри, як-от час відгуку системи, пропускна спроможність, відсоток часу простою, та ефективність використання ресурсів.

Важливість *KPIs* та метрик полягає в тому, що вони забезпечують чіткість та об'єктивність у процесі управління IT-системами. Вони допомагають керівникам проектів та IT-спеціалістам зрозуміти, як система функціонує в реальному часі, і виявити потенційні проблемні зони для попередження відмов чи зниження продуктивності.

Наприклад, високий відсоток часу простою може вказувати на нестабільність системи, тоді як низька пропускна спроможність може свідчити про неефективне використання ресурсів або проблеми з мережевою інфраструктурою. Точне відстеження цих показників дозволяє швидко вжити коригувальних заходів для оптимізації роботи системи.

Таким чином, *KPIs* та метрики не просто відіграють роль у забезпеченні безперервної та ефективної роботи систем контейнерізації, а й є невід'ємною частиною стратегічного планування та управління в IT. Вони допомагають

визначити, чи відповідає система бізнес-вимогам та чи готова вона до майбутніх завдань, що, у свою чергу, є критично важливим для успішної діяльності будь-якої організації.

Створення локального монітора запущених контейнерів Docker. Отримання *KPIs* метрик.

Метрики я розбив на дві таблиці:

- метрики, які характеризують контейнери;
- метрики хоста (локального комп'ютера).

Ось список *KPIs* (ключових показників ефективності), які можуть бути важливими для контейнерізації.

Таблиця 2.1

Список *KPIs* для контейнерізації

Номер	Назва (англ.)	Значення	Збірник даних
1	Memory usage, %	Відсоток використаної пам'яті	Node Exporter
2	CPU usage, %	Відсоток навантаження на процесор	Node Exporter
3	Filesystem usage, %	Відсоток використання простору на диску	Node Exporter
4	Uptime	Час з моменту останнього перезапуску системи	Node Exporter
5	Node Network Traffic	Обсяг даних, що передаються та приймаються вузлом	Node Exporter
6	Node Memory	Загальний обсяг доступної пам'яті на вузлі	Node Exporter
7	Container Memory Usage	Використання пам'яті конкретним контейнером	cAdvisor
8	Running Container	Кількість контейнерів, що зараз працюють	cAdvisor/Docker API

Продовження таблиці 2.1

9	Running Version	Версія запущеної програми / контейнера	cAdvisor/Docker API
10	Container Performance	Метрики продуктивності контейнерів, такі як використання ЦП, пам'яті	cAdvisor
11	Container Network Output	Обсяг мережевого трафіку, генерованого контейнером	cAdvisor

Ось список KPIs (ключових показників ефективності), які можуть бутиважливими для локального комп'ютера.

Таблиця 2.2

Список KPIs для локального комп'ютера

Номер	Назва (EN)	Значення	Збірник даних
1	Processes N	Кількість активних процесів	Node Exporter /WMI
2	CPU Utilization	Відсоток часу використання ЦП	Node Exporter /WMI
3	RAM Utilization	Відсоток використання оперативної пам'яті	Node Exporter /WMI
4	Disc Utilization	Відсоток використання диску	Node Exporter /WMI
5	Disc Total Reads	Загальна кількість операцій читання з диску	Node Exporter /WMI
6	Net Total Transfers	Загальний обсяг передачі даних через мережу	Node Exporter /WMI
7	Threads N	Кількість потоків, що використовуються процесором	Node Exporter /WMI

8	vCPU Cores N	Кількість віртуальних ядер ЦП	Node Exporter /WMI

Продовження таблиці 2.2

9	CPU Frequency	Поточна частота процесора в МГц	Node Exporter /WMI
10	RAM Total	Загальний обсяг доступної оперативної пам'яті	Node Exporter /WMI
11	Disc Total	Загальна ємність диску	Node Exporter
12	Disc Total Writes	Загальна кількість операцій запису на диск	Node Exporter /WMI
13	CPU Utilization Total	Загальний час, проведений ЦП на обробку задач	Node Exporter /WMI
14	CPU Utilization per Mode	Час, проведений ЦП у різних режимах	Node Exporter /WMI
15	CPU Utilization per Core	Відсоток використання кожного ядра ЦП	Node Exporter /WMI
16	CPU Frequency AVG	Середня частота процесора за певний період часу	Node Exporter /WMI
17	Processes Count	Загальна кількість процесів	Node Exporter /WMI
18	Threads Count	Загальна кількість потоків	Node Exporter /WMI
19	Network Utilization	Відсоток використання смуги пропускання мережі	Node Exporter /WMI
20	Network Transfers	Загальна кількість переданих мережевих пакетів	Node Exporter /WMI

2.2 Архітектура системи моніторингу

У рамках моєї дипломної роботи я розглядаю архітектуру системи моніторингу для контейнерізованих IT-середовищ, зокрема, на основі *Docker*.

Цей розділ деталізує ключові компоненти системи та обґрунтування вибору архітектурного стилю.

Для збору даних про стан контейнерів використовуються спеціалізовані агенти моніторингу, які встановлені в кожному контейнері. Вони збирають інформацію про використання ресурсів, стан процесів, логи та інші критичні метрики. Наприклад *cAdvisor, WMI, Node exporter*.

Центральний сервер моніторингу збирає, обробляє, та зберігає дані, отримані від агентів. Він відповідає за агрегацію даних, їх аналіз, та виявлення аномалій.

Для ефективного зберігання та швидкого доступу до історичних даних моніторингу використовується база даних. Вона може бути реалізована як *SQL* або *NoSQL* рішення залежно від вимог до обсягу даних та швидкості доступу.

Веб-інтерфейс або додаток для настільних комп'ютерів, який дозволяє користувачам переглядати зібрану інформацію, аналітичні звіти, графіки та сповіщення про стан контейнерів.

Компонент, який відповідає за надсилання сповіщень та тривоги у разі виявлення проблем або аномалій у роботі контейнерів.

При проектуванні системи моніторингу я зробив вибір на користь мікросервісної архітектури, та використання готових програм для систем моніторингу з наступних причин:

Гнучкість та масштабованість: Мікросервісна архітектура дозволяє легко масштабувати окремі компоненти системи, що є важливим для обробки зростаючих обсягів даних моніторингу.

Надійність: Завдяки розподілу функціоналу на незалежні сервіси, збій одного компоненту не призводить до виходу з ладу всієї системи.

Легкість обслуговування та оновлення: Кожен мікросервіс може бути оновлений незалежно, що спрощує процес внесення змін та випуску нових версій.

Технологічна гнучкість: Мікросервісна архітектура дозволяє використовувати найбільш підходящі технології для кожного компоненту системи.

Враховуючи вищезазначене, мікросервісна архітектура здається оптимальним вибором для системи моніторингу, що забезпечує високу гнучкість, масштабованість та надійність.

2.3 Огляд готових систем моніторингу

У цьому розділі я розглядаю наявні системи моніторингу, зокрема *Docker*, *Prometheus* та *Grafana*, та їхнє застосування та інтеграцію в реальних IT-проектах.

Prometheus - це відкритий інструмент моніторингу та оповіщення, який став популярним у спільноті через свою простоту та ефективність. Він збирає метрики від об'єктів, що перебувають під спостереженням, зберігаючи дані у вигляді часових рядів. Це дозволяє легко відстежувати зміни показників та визначати тренди. Prometheus підтримує гнучкі запити та має потужну систему оповіщень.

Grafana - це платформа для візуалізації та аналізу даних, яка часто використовується разом з *Prometheus*. *Grafana* дозволяє створювати інтуїтивно зрозумілі дашборди, які відображають метрики у реальному часі. Це робить її ідеальним інструментом для візуального представлення даних моніторингу (рис 2.1).

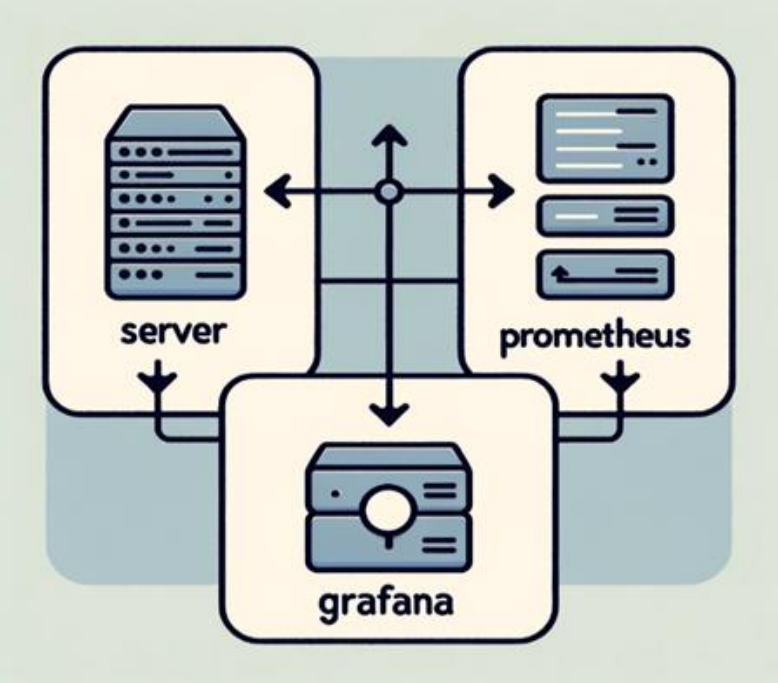

Рис 2.1 - Схема моніторингу

Server - різні сервери або комп'ютери, які можуть перебувати під спостереженням..

Prometheus — система, яка збирає метрики з серверів.

Grafana - Це інструмент для візуалізації даних, який бере зібрані Prometheus метрики та відображає їх у вигляді графіків та дашбордів для аналізу.

Розглянемо основні переваги *Prometheus* та *Grafana*:

1. Гнучкість у зборі даних: *Prometheus* може збирати дані як зі статичних, так і з динамічних джерел.

2. Висока масштабованість: Обидва інструменти підходять для масштабних систем і можуть обробляти великі обсяги даних.

3. Легкість інтеграції*:* Вони легко інтегруються з різноманітними сервісами та платформами.

4. Спільнота та підтримка: Велика спільнота користувачів та розробників забезпечує постійне оновлення та підтримку.

Інтеграція *Prometheus* та *Grafana* в реальних проектах зазвичай передбачає наступні кроки:

1. Налаштування *Prometheus* для збору даних: Це включає конфігурацію цілей моніторингу та налаштування інтервалів збору даних.

2. Зберігання та обробка даних: *Prometheus* ефективно обробляє зібрані дані, забезпечуючи швидкий доступ до історичних даних.

3. Візуалізація даних за допомогою *Grafana*: Створення дашбордів, які демонструють ключові показники, які ми вказали.

2.4 Інтеграція з інфраструктурою контейнерів

Інтеграція систем моніторингу, таких як *Prometheus*, з інфраструктурою контейнерів, яка зазвичай використовує *Docker* або *Kubernetes*, є ключовим аспектом для забезпечення високої доступності та надійності сервісів.

Розглядаючи методи інтеграції, основну увагу слід зосередити на спроможності системи збирати метрики з кожного контейнера, що включає використання ресурсів, стан процесів, а також логи.

Один зі способів інтеграції – використання *Docker API* для збору метрик. Prometheus може налаштовуватись на збір даних безпосередньо з *Docker daemon,* використовуючи вбудовані механізми моніторингу, які надає *Docker*.

Інший варіант – використання *cAdvisor* для збору даних про використання ресурсів і продуктивність роботи контейнерів. *cAdvisor* інтегрується з *Prometheus* і дозволяє відстежувати використання ЦПУ, пам'яті, файлової системи та мережевих ресурсів на рівні кожного контейнера.

Коли справа доходить до *Kubernetes*, інтеграція включає використання *Prometheus Operator*, який спрощує розгортання *Prometheus* та його налаштування у кластері *Kubernetes*. Він автоматично виявляє сервіси та поди, які потребують моніторингу, і генерує конфігурації для збору метрик.

Мережева взаємодія в контейнеризованих середовищах вимагає моніторингу трафіку між контейнерами та сервісами. Prometheus може збирати дані про мережеві запити, затримки та кількість трафіку, що дозволяє виявляти мережеві збої або неефективне використання мережевих ресурсів.

Що стосується зберігання даних, то системи моніторингу мають забезпечувати збереження великих обсягів логів та метрик. Використання таких рішень, як бази даних часових рядів, наприклад *TSDB*, що використовується в

Prometheus, або інтеграція з зовнішніми системами зберігання, такими як *Amazon S3* або *Google Cloud Storage*, може бути необхідним для довгострокового зберігання історичних даних.

Інтеграція з інфраструктурою контейнерів також передбачає впровадження політик безпеки та дотримання норм регулювання, що забезпечують захист зберігання даних та їх передачі. Важливо, щоб система моніторингу враховувала ці аспекти, гарантуючи, що моніторинг і зберігання даних відбуваються в безпечному та відповідальному режимі.

2.5 Безпека та конфіденційність

У моїй дипломній роботі я приділяю значну увагу аспектам безпеки та конфіденційності в контексті моніторингу систем контейнерізації. Оскільки системи моніторингу збирають та обробляють великі обсяги даних, включаючи потенційно чутливу інформацію, необхідно впровадити строгі заходи безпеки, щоб захистити ці дані від несанкціонованого доступу або витоку.

Основні заходи, які я розглядаю, включають:

1. Шифрування: Всі дані повинні бути зашифровані під час передачі та зберігання. Це включає використання *TLS* для шифрування даних, які передаються між компонентами системи моніторингу, а також шифрування даних на диску.

2. Керування доступом: Необхідно впровадити систему керування доступом на основі ролей (*RBAC*), щоб обмежити доступ до даних та оперативних функцій системи лише уповноваженим особам.

3. Аудит: Систематичний аудит діяльності користувачів та системних процесів допомагає виявляти підозрілу активність та запобігати потенційним порушенням безпеки.

4. Резервне копіювання: Регулярне створення резервних копій даних допомагає відновити інформацію у випадку її втрати або пошкодження через атаки або технічні збої.

Важливим аспектом розробки системи моніторингу є забезпечення її відповідності до нормативних вимог, таких як *GDPR* у Європейському Союзі чи *HIPAA* у США, які регулюють зберігання та обробку персональних даних.

Для цього необхідно:

1. Класифікація даних: Визначити, які дані є персональними та підлягають особливому захисту.

2. Політики приватності та використання даних: Розробити та впровадити чіткі політики, що описують, як і для яких цілей збираються та використовуються дані.

3. Згода на обробку даних: Забезпечити можливість отримання згоди від осіб, чиї дані обробляються, та їхнє право на видалення цих даних.

4. Заходи щодо захисту даних за замовчуванням: Впровадження технічних та організаційних заходів для захисту даних від початку їх збору.

Завдяки цим заходам, система моніторингу не лише забезпечує високий рівень безпеки даних, а й допомагає організації уникнути юридичних ризиків, пов'язаних з обробкою та зберіганням чутливої інформації.

Резюмуючи, безпека та конфіденційність є критично важливими компонентами системи моніторингу. Необхідно враховувати як технічні, так і юридичні аспекти для створення надійної та відповідальної системи, що може забезпечити ефективний захист даних і дотримання відповідних нормативних стандартів.

2.6 Тестування та оптимізація

Як студент, який зосереджений на розробці та удосконаленні систем моніторингу в контексті контейнерізації, я визнаю критичну важливість тестування та оптимізації. Ці процеси забезпечують не тільки безпеку та стабільність системи, але й гарантують, що вона працює з максимальною продуктивністю.

Тестування системи моніторингу має включати:

1. Функціональне тестування: Переконатися, що кожен компонент системи працює згідно зі специфікацією і виконує свої задачі правильно.

2. Навантажувальне тестування: Оцінити здатність системи обробляти великі обсяги даних, які виникають у реальних умовах роботи.

3. Тестування безпеки: Перевірити систему на вразливості та потенційні зловживання, включно з тестуванням на проникнення та аудитом коду.

4. Тестування продуктивності: Вимірювання часу відгуку системи та швидкості обробки даних.

5. Тестування відмовостійкості: Визначення, наскільки надійно система продовжує працювати при відмовах окремих компонентів або підсистем.

Оптимізація продуктивності системи моніторингу вимагає комплексного підходу, який включає:

1. Рефакторинг коду: Перегляд та оптимізація коду для підвищення ефективності та зменшення споживання ресурсів.

2. Масштабування: Використання горизонтального (додавання екземплярів) або вертикального (додавання ресурсів) масштабування для збільшення продуктивності системи.

3. Кешування: Застосування кешування даних для зниження затримок та зменшення навантаження на базу даних.

4. Балансування навантаження: Розподіл запитів або задач між кількома серверами або процесами для забезпечення рівномірного розподілу навантаження.

5. Оптимізація баз даних: Налаштування індексів, оптимізація запитів та структур баз даних для підвищення швидкості доступу до даних.

Крім технічних аспектів, важливим елементом оптимізації є встановлення процесів безперервного моніторингу та аналізу продуктивності, що дозволяє своєчасно виявляти та усувати "вузькі місця" в системі.

Висновки до другого розділу

В аналізі вимог були чітко визначені основні цілі моніторингу, які включають забезпечення стабільності роботи контейнерізованих систем та оперативне виявлення та реагування на інциденти. Ключові показники ефективності (*KPIs*) та метрики були встановлені для оцінки продуктивності та надійності систем.

Було розроблено архітектуру системи моніторингу, що включає в себе вибір компонентів та архітектурного стилю, спрямованого на гнучкість та масштабованість, забезпечуючи тим самим можливість адаптації до мінливих вимог бізнесу.

Проведено ретельний огляд існуючих готових рішень для моніторингу, зокрема *Prometheus* та *Grafana*, і їх інтеграцію у реальні проекти.

Ці висновки підтверджують важливість глибокого аналізу та ретельного планування при проектуванні систем моніторингу і слугують основою для практичної частини роботи, де буде здійснено реалізацію та випробування розробленої системи.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ МОНІТОРИНГУ КОНТЕЙНЕРІЗАЦІЇ

3.1. Встановлення та налаштування Docker

Для встановлення *Docker* на операційній системі *Windows*, важливо забезпечити відповідність системи наступним вимогам: наявність windows 10 Pro, Enterprise або Education, активований Hyper-V та мінімум 4 ГБ оперативної пам'яті.

Процес встановлення *Docker Desktop*. Використання *PowerShell* з правами адміністратора дозволяє завантажити інсталяційний файл *Docker Desktop* через відповідну команду Invoke-WebRequest. Після завантаження, інсталяційний файл запускається, ініціюючи процес установки Docker на комп'ютер, як показано в лістингу 3.1.

Лістинг 3.1. Процес встановлення *Docker Desktop*

```
Invoke-WebRequest -Uri
"https://download.docker.com/win/stable/Docker%20for%20Windows%20Installe
r.exe" -OutFile "DockerInstaller.exe"

Start-Process -FilePath "DockerInstaller.exe" -Wait
```

По завершенню інсталяції, необхідно перезавантажити комп'ютер та запустити *Docker Desktop*. Правильна робота Docker підтверджується наявністю іконки Docker в системному треї та відповідним повідомленням.

Виконання команди *docker –version* у *PowerShell* дозволяє перевірити успішність установки, виводячи версію встановленого Docker.

Docker Hub — це хмарний сервіс, який функціонує як централізований ресурс для розміщення та розподілу контейнерних образів. Він пропонує величезну бібліотеку публічних образів, які підтримуються як спільнотою, так і комерційними провайдерами. *Docker Hub* також дозволяє користувачам створювати та зберігати приватні репозиторії.

Сайт *Docker Hub*, який доступний за адресою *hub.docker.com*, є місцем, де ви можете знайти та завантажити готові до використання образи для різноманітних застосунків та сервісів. Ці образи можна використовувати як основу для своїх власних контейнерів або використовувати без змін.

Наприклад, для швидкого запуску веб-сервера на базі *nginx*, можна скористатися офіційним образом *Nginx*, знайденим на *Docker Hub*, та запустити його за допомогою відповідних команд *Docker Pull* та *Docker Run*, як показано в лістингу 3.2.

Лістинг 3.2. Процес встановлення та запуску офіційного образа *nginx*

```
docker pull nginx
docker run -d -p 80:80 nginx
```

Ці команди завантажать останню версію образу *Nginx* з *Docker Hub* та запустять новий контейнер з веб-сервером, який буде слухати порт 80. Таким чином, *Docker Hub* є незамінним ресурсом для розробників та системних адміністраторів, які використовують контейнеризацію для розгортання та управління своїми застосунками.

Основні команди *Docker*, які мають значення в контексті розробки та управління контейнеризованими середовищами, включають наступні:

1. Перевірка активних контейнерів:

Запуск команди *docker ps* в командному рядку або терміналі відображає список усіх запущених контейнерів. Це дозволяє легко ідентифікувати працюючі контейнери, включно з такими, як nginx.

2. Перегляд усіх контейнерів (активних та зупинених):

Виконання команди *docker ps –a* дозволяє оглянути усі контейнери, включно з тими, що були зупинені.

3. Запуск контейнера:

Команда *docker run hello-world* створює та запускає новий контейнер на базі образу "*hello-world*", що є важливим для тестування та перевірки встановлення *Docker*.

4. Перегляд локально доступних образів:

Запуск *Docker Images* відображає список усіх образів, доступних локально, що допомагає управлінню ресурсами та вибору потрібних образів для контейнерів.

5. Створення образу з *Dockerfile*:

Команда *docker build -t my-app .* створює новий образ під назвою "my-app" на основі інструкцій, вказаних у *Dockerfile*, що знаходиться у поточній директорії.

6. Отримання логів контейнера:

Використання *docker logs [container_id]* надає доступ до логів вказаного контейнера, де *[container_id]* - це унікальний ідентифікатор контейнера.

7. Зупинка контейнера:

Команда *docker stop [container_id]* призупиняє роботу активного контейнера.

8. Видалення контейнера:

Використання *docker rm [container_id]* дозволяє видалити контейнер з системи.

9. Видалення образу:

Команда *docker rmi [image_id]* використовується для видалення образу за допомогою його ідентифікатора.

Ці команди відіграють ключову роль у щоденній роботі з *Docker*, дозволяючи ефективно керувати контейнерами та образами.

3.2. Встановлення та налаштування Prometheus

В рамках моєї роботи я встановив та налаштував Prometheus, який є відкритим інструментом системного моніторингу та сповіщення, широко використовуваним у спільноті *DevOps*. Етапи встановлення Prometheus:

1. Спочатку я завітав на офіційний сайт *Prometheus* та завантажив останню версію програми для *Windows*.

2. Використовуючи утиліту для розпакування архіву, я розмістив файл *Prometheus* у вказаній директорії.

3. Через *PowerShell*, запущений у папці з *Prometheus*, я ініціював запуск програми, використовуючи команду *.\prometheus.exe*.

Налаштування *Prometheus*:

1. Я ознайомився з файлом конфігурації prometheus.yml, який визначає основні параметри збору даних.

2. У файлі *prometheus.yml* я налаштував цілі (*targets*), наприклад, мій локальний сервер і *Docker* контейнери, які я хотів моніторити. Налаштування *prometheus.yml* можна переглянути у Додатку А.

3. Зміни у файлі конфігурації вимагали перезапуску *Prometheus* для їх активації.

4. Веб-інтерфейс: Я перевірив функціональність *Prometheus*, відкривши його веб-інтерфейс за адресою http://localhost:9090. (рис 3.1).

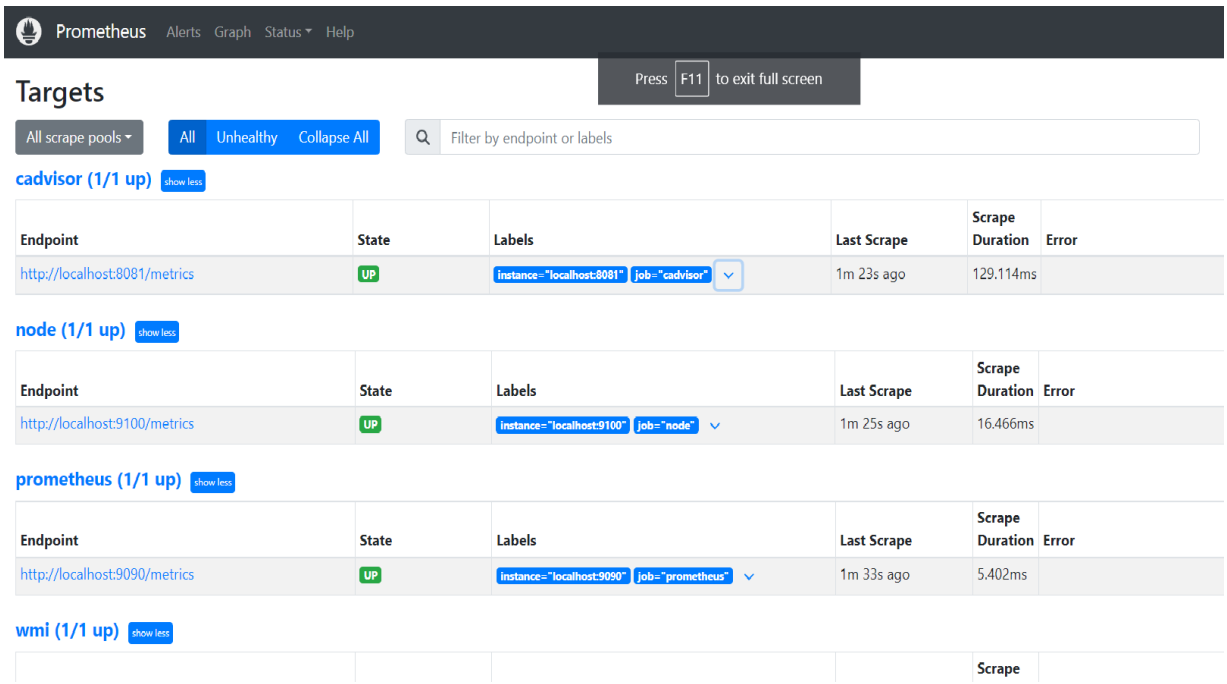

Рис.3.1 - *Prometheus* із встановленими програмами для збору метрик

5. Далі я експериментував із створенням запитів у *Prometheus* для отримання різноманітних метрик з моїх систем. Я також використовував *Prometheus* для зберігання історії метрик на локальному комп'ютері. Для цього я додав рядок, як показано в лістингу 3.3. Ця команда запускає *Prometheus* з параметром --storage.tsdb.retention.time=1y, який вказує, що дані метрик мають зберігатися протягом одного року.

Лістинг 3.3. Додатковий параметр (час збереження метрик)

```
C:\Prometheus\prometheus-2.48.0.windows-amd64\prometheus.exe --
storage.tsdb.retention.time=1y
```

Ця команда додається до ярлика програми запускає *Prometheus* з параметром --storage.tsdb.retention.time=1y, який вказує, що дані метрик мають зберігатися протягом одного року.

Prometheus виявився надзвичайно цікавим інструментом для моніторингу. Його гнучкість у зборі метрик, візуалізації даних та налаштуванні сповіщень значно покращили моє розуміння систем моніторингу та допомогли з ефективним управлінням моїми проектами.

3.3. Встановлення та налаштування *cAdvisor, Node, WMI*

Під час моєї роботи з системами моніторингу я використовував *cAdvisor* (*Container Advisor*) від *Google* як інструмент для аналізу ресурсів та продуктивності контейнерів, особливо у середовищах *Docker*.

Я здійснив запуск *cAdvisor* через *Docker*, для чого спочатку переконався у наявності *Docker* на моєму комп'ютері. Потім я виконав команду в терміналі, як показано в лістингу 3.4. Ця команда ініціювала завантаження та запуск останньої версії *cAdvisor*, а також ця команда включала ряд параметрів, зокрема вказівки на використання певних томів та налаштування мережі, щоб забезпечити доступ до веб-інтерфейсу cAdvisor. Оскільки порт 8080 на моєму комп'ютері був уже зайнятий, я змінив налаштування порту на 8081. Це дозволило мені успішно запустити cAdvisor та доступити до його веб-інтерфейсу за адресою

http://localhost:8081. У веб-інтерфейсі я міг спостерігати за метриками моїх контейнерів, що давало мені цінну інформацію про їх стан та продуктивність.

Лістинг 3.4. Запуск cAdvisor

```
docker run \
  --volume=/:/rootfs:ro \
  --volume=/var/run:/var/run:rw \
  --volume=/sys:/sys:ro \
  --volume=/var/lib/docker/:/var/lib/docker:ro \
  --publish=8081:8080 \
  --detach=true \
  --name=cadvisor \
  google/cadvisor:latest
```

Відкрийте веб-браузер і перейдіть за адресою http://localhost:8081. Ви повинні побачити веб-інтерфейс cAdvisor, який відображає метрики ваших контейнерів (рис. 3.2).

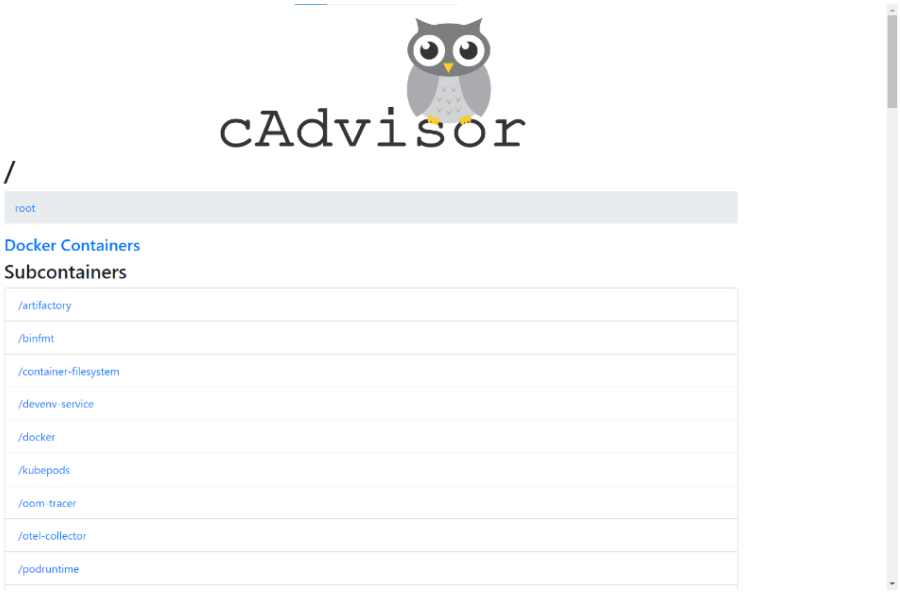

Рис. 3.2 - cAdvisor

При виконанні мого проекту я встановив *Node Exporter* на *Windows* в якості *Docker* контейнера, щоб моніторити системні метрики. Процес включав кілька кроків:

1. Я відкрив *PowerShell* та переконався, що *Docker* був встановлений і активний. Потім я виконав команду для запуску *Node Exporter* у *Docker* контейнері, як показано в лістингу 3.5. Ця команда запустила контейнер у

фоновому режимі, перенаправила порт 9100 з контейнера на мій хост-комп'ютер, встановила ім'я контейнера як node_exporter, забезпечила автоматичний перезапуск контейнера при необхідності та використала офіційний образ *Node Exporter* від *Prometheus*.

Лістинг 3.5. Запуск *Node Exporter* у *Docker* контейнері

```
docker run -d -p 9100:9100 --name=node_exporter --restart=always
prom/node-exporter
```

Ця команда виконає наступне:

a) -d: Запуск контейнера в фоновому режимі (detached mode);

b) -p 9100:9100: Перенаправлення порту 9100 з контейнера на хост-машину;

c) --name=node_exporter: Назва контейнера;

d) --restart=always: Автоматичний перезапуск контейнера при перезавантаженні системи або при виході контейнера з ладу;

e) prom/node-exporter: Використання офіційного образу *Node Exporter* від *Prometheus*.

2. Після виконання команди, я перевірив статус запущеного контейнера, використовуючи *docker ps*, щоб переконатися, що *Node Exporter* працює правильно. Також я оновив файл конфігурації prometheus.yml, додавши *Node Exporter* як ціль для моніторингу. Налаштування prometheus.yml можна переглянути у Додатку А.

3. Я відкрив веб-браузер і перейшов за адресою http://localhost:9100/metrics для перегляду сторінки з метриками, які збирав *Node Exporter*. Це дало мені можливість використовувати ці метрики для моніторингу стану моєї системи у *Prometheus*.

Встановлення *Node Exporter* у *Docker* контейнері на *Windows* виявилося простим та ефективним способом моніторингу системних метрик. Цей метод забезпечив легку інтеграцію збору метрик з системами моніторингу та додав гнучкості у використанні різноманітних інструментів для аналізу даних.

Після завершення процесу встановлення *Node Exporter*, я перейшов до наступного важливого етапу своєї роботи – установки *WMI* (*Windows Management Instrumentation*) Exporter. Цей крок був необхідним для розширення можливостей моніторингу системних параметрів *Windows* у моєму проекті.

Я відкрив *PowerShell* та перевірив, що *Docker* був встановлений і активний на моєму комп'ютері. Далі я виконав команду для запуску *WMI Exporter* у *Docker* контейнері, як показано в лістингу 3.6.

Лістинг 3.6. Запуск *WMI Exporter* у *Docker* контейнері

```
docker run -d -p 9182:9182 --name=wmi_exporter prom/wmi-exporter:latest
```

Ця команда виконає наступне:

a) -d: Запуск контейнера в фоновому режимі;

b) -p 9182:9182: Перенаправлення порту 9182 з контейнера на хост-машину;

c) --name=wmi_exporter: Назва контейнера;

d) prom/wmi-exporter:latest: Використання останньої версії образу *WMI Exporter*.

Я оновив файл конфігурації prometheus.yml, додавши *WMI Exporter* як ціль для моніторингу. Це забезпечило інтеграцію *Prometheus* з *WMI Exporter*. налаштування prometheus.yml можна переглянути у Додатку А.

Щоб перевірити, що *WMI Exporter* працює належним чином, я відкрив веб-браузер і перейшов за адресою http://localhost:9182/metrics. На цій сторінці я побачив метрики, які збирав *WMI Exporter*.

3.4. Встановлення та налаштування Grafana

У рамках своєї роботи я установив та налаштував *Grafana* на моєму локальному комп'ютері під управлінням *Windows*. *Grafana*, як відомий інструмент для візуалізації даних, стала ключовою частиною мого проекту для

аналізу метрик, зібраних з різних джерел, включаючи *Prometheus*. Процес включав кілька кроків:

1. Я відвідав офіційну сторінку завантаження *Grafana* та завантажив потрібну версію для *Windows*. Після завантаження файлу я запустив інсталяційний процес, слідуючи зазначеним інструкціям, для встановлення *Grafana*.
2. Після завершення встановлення, я запустив *Grafana* через ярлик на робочому столі. Щоб перевірити, що *Grafana* працює, я відкрив веб-браузер і перейшов за адресою http://localhost:3000, де зазвичай розташовується веб-інтерфейс *Grafana*.
3. Я увійшов у систему за стандартними обліковими даними (ім'я користувача: admin, пароль: admin) та змінив пароль.
4. Перейшов у меню "*Create*" та вибрав опцію "*Import*". Після відвідування сайту *Grafana Dashboards*, я знайшов і скопіював ID необхідних панелей, зокрема "*Windows System Overview*" та "*Docker and Host Monitoring*". Ці *ID* я вставив у поле "*Import via grafana.com*" у *Grafana*. Після імпортування панелей, я налаштував їх, вказавши *Prometheus* як джерело даних. Застосував необхідні налаштування та зберіг панелі.
5. Я активно використовував панель "*Windows System Overview*" для візуалізації ключових метрик моєї *Windows* системи. А панель "*Docker and Host Monitoring*" для візуалізації ключових метрик встановлених та працюючих контейнерів у *Docer*. Це допомогло мені краще зрозуміти стан моєї системи та спростити процес моніторингу.
6. Крім технічного аспекту, я також зосередив увагу на оптимізації використання ресурсів під час моніторингу. Це означало вибір оптимальних інтервалів збору даних, налаштування порогів сповіщень та ефективне використання візуалізацій для швидкого виявлення та вирішення проблем у системі. Особлива увага була приділена мінімізації навантаження на систему, що дозволило підтримувати

високу продуктивність та стабільність роботи як контейнеризованих додатків, так і всієї IT-інфраструктури."

Використовуємо панель "*Windows System Overview*" для візуалізації ключових метрик нашої *Windows* системи (рис. 3.3).

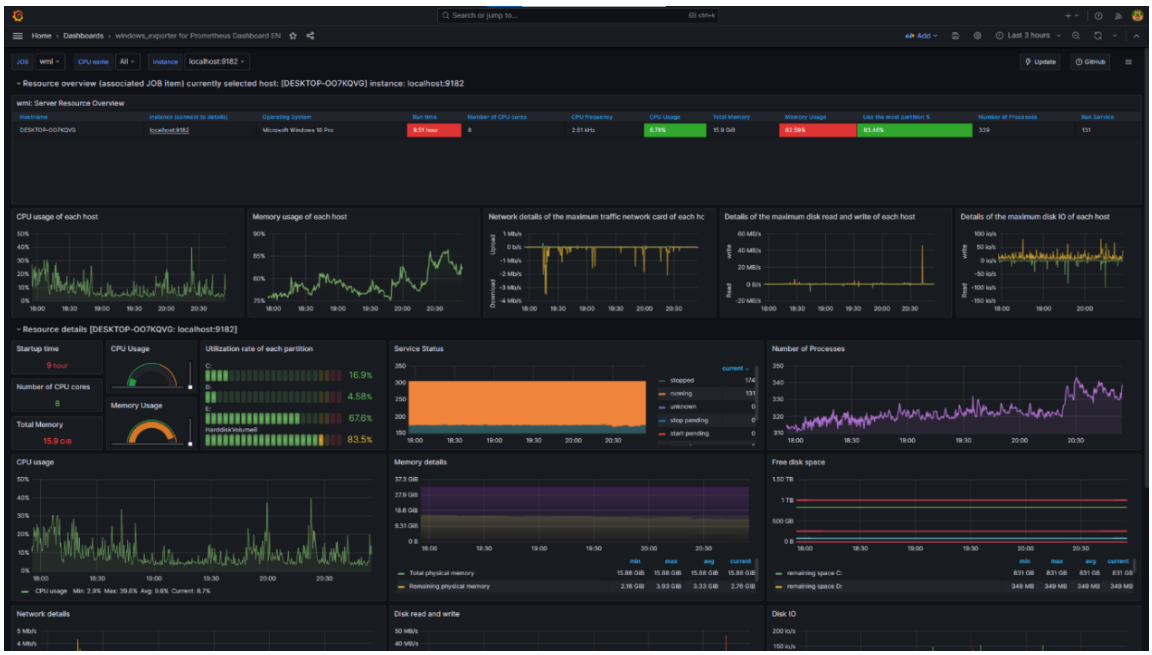

Рис. 3.3 - Панель "Windows System Overview"

Використовуємо панель "Docker and Host Monitoring" для візуалізації ключових метрик Docker контейнерів та хосту (рис. 3.4).

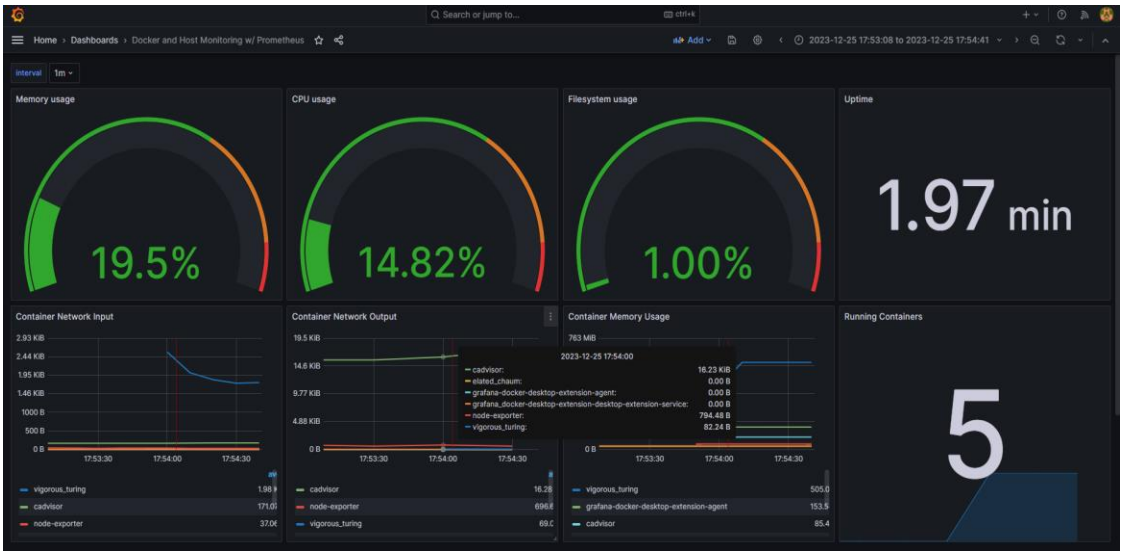

Рис. 3.4 - Панель "Windows System Overview"

Висновки до третього розділу

У третьому розділі нашої роботи ми розглянули ключові елементи системи моніторингу контейнерізації:

1. *Docker*: Ефективний інструмент для створення та управління контейнерами, що забезпечує ізоляцію та портативність.

2. *Docker Hub*: Важливе онлайн сховище для зберігання та обміну *Docker* образами.

3. Основні команди *Docker*: Набір команд для управління контейнерами та образами, необхідний для повсякденної роботи з *Docker*.

4. *Prometheus*: Могутній інструмент для збору та аналізу метрик, який інтегрується з багатьма системами.

5. *cAdvisor, Node Exporter, WMI Exporter*: Інструменти для детального моніторингу використання ресурсів в контейнерах та хост-системах.

6. *Grafana*: Платформа для візуалізації метрик, що дозволяє легко відстежувати стан систем.

Ці інструменти разом формують повноцінну систему моніторингу, яка дозволяє нам ефективно відслідковувати та аналізувати стан наших контейнеризованих середовищ.

РОЗДІЛ 4
ЄКОНОМІКА ПРОЕКТУ

4.1. Основні системи моніторингу та великі компанії, які їх використовують

В даному розділі ми розглядаємо основні інструменти моніторингу, які використовуються у сучасних інформаційних системах, їхні характеристики та потенційний вплив на вартість та продуктивність проекту. Ось перелік цих систем:

Таблиця 4.1

Основні системи моніторингу

Назва Інструменту	Переваги	Недоліки
Prometheus	Ефективний збір часових рядів, гнучкі оповіщення та PromQL	Складність налаштування
Grafana	Візуально привабливий, підтримка багатьох джерел даних	Потребує інтеграції з іншими інструментами
Docker	Легкість у використанні, широка підтримка	Менші функції безпеки, може бути ресурсоємним
Zabbix	Відкритий код, широкі можливості моніторингу	Складність налаштування
Nagios	Гнучкість конфігурації, сильна спільнота	Застарілий інтерфейс
Datadog	Інтуїтивний UI, автоматизоване виявлення сервісів	Висока вартість
New Relic	Поглиблений моніторинг додатків, інтеграція APM	Висока вартість
Splunk	Потужний аналіз логів, гнучка налаштовуваність даних	Висока вартість
Sysdig	Підтримка безпеки контейнерів, деталізований моніторинг	Складність у використанні
ELK Stack	Відкритий код, гнучка обробка даних	Ресурсоємність
cAdvisor	Легке впровадження, відстеження ресурсів контей-в	Обмежена функціональність

Ця таблиця надає детальний огляд основних систем моніторингу, які використовуються для контейнерів, з акцентом на їх ключові переваги, та недоліки.

Таблиця 4.2

Компанії, які використовують системи контейнеризації

Назва компанії	Переваги використання систем контейнеризації	Недоліки використання систем контейнеризації
Google Cloud	Інноваційні рішення, велика інфраструктура	Висока вартість, комплексність сервісів
Amazon Web Services	Широкий спектр сервісів, глобальне покриття	Складність управління, прив'язка до платформи
Microsoft Azure	Інтеграція з іншими продуктами Microsoft, підтримка гібридних рішень	Вартість, складність конфігурації
IBM Cloud	Фокус на AI та машинне навчання, безпека даних	Менший вибір сервісів, комплексність інтерфейсу
Oracle Cloud	Спеціалізація на базах даних, висока продуктивність	Обмежені функціональні можливості, вартість
Alibaba Cloud	Оптимізовано для азіатського ринку, конкурентні ціни	Обмеження за межами Азії, мовний бар'єр
DigitalOcean	Простота використання, центрованість на розробниках	Обмежена інфраструктура, функціональні обмеження
Linode	Доступні ціни, легкість управління	Обмежений набір сервісів, менший обсяг ресурсів
Rackspace	Підтримка та управління клієнтами, гнучкість	Високі ціни, орієнтація на великий бізнес
VMware	Інтеграція з існуючою інфраструктурою, розширені можливості безпеки	Висока вартість рішень, складність у використанні

Ця таблиця відображає як переваги, так і можливі недоліки використання систем контейнеризації великими технологічними компаніями. Вона може бути корисною для аналізу різних підходів та рішень, які застосовуються у цій сфері.

4.2. План Стартап-проекту "Дослідження стану застосування у системах контейнеризації"

План проекту:

1. Визначення проекту та його цілей
2. Огляд ринку та потреб у контейнеризації.
3. Визначення ключових цілей проекту.
4. Аналіз цільової аудиторії
5. Визначення основних користувачів та їх потреб.
6. Створення персон користувачів.
7. Дослідження ринку
8. Аналіз конкурентів та їх стратегій.
9. Вивчення трендів у галузі контейнеризації.
10. Розробка продукту
11. План розробки та тестування.
12. Визначення функціональності та інтерфейсу продукту.
13. Технологічний стек
14. Вибір технологій для розробки продукту.
15. План інтеграції та розгортання.
16. Маркетингова стратегія
17. Розробка бренду та позиціонування.
18. План маркетингових кампаній та просування.
19. Фінансовий план
20. Розробка бюджету проекту.
21. Пошук джерел фінансування (інвестори, гранти).
22. Управління ризиками
23. Ідентифікація потенційних ризиків.
24. Розробка плану мінімізації ризиків.
25. План розвитку та масштабування

26. Плани на майбутнє.

27. Запуск проекту

28. План запуску продукту.

29. Стратегія виходу на ринок.

30. Оцінка та аналіз ефективності

31. Постійне вдосконалення на основі зворотного зв'язку.

Ключові елементи успіху:

1. Інноваційний підхід**:** Розробка унікального продукту, який вирізняється на ринку.

2. Залучення професіоналів з відповідним досвідом у галузі.

3. Розуміння потреб цільової аудиторії.

4. Гнучкість у підході до змін на ринку та у вимогах користувачів.

4.3. Організаційно - економічне та маркетингове обґрунтування проекту

У даній роботі досліджено контейнерні технології, моніторинг метрик комп'ютера, що використовує ці технології, зберігання даних і виведення графічного результату роботи контейнерної платформи *Docker*, досліджено хмарні служби. Як результат було запущено службу *Docker*. Було розгорнуто служби *Prometheus* і *Graphana*. І за допомогою систем збору інформації *Docker, Node* та *Prometheus* програма *Graphana* відображає інформацію по комп'ютеру на підставі 11 заздалегідь встановлених метрик.

Для створення робочої моделі моніторингу було проаналізовано основні проблеми предметної області, проведено аналіз наявних аналогів, проведено аналіз та обґрунтування вибору засобів реалізації, було досліджено наявні моделі. Мій проект має характер дослідження. Існуючі системи дуже громіздкі та дорогі. Моє компонування наявних програм дасть змогу швидко та легко налагодити моніторинг *Docker* систем.

Створено програмний пакет, що допоможе відстежувати параметри та логи комп'ютера, сервера зі встановленою системою контейнеризації *Docker*,

Prometeus та *Graphana*, а також зберігати та аналізувати всю отриману інформацію.

Таблиця 4.3

Класифікаційна оцінка проекту

клас	монопроект
тип	змішаний
вид	комбінований
тривалість	короткостроковий
за степенем складності	проект середньої складності
рівень	галузевий

Мета - дослідити системи контейнеризації, і зосередитися на системі *Docker*, розібратися, налагодити та зробити доступним алгоритм моніторингу комп'ютерних систем за ключовими метриками.

Результат - розроблений алгоритм з розгортання програмних продуктів та інструкція з їх використання. І рекомендації щодо подальшого поліпшення.

Етапи виконання розділів кваліфікаційної роботи магістра:

– постановка технічного завдання: В цьому розділі представлені: аналіз існуючих систем контейнеризації та хмарних технологій, методи їх навчання; порівняння характеристик існуючих моделей та методів; вимоги до розробки програмного продукту; постановка задачі.

Термін виконання – 15 днів;

– розробка робочого проекту: розробка логічної моделі програми, шляхом складання алгоритму. На стадії робочого проекту створюється вся необхідна документація об’єкта (специфікації, пояснювальна записка, інструкції та ін.).

Термін виконання – 5 днів;

– техніко-економічна частина: проведення розрахунків собівартості та обґрунтування економічної доцільності впровадження даного програмного продукту.

Термін виконання – 5 днів;

У кваліфікаційній роботі магістра представлений наступний склад робіт:

– технічне завдання. Термін виконання – 7 днів;

– розробка і побудова робочої системи для збору метрик комп'ютера. підбір і налаштування програм Термін виконання – 15 днів;

– розробка робочого проекту. Термін виконання –15 днів;

– впровадження проекту. Термін виконання – 7 днів.

За структуру розроблювального проекту прийнята структура, орієнтована на результати проекту. Така структура заснована на побудові мережного графіка.

Таблиця 4.4

Склад робіт по життєвому циклу проекту

№ код роботи	Назва роботи	Т, дні
1-2	Збір та аналіз даних, існуючих систем	5
2-3	Встановлення потреби в результатах	5
2-4	Затвердження концепцій	2
3-5	Визначення цілей, мотивів та вимог замовника та власника	5
4-5	Планування предметної області та інших елементів проекту	3
5-6	Розробка та затвердження зведеного плану	4
5-7	Організація виконання робіт	5
6-8	Інформаційний контроль за виконанням робіт	3
7-8	Детальне моделювання	7
8-9	Керівництво і координація робіт, корегування основних показників проекту	15
9-10	Підтвердження закінчення робіт	2
9-11	Експлуатаційні випробування остаточного продукту проекту	3
10-12	Підготовка звітів	4

11-12	Підготовка кадрів для експлуатації програмного продукту	1
12-13	Оцінка результатів проекту	1
13-14	Підготовка керівництва з користування прогр. продукту	2

За складом робіт складений мережевий графік (рис. 4.2).

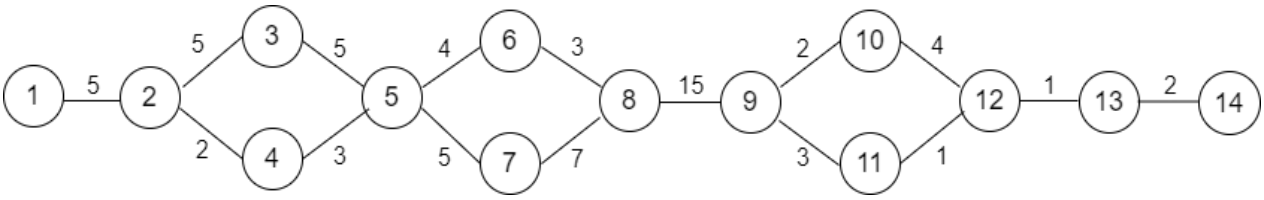

Рис. 4.2 – Мережевий графік проекту

Надалі проведемо розрахунок параметрів мережевого графіку. Для цього визначимо: ранній строк здійснення роботи (T_i); тривалість роботи (T_{ij}); ранній строк здійснення події (T_i); раннє закінчення робіт (T_{po}), пізній строк здійснення події (T_j); пізнє закінчення робіт ($T_{по}$), повний резерв часу роботи (R_j); вільний резерв часу роботи (R_c).

Таблиця 4.5

Розрахунок параметрів мережного графіку

Попередня робота	Фактична робота	T_{ij}	$T_{рн}$	$T_{ро}$	$T_{пн}$	$T_{по}$	R_c	$R_п$	R_j
1	2	5	0	5	0	5	0	0	0
2	3	5	5	10	5	10	0	0	0
2	4	2	5	7	5	7	10	10	10
3	5	5	10	15	10	15	0	0	0
4	5	3	7	10	7	10	10	10	10
5	6	4	15	19	15	19	7	7	7
5	7	5	15	20	15	20	0	0	0
6	8	3	19	22	19	22	7	7	7
7	8	7	22	29	22	29	0	1	0
8	9	15	29	44	29	44	0	0	0
9	10	2	44	46	44	46	4	4	4

9	11	3	44	47	44	47	0	0	0
10	12	4	46	50	46	50	4	4	4
11	12	1	47	48	47	48	0	0	0
12	13	1	48	49	48	49	0	0	0
13	14	2	49	51	49	51	0	0	0

Тривалість критичного шляху 51 день, не перевищує часу на

проектування (51 день), тому оптимізувати мережеву модель немає необхідності. В умовах даного проекту склад учасників буде наступним:

ініціатор (автор ідеї проекту Сиренко О. І.);

замовник-інвестор (майбутній власник);

керівник проекту (Сиренко О. І.);

споживач (тому що проект створюється в тісному співробітництві з побажаннями споживача).

Маркетингове обґрунтування проекту:

1. Для створення алгоритму програмного продукту були проаналізовані основні проблеми предметної області, проведений аналіз існуючих аналогів, проведено аналіз та обґрунтування обрання засобів реалізації, були досліджені моделі та методи проектування та застосування подібних систем.
2. Створено програмний пакет моніторингу, який зможе підвищити ефективність реалізації поставлених алгоритмів, певних задач та роботи системи контейнерізації в цілому, прискорити швидкість, здешевити і зробити доступним для використання.
3. Очікувані конкурентні переваги. Даний продукт не дивлячись на деякі недоліки, являє собою простий і зручний у застосуванні інструмент, проста реалізація функцій, ефективність, що має забезпечити його затребуваність та актуальність для невеликих комп'ютерних систем, стартапів і як приклад для навчання.

4.4. Розрахунки економічної ефективності впровадження програмного продукту

Визначення трудомісткості розробки програмного продукту (ПП). Тривалість розробки ПП залежить від обсягу ІС, трудомісткості її розробки, кваліфікації кадрів, а також планових термінів, що диктуються умовами ринку. У якості вихідних даних для визначення трудомісткості розробки ІС визначається обсяг програмних засобів в тисячах умовних машинних команд програми-аналога. Вибравши аналог програмного засобу (ПЗ), що містить V_0 в умовних машинних командах. У даному проекту розробляється новий програмний продукт, який відповідає аналогу ПЗ оптимізаційних розрахунків с $V_0 = 1500$ умовних машинних команд із трудомісткістю $T_р = 131$ чол/год.

Трудомісткість розробки ІС повинна включати розробку наступних етапів: технічного завдання – ТЗ; технічного проекту – ТП; робочого проекту – РП; впровадження – ВП.

Трудомісткість розроблювального ІС визначається на кожному етапу окремо на підставі трудомісткості аналога, з урахуванням складності розробки, ступеня новизни та ступеня використання в розробці стандартних модулів на підставі формул 4.1–4.4:

$$T_{тз} = T_р * L_1 * К_н \quad (4.1)$$

$$T_{тп} = T_р * L_2 * К_н \quad (4.2)$$

$$T_{рп} = T_р * L_3 * К_н * К_т \quad (4.3)$$

$$T_{вн} = T_р * L_4 * К_н \quad (4.4)$$

$T_р$ – укрупнення норма часу на розробку аналога ПЗ, чол/год, що коректується поправочним коефіцієнтом, що враховує умови розробки ПЗ, тобто в умовах комп'ютера, $К_н=0,7$.

У даному проекті $T_р=131*0,7=91.5$ чол/год

L_j – питома вага і-го етапу розробки. У даному проекті залежно від ступеня новизни проекту (В): L1= 0,12; L2= 0,11; L3= 0,61; L4= 0,16.

$К_н$ – поправочний коефіцієнт, що враховує ступінь новизни, у нашому випадку $К_н$= 0,7.

$К_т$ – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм, Кт=0,6.

При розрахунках прийняті наступні об'єми розробленої документації по етапах проекту:

$N_{тз}$ = 4 – кількість сторінок технічного завдання;

$N_{тп}$ = 15 – кількість сторінок технічного проекту;

$N_{рп}$ = 27,5 – кількість сторінок робочого проекту;

$N_{інстр}$=5 – кількість сторінок інструкції по налагодженню та впровадженню;

$N_{пр}$ = 42,5 – кількість сторінок пояснювальної записки.

Розрахунок трудомісткості розробки ІС:

1. Технічне завдання

$Т_{тз} = Т_р * L_1 * К_н = 91,5 * 0,12 * 0,7 = 7,69$ чол/год

$Т_{кк} = 0,7 * N_{тз} = 0,7 * 4 = 2,8$ чол/год

Тнк=0,15*Nтз=0,15*4=0,6 чол/год

2. Розробка технічного проекту (алгоритму й блок-схеми)

$Т_{тз}= Т_р * L_2 * К_н = 91,5*0,11*0,7 = 7,045$ чол/год

$Т_{кк} = 0,7* N_{тп} = 0,7 * 15 = 10,5$ чол/год

$Т_{нк} = 0,15* N_{тп} = 0,15*15 = 2,25$ чол/год

3. Розробка робочого процесу

$Т_{тз} = Т_р * L_3 * К_н * К_т = 91,5*0,61*0,7*0,6 = 23,44$ чол/год

$Т_{кк} = 0,7* N_{рп} = 0,7 * 27,5 = 19,25$ чол/год

$Т_{нк} = 0,15* N_{рп} = 0,15 * 27,5 = 4,125$ чол/год

4. Налагодження і впровадження

$Т_{тз} = Т_р * L_4 * К_н = 91,5 * 0,16*0,7 = 10,25$ чол/год

$Т_{кк} = 0,7* N_{інст} = 0,7 * 5 = 3,5$ чол/год

$Т_{нк} = 0,15 * N_{інст} = 0,15 * 5 = 0,75$ чол/год

5. Пояснювальна записка

$T_{тз}$ = 1,5 * $N_{пз}$ = 1,5 * 42,5= 63,75 чол/год

$T_{кк}$ = 0,7 * $N_{пз}$ = 0,7 * 42,5 = 29,75 чол/год

$T_{нк}$ = 0,15* $N_{пз}$ = 0,15*42,5 = 6,375 чол/год

Всього: $T_р$ = 7,69 + 7,045+23,44+10,25+63,75 = 112,175 чол/год

$T_{кк}$ = 2,8+10,5+19,25+3,5+29,75=65,8 чол/год

$T_{нк}$ = 0,6+2,25+4,125+0,75+6,375=14,1 чол/год

$T_{общ}$ = 112,18+65,8+14,1 = 192,08 чол/год

Тривалість розробки ІС у днях визначається за формулою 4.5:

$$T_{ПП} = \frac{\sum T_{ij}}{8 * 0.73} \text{ дн.}, \quad (4.5)$$

де ΣT_{ij} – сумарна тривалість розробки, год;

8 – тривалість робочого дня (коефіцієнт переказу в робочі дні), ч;

0,73 – коефіцієнт переведення в календарні дні;

T_{ij} – трудомісткість j-го виду робіт по і-му етапу.

$T_{пп}$ = 192,08 / 5,84 = 32,885 дн.

Порівнюючи отримані результати з розрахунками тривалості розробки ІС за допомогою мережевої моделі, вибираємо для подальших розрахунків останнє значення, так як воно, на погляд, ближче до фактичної тривалості розробки ІС.

Визначення вартості програмного продукту. Ціна визначається за формулою:

$$Ц = К * Вндр + П_р, \quad (4.6)$$

де:

Вндр – витрати на розробку програмної продукції (кошторисна собівартість);

К = 1,1;

$П_р$ – нормативний прибуток, розраховується за формулою:

$П_р$ = С * $Р_н$ / 100,

де:

$Р_н$ – норматив рентабельності, % $Р_н$ = 20 %;

$С_м$ – матеріальні витрати.

Вндр визначаються на підставі складання кошторису витрат на проведення НДР у таблиці 4.10.

При визначенні витрат на матеріали враховують: вартість сировини та матеріалів для проведення досліджень з урахуванням додаткових накладних витрат (витрат на транспорт, комісійних зборів тощо), вартість канцелярських матеріалів (паперів тощо), вартість інших матеріалів (табл. 4.6)

Таблиця 4.6

Витрати на матеріали

Найменування матеріалів	Ціна, грн/од	Кількість	Сума, грн.
Папір	200	1	200,0
Ручки	11	2	22,0
Інформаційні носії	160	1	160,0
Разом			388,0

Витрати по заробітній платі визначаються як сума заробітної плати усіх учасників НДР. Орієнтовний склад учасників, ступінь їх участі у НДР та заробітна плата наведені у таблиці 4.7.

Таблиця 4.7

Орієнтовний склад учасників НДР, їх заробітна плата та ступінь участі

Учасник НДР	Місячна заробітна плата, грн.	Тривалість роботи, міс	Ступінь участі, %
Студент-дослідник	1875	6	90
Науковий керівник роботи	8900	4	10

Основна заробітна плата виконавця з урахуванням окладу і часу.

Взп = 8900*32,89*0,1 / 22 + 1875*32,89*0,9 / 22 = 3853 грн

$З_і$ – середньомісячний оклад

$Д_р$ – середня кількість робочих днів (Др = 22)

τ_i - трудомісткість робіт (τ_i=36)

$К_0$ – коефіцієнт обліку окладу керівників і консультантів проекту ($К_0$ = 0,1)

Додаткова заробітна плата 10 % від основної заробітної плати.

Взд = 3853*0,1 = 385,3 грн.

Єдиний соціальний внесок беруть у розмірі 22 % від величини заробітної плати.

ЄСВ =0,22* (3853+385,3) = 932,42 грн.

Амортизаційні відрахування беруть від вартості основних виробничих фондів за встановленими нормативами до кожної групи фондів, які використовують при проведенні НДР (основного та додаткового обладнання, комп'ютерної техніки, інших фондів, крім приміщення). Амортизаційні відрахування необхідно розраховувати, виходячи з терміну їх використання.

Таблиця 4.8

Розрахунок сум амортизаційних відрахувань за відповідний рік

Група ОЗ	Елементи основних засобів	Первісна вартість, грн.	Норми АВ, %	Вартість, грн.
1	Оргтехніка	9300	15	1395,0
2	Інші основні засоби	1300	6	78,0
Всього		-	-	1473,0

Витрати, пов'язані зі споживанням електроенергії при розробки програмного продукту, дорівнюють множенню вартості 1 кВт*г електроенергії та її потреби. Розрахунок надано в таблиці 4.9.

Таблиця 4.9

Розрахунок витрат на електроенергію

Напрямок використання	Потреба в е/енергії, кВт*год.	Вартість 1 кВт*год, грн.	Витрати на е/енергію, грн.
Освітлювальні та інші потреби	600	0,86	516,0

Отже, витрати на електроенергію складуть 516,0 грн**.**

Інші витрати беруть у розмірі 10 % від суми витрат по статтях 1-5.

Накладні витрати беруть у розмірі 30 % від суми витрат по статтях 1-6.

Таблиця 4.10

Кошторис витрат на проведення прикладних НДР

Найменування статей витрат	Сума витрат, грн.
1. Матеріали	388,0
2. Паливо та енергія	516,0
3. Заробітна плата (основна і додаткова)	4238,30
4. Єдиний соціальний внесок	932,42
5. Амортизаційні відрахування	1473,00
6. Інші витрати	754,77
7. Накладні витрати	2490,60
Виробнича собівартість	10793,09
Адміністративні витрати 3,5 % від виробничої собівартості	377,76
Витрати на збут – 1 % від виробничої собівартості	107,93
Всього повна собівартість	11278,78

Нормативний прибуток: $П_р$ = (11278,78-388)*0,15= 1633,62 грн.

Ціна програмного продукту складе: Ц = 1,1 × 11278,78 + 1633,62 = 14040,278 грн.

Розрахунок капітальних витрат, пов'язаних з впровадженням (вдосконаленням) ІС здійснюється за формулою:

$$К = К_п + К_{ко} + К_{во} + К_с \quad (4.7)$$

$К_п$ – довиробничі витрати;

$К_{ко}$ – вартість комп'ютерного устаткування: $К_{ко}$ = 4000 грн.;

$К_{во}$ – вартість допоміжного устаткування: $К_{во}$ = 550 грн.;

$К_с$ – вартість будівництва у зв'язку з впровадженням: $К_с$ = 0.

$К_п$ = Ц = 14040,278 грн. – перед виробничі витрати:

К=14040,278+4000+550+0=18590

Розрахунок поточних (експлуатаційних) витрат

$$С = С_{опл} + С_а + С_{ел} + С_п + С_р + С_{всп} \quad (4.8)$$

$С_{опл}$ – річний фонд основної і додаткової оплати праці персоналу;

$С_а$ – сума річних амортизаційних відрахувань від вартості основного і допоміжного устаткування;

$С_{ел}$ – вартість витрат на енергію за рік;

$С_р$ – вартість річного ремонту:

6 % $К_{ко}$ = 240 грн.;

$С_{всп}$ – річна вартість допоміжних матеріалів, пов'язаних з експлуатацією ІС:

2 % $К_{ко}$ = 80 грн.;

$С_п$ – вартість утримання приміщень:

$С_п$ = 2500 грн.

Річний фонд заробітної плати:

1. до впровадження програмного продукту дана задача вирішувалася двома співробітниками з окладом по 4800 грн/міс:

$З_{осн}^{до}$ = (2400*2)*12 = 57600 грн.

2. після впровадження програмного продукту чисельність фахівців скоротилася до одного фахівця – з захисту інформації з окладом 6000 грн.:

$З_{осн}^{після} = 3000*12 = 36000$ грн.

Фонд додаткової заробітної плати:

$З_{доп} = З_{осн} * К_{доп}$,де

$К_{доп}$ – коефіцієнт додаткової заробітної плати: визначається в розмірі $К_{доп} = 0,1$.

$З_{доп}^{до} = 57600 * 0,1 = 5760$ грн.

$З_{доп}^{після} = 36000 * 0,1 = 3600$ грн.

Єдиний соціальний внесок 22 %:

$Зєсв^{до} = (57600 + 5760) * 0,22 = 13939,20$ грн.

$Зєсв^{після} = (36000 + 3600) * 0,22 = 8712$ грн.

Загальні витрати на оплату праці:

$$С_{опл} = З_{осн} + З_{доп} + З_{єсв} \quad (4.9)$$

Разом:

$С_{опл}^{до} = 57600 + 5760+13939,20 = 77299,20$ грн.

$С_{опл}^{після} = 36000 + 3600+ 8712 = 48312$ грн.

Розрахунок амортизаційних відрахувань визначається за формулою:

$С_а = К_{кко} * Н_а / 100$

де $Н_а$ – норма амортизаційних відрахувань ($Н_а = 15$ %)

$С_а = 14040,278 * 0,15 = 2106,04$ грн.

Річна вартість споживаної електроенергії, визначається за формулою:

$$С_{ел} = М_у * Т_{ко} * Ц_э * К_и \quad (4.10)$$

$М_у$ – установлена сумарна потужність комп'ютерного устаткування

$М_у = 0,6$ кВт;

$Т_{ко}$ – річний фонд роботи ЕОМ з урахуванням часу на профілактичні огляди: $Т_{ко} = 6320$ год;

$Ц_э$ – вартість 1 кВт – години ел. енергії 1,68 грн.;

$К_и$ – коефіцієнт інтенсивного використання потужності Кп = 0,9.

$С_{ел} = 0,6 * 6320 * 1,68 * 0,9 = 5733,50$ грн.

$$\text{Разом: } C = C_{опл} + C_а + C_{ел} + C_р + C_п \quad (4.11)$$

$C^{до}$ =7299,2 + 2106,04+5733,50+192+64+2100 = 87494,74 грн.

$C^{після}$ = 48312 + 2106,04+5733,50+192+64+2100 = 58507,54 грн.

Розрахунок показників економічної ефективності проекту.

$$Э_о = Э_г – Е_н * К_п \quad (4.12)$$

$Э_г$ – річна економія на поточних витратах;

$К_п$ – передвиробничі витрати на проект: $К_п$ = 21337,49 грн.;

$Е_н$ – нормативний коефіцієнт: $Е_н$ = 0,25.)

$$Э_г = (C^{до} – C^{після}) + ΔП \quad (4.13)$$

ΔП – приріст прибутку господарчого суб'єкта та його структурного розподілу ΔП = 0

$Э_г$ = 87494,74 – 58507,54 + 0 = 28987,20 грн.

$Э_о$ = 28987,20 – 0,25 * 14040,278 = 25477,13 грн.

Коефіцієнт ефективності одноразових витрат:

$$Е = Э_г / К_п \quad (4.14)$$

Е = 57974,5 / 14040,278 = 4,13

4,13 > 0,25 проект ефективний

Строк окупності одноразових витрат проекту:

Т = 1 / 4,13 = 0,24 року

Т = 2,9 місяця

Таблиця 4.11

Техніко-економічні показники

№	Показники	Одиниця виміру	Значення показника
1	Час роботи над проектом	дні	32,89
2	Ціна ПП	грн.	14040,28
3	Капітальні витрати	грн.	18590,00

4	Поточні витрати: - до впровадження; - після впровадження.	грн./рік	87494,74 58507,54
5	Економія на поточних витратах	грн./рік	28987,20
6	Термін окупності	рік	0,24
7	Коефіцієнт економічної ефективності	-	4,13

Висновки до четвертого розділу

Результати економічних розрахунків підтверджують економічну доцільність пропонованого програмного продукту для використання в навчально-освітніх цілях і підготовки майбутніх випускників/магістрів. Знижуються експлуатаційні витрати за рахунок зростання продуктивності праці, економії на поточних витратах за рік 28987,20 грн. Термін окупності становить 2,9 місяця, що є прийнятним результатом і доводить економічну доцільність впровадження даного проекту.

РОЗДІЛ 5
ОХОРОНА ПРАЦІ

Охорона праці в системах контейнерізації (серверних кімнат, та комп'ютерних робочих місць далі назвемо комп’ютерних системи) є надзвичайно важливою, особливо у контексті забезпечення належного рівня освітленості та безпечного використання електроприладів. Враховуючи специфіку роботи у серверних комнатах, де працівники часто стикаються з обмеженим простором та нестандартними робочими умовами, питання безпеки набуває особливого значення.

В Україні регулювання освітленості та безпеки при використанні електроприладів на робочих місцях здійснюється згідно з низкою нормативних документів. Одним із ключових актів у цій галузі є "Правила улаштування електроустановок" (ПУЕ - 2017р), які визначають основні вимоги до електрообладнання, а також "Державні будівельні норми України"(ДБН-2018р), що включають норми освітленості робочих місць.

Цей розділ дипломної роботи має на меті детально розглянути вищезазначені нормативні акти, щоб забезпечити глибоке розуміння стандартів безпеки у контексті сучасних комп’ютерних систем. Зокрема, буде зосереджено увагу на аналізі специфічних вимог до освітленості, які є критичними для забезпечення безпечних та ефективних умов праці.

Нормативи освітленості:

Освітленість на робочих місцях є критичним фактором, що впливає на продуктивність та безпеку працівників. У контексті комп’ютерних систем, де простір часто обмежений, а умови роботи можуть варіюватися, забезпечення належного рівня освітленості набуває особливої важливості.

Згідно з Державними будівельними нормами України (ДБН В.2.5-28:2018), норми освітленості для робочих місць встановлюються в залежності від характеру та складності виконуваних завдань. Наприклад, для роботи з

документами та комп'ютерною технікою вимагається освітленість не менше 500 люкс, тоді як для загальних робочих зон — не менше 200 люкс.

Освітленість вимірюється в люксах і виміряти його можна двома шляхами:

1. Перший шлях інструментальний. Можна придбати побутовий або професійний спеціалізований прилад – люксметр, або ж встановити на свій смартфон додаток з функцією люксметра. Точність вимірювання побутових приладів та додатків обмежена – іноді покази різних додатків відрізняються в два рази.
2. Другий шлях вимірювання освітленості розрахунковий. На основі відомих технічних характеристик приміщення та світильників за формулами можна розрахувати величину освітленості приміщення (в люксах):

Освітленість = Світловий потік / (Площа*Коефіцієнт висоти стелі) (5.1)

Також можна розрахувати необхідну кількість ламп для певної освітленості (сумарний світловий потік в Люменах)(5.2).

Світловий потік = Норма освітленості * Площа * Коеф. вистоти стелі (5.2)

Коефіцієнт висоти стелі:

1 при висоті від 2,5 до 3

1,5 при висоті від 3 до 3,5

2 при висоті від 3,5 до 4,5

Приклад: Освітлення серверного приміщення (рис. 5.3).

Необхідно забезпечити освітлення не менше 500 люкс.

Рівень освітленості вимірюється на висоті 1 метра від рівня підлоги.

Електроживлення освітлення серверного приміщення та електроживлення телекомунікаційного обладнання, встановленого в ньому має подаватися від різних розподільних електричних щитів.

Рис.5.3 - Освітлення серверного приміщення

Світильники необхідно розміщувати на стелі. Для управління освітленням потрібно використовувати однин або декілька вимикачів і розташовувати їх поряд з дверима на висоті 1.5м від рівня підлоги. У серверному приміщенні забороняється використовувати пристрої плавного регулювання освітлення (дімери).

Приклад розрахунку. Визначимо яка величина світлового потоку потрібна для приміщення прощею 100 м2 з висотою стелі 3м:

500*100*1=5000 люмен

* норма освітленості різні і залежать від функціоналу приміщення, від того, що людина робить в цих приміщеннях і на скільки важливий зір для цієї роботи. Для серверних - 500люкс

5000:500 = 10 ламп по 500 люмен.

Для такої серверної достатньо буде використання 10 ламп по 500 люмен.

Для комп’ютерних систем, де робочі умови можуть значно відрізнятися від традиційних офісних просторів, важливо забезпечити достатнє штучне освітлення, щоб компенсувати відсутність природного світла. Тут варто звернути увагу на вибір світлодіодних джерел світла, які не тільки економічні, але й забезпечують рівномірне освітлення, мінімізуючи тіні та відблиски, що є важливим для запобігання втомі очей.

Крім того, відповідно до "Правил охорони праці при експлуатації електроустановок" необхідно регулярно перевіряти стан освітлювальних приладів, а також забезпечувати захист від прямого дотику до джерел світла та електрообладнання.

Враховуючи ці вимоги, можна зробити висновок, що адекватне освітлення в комп’ютерних системи не лише підвищує ефективність роботи, але й сприяє збереженню здоров'я та безпеки працівників.

Безпека використання електроприладів в серверних кімнатах та на комп'ютерних робочих місцях.

У контексті серверних кімнат та комп'ютерних робочих місць важливим аспектом охорони праці є забезпечення безпеки при використанні електроприладів. Це включає в себе заходи з попередження перевантаження електромережі, захисту від електричних ударів, та забезпечення стабільного електропостачання.

Правила та нормативи:

1. Згідно з "Правилами охорони праці при експлуатації електроустановок" (НПАОП 40.1-1.32-01), основними вимогами є наявність заземлення та захисного вимикання для всіх електроприладів. Це особливо важливо для серверних кімнат, де використовується велика кількість електронного обладнання.
2. Обслуговування та перевірка обладнання
3. Регулярне технічне обслуговування та перевірка стану електроприладів, відповідно до "Правил технічної експлуатації електричних установок споживачів" (НПАОП 40.1-1.21-98), допомагає запобігти несправностям та відмовам у роботі, які можуть призвести до аварійних ситуацій.

Для забезпечення електропостачання рекомендується встановити як мінімум два окремі блоки з подвійними електричними розетками. Ці блоки слід живити від різних джерел електроживлення, а розетки повинні бути розраховані на змінний струм до 16А. Додатково необхідно встановити блоки з подвійними

електричними розетками з інтервалом 1,8 метра уздовж стіни на висоті не менше ніж 0,15 метра від рівня підлоги.

Електропостачання в серверне приміщення має здійснюватися через окремий силовий кабель, бажано безпосередньо від головного розподільного щита. Якщо встановлена система резервного електроживлення, серверне приміщення має бути до неї підключене. Потрібно встановити окремий електричний розподільний щит. Дозволяється установка джерел безперебійного живлення (ДБЖ) до 100 кВА. ДБЖ потужністю понад 100 кВА мають бути встановлені у відокремленому приміщенні.

У апаратному приміщенні має бути встановлена головна телекомунікаційна заземлювальна шина, до якої повинні бути підключені заземлюючі та з'єднувальні провідники від монтажних конструкцій, телекомунікаційного обладнання, металевих кабель-каналів.

Для розподілу кабелів та організації кабельних потоків у телекомунікаційному приміщенні необхідно використовувати кабель-канали та організатори. Ці засоби мають бути надійно закріплені, витримувати вагу кабелю, забезпечувати захист та розподіл кабелів із мінімально допустимим радіусом вигину кабелю.

Кабель-канали мають бути встановлені від кабельного вводу в телекомунікаційне приміщення до телекомунікаційних шаф. Розташовані під стелею кабель-канали мають бути відкриті та доступні для подальших робіт з прокладання кабелів, шнурів або перемичок. Кабельні вводи в серверне приміщення рекомендується розміщувати поруч з дверима апаратного приміщення. Також важливо забезпечити адекватну вентиляцію та охолодження обладнання для запобігання перегріву, що може спричинити пожежі чи інші аварійні ситуації.

Висновки до п’ятого розділу

Аналізуючи нормативи охорони праці, зокрема для серверних кімнат та комп'ютерних робочих місць, я звернув увагу на важливість комплексного

підходу до безпеки. Це охоплює аспекти як адекватне освітлення, безпека використання електроприладів, ефективна вентиляція, так і регулярні перевірки технічного стану обладнання.

Важливим є дотримання норм, встановлених у таких документах, як "Правила охорони праці при експлуатації електроустановок" (НПАОП 40.1-1.32-01), "Правила технічної експлуатації електричних установок споживачів" (НПАОП 40.1-1.21-98), та Державні будівельні норми України (ДБН В.2.5-28:2018).

Забезпечення безпеки електроприладів у комп’ютерних системах потребує інтегрованого підходу, який включає в себе не тільки дотримання нормативних актів, але й регулярне технічне обслуговування та впровадження спеціальних заходів, таких як системи охолодження та УБЖ.

Для програмістів особливо важливими є заходи щодо охорони праці, пов'язані з тривалим сидінням та роботою за комп'ютером. Ергономічне робоче місце, регулярні перерви для розтяжки, вправи для спини, правило 20-20-20 для зниження напруги на очі, використання антивідблискових фільтрів, регулярні огляди у окуліста, а також заходи для зменшення напруги у кистях рук є ключовими для здоров'я та безпеки працівників.

ЗАГАЛЬНІ ВИСНОВКИ

У цій роботі я зосередився на практичному аспекті розробки системи моніторингу для контейнерізованих середовищ. Як результат моєї роботи:

1. Встановлення та налаштування *Docker*: Я успішно встановив та налаштував *Docker* на своєму комп'ютері. Це дозволило мені з легкістю створювати та керувати контейнерами, що є невід'ємною частиною моїх IT-проектів.
2. Використання *Docker Hub*: Я виявив, що *Docker Hub* є важливим ресурсом для швидкого доступу до численних готових образів, що спрощує процес розробки та тестування.
3. Освоєння основних команд *Docker*: Практичне застосування команд *Docker* було ключовим для ефективного управління моїми контейнерами та образами.
4. Встановлення та налаштування *prometheus*: Я встановив *Prometheus* як цінний інструмент для збору та аналізу метрик, який дозволяє моніторити стан моїх систем у реальному часі.
5. Використання *cAdvisor, Node Exporte*r та *WMI Exporter*: Ці інструменти дали мені можливість збирати детальні метрики, які виявилися важливими для оцінки продуктивності та ресурсів моїх систем та контейнерів.
6. Встановлення та налаштування *Grafana*: Завдяки *Grafana* я отримав зручний інструмент для візуалізації метрик, що дозволяє ефективно оцінювати та аналізувати продуктивність системи. Використання готових панелей *Grafana* значно спростило цей процес.

В цілому, я прийшов до висновку, що комплексний підхід до моніторингу в системах контейнерізації є ключовим для підтримання їх стабільності та ефективності. Мій досвід підтвердив, що інтеграція цих інструментів є необхідною для ефективного управління та моніторингу різних аспектів контейнеризованих середовищ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Docker: [Website]. Odesa, 2023. URL: https://www.docker.com/resources/what-container (viewed on: 01.08.2023).

2. Kubernetes: [Website]. Odesa, 2023. URL: https://kubernetes.io/uk/docs/home/ (viewed on: 07.08.2023).

4. GrafanaLabs: [Website]. Odesa, 2023. URL: https://grafana.com/docs/ (viewed on: 15.09.2023).

5. Poulton N. Docker Deep Dive / editor. by Independently published . UK, 2017. 249 p.

6. Burns B. , Beda J. , Kelsey H. Kubernetes: Up and Running: Dive into the Future of Infrastructure / editor. by O''Reilly Media. 2022. 326 p.

7. Poulton N. Getting Started with Docker / editor. by Packt Publishing. Oxford, UK, 2023. 115 p.. – 115c.

8. Poulton N. The Kubernetes Book: 2023 Edition (Mastering Kubernetes Book 2) / editor. by O''Reilly Media. St. Louis, Missouri, USA, 2023. 311 p.

9. Cannon J. Docker: A Project-Based Approach to Learning / editor. by Packt Publishing. Oxford, UK, 2021. 269 p.

10. Артеменко С.В., Шестопалов С.В., Сахарова С.В., Жуковецька С.Л., Нєнов О.Л. Методичні вказівки до виконання кваліфікаційних робіт для здобувачів освіти СВО «Магістр» спеціальності 123 «Комп'ютерна інженерія». Одеський національний технологічний університет, 2023р.-52с.

11. Методичні вказівки до оцінки науково-технічної ефективності розробки нової технології, нового обладнання та інших інновацій. Укладачі Басюркіна Н.Й., Свистун Т.В. Одеса: ОНТУ, 2023р. 18с.

12. Dixon J. , Kelly M. Prometheus – Up & Running: Infrastructure and Application Performance Monitoring / editor. by O''Reilly Media. St. Louis, Missouri, USA, 2023. 304 p.

13. Mccollam R. Getting Started with Grafana: Real-Time Dashboards for IT and Business Operations / editor. by O"Reilly Media. St. Louis, Missouri, USA, 2022. 336 p.

14. Lukša M. Kubernetes in Action (2nd) / editor. by Manning Publications. St. Louis, Missouri, USA, 2023. 624 p

15. Turnbull J. The Docker Book: Containerization is the New Virtualization / editor. by O"Reilly Media. St. Louis, Missouri, USA, 2019. 387p