ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «АПАРАТНО-ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ МОНІТОРИНГУ СТАНУ КОМП'ЮТЕРА»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерна інженерія
(код, найменування спеціальності)
освітньо-професійної програми «Комп'ютерна інженерія»
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Павло ТАРАСЕНКО*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач(ка) вищої освіти гр. КІД-42
Павло ТАРАСЕНКО
Ім'я, ПРІЗВИЩЕ

Керівник: Людмила АСЄЄВА
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерна інженерія

Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Тарасенко Павло Русланович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Апаратно-програмний комплекс для моніторингу стану комп'ютера»
керівник кваліфікаційної роботи: Людмила АСЄЄВА, phD (доктор філософії), доцент ,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025р. №56
2. Строк подання кваліфікаційної роботи«05» червня 2025р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, стандарти моніторингу IT-систем, протоколи передачі даних (HTTP API, SNMP), документація інструментів Glances, Docker, Node Exporter, методи візуалізації та зберігання метрик, приклади інтеграції з Telegram-ботами, підходи до аналізу навантаження на апаратні ресурси, практики побудови мінімального MVP.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Теоретичні основи побудови систем моніторингу комп'ютера
    2. Розробка архітектури апаратно-програмного комплексу
    3. Оптимізація, аналіз та перспективи розвитку
5. Перелік ілюстративного матеріалу: *презентація*
    1. Мета, об'єкт, предмет дослідження
    2. Актуальність роботи
    3. Потреба у локальному моніторингу комп'ютерних систем
    4. Загальна архітектура запропонованої системи
    5. Розгортання віртуального ізольованого середовища
    6. Алгоритми функціонування у ізольованій системі
    7. Демонстрація функціонування системи (веб-інтерфейс)
    8. Методи покращення існуючої системи
    9. Порівняння запропонованої системи зі альтернативами

10. Висновки

6. Дата видачі завдання «24» лютого 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	25.02-19.03.2025	
2	Обробка матеріалу	20.03-02.04.2025	
3	Розробка теоретичної частини	03.04-07.04.2025	
5	Розробка архітектури апаратно-програмного комплексу	08.04-27.04.2025	
6	Оптимізація, аналіз та перспективи розвитку	28.04-11.05.2025	
7	Висновки за результатами аналізу	12.05-15.05.2025	
8	Розробка презентації	16.05-26.05.2025	
9	Передзахист	27.05-29.05.2025	
10	Здача в деканат	30.05-05.06.2025	

Здобувач(ка) вищої освіти _____ Павло ТАРАСЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Людмила АСЄЄВА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 92 стор., 41 рис., 8 табл., 30 джерел.

Мета роботи – спрощення впровадження базового моніторингу апаратних ресурсів комп'ютера за рахунок використання принципів модульності, контейнеризації та відкритих кросплатформених засобів візуалізації.

Об'єкт дослідження – процес моніторингу стану апаратних компонентів комп'ютера в умовах обмежених ресурсів.

Предмет дослідження – технології збору, обробки та представлення метрик системних ресурсів з використанням відкритих інструментів.

Короткий зміст роботи: У роботі проаналізовано вимоги до мінімальної інфраструктури моніторингу комп'ютерної системи, охарактеризовано функціональні та нефункціональні вимоги до програмно-апаратного комплексу, а також представлено короткий огляд і порівняння популярних рішень: Glances, Prometheus зі Grafana, Zabbix.

Розроблено тестове середовище з використанням Glances у контейнері Docker, налагоджено Web-доступ до метрик і забезпечено базове візуальне представлення даних про навантаження на CPU, пам'ять, дискову підсистему, мережу та контейнери Docker. Наведено алгоритм обробки даних та логіку побудови MVP з можливістю подальшого розширення (сповіщення, підтримка кластерів, збереження історичних даних).

КЛЮЧОВІ СЛОВА: МОНІТОРИНГ, GLANCES, PROMETHEUS, GRAFANA, ZABBIX, WEB-ІНТЕРФЕЙС, ПСЕВДОРЕАЛЬНИЙ ЧАС, ДОКЕР, СИСТЕМНІ МЕТРИКИ, АПАРАТНИЙ СТАН, ІНТЕГРАЦІЯ, СПОВІЩЕННЯ, ЕФЕКТИВНІСТЬ, КОНТЕЙНЕРИЗАЦІЯ.

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ СИСТЕМ МОНІТОРИНГУ КОМП'ЮТЕРА 9
1.1 Поняття та класифікація систем моніторингу 9
1.2 Протоколи та технології збору телеметрії 16
1.3 Методи візуалізації та подання даних 25
1.4 Огляд готових рішень для моніторингу 33
1.5 Вимоги до апаратно-програмного комплексу моніторингу 41
РОЗДІЛ 2. РОЗРОБКА АРХІТЕКТУРИ АПАРАТНО-ПРОГРАМНОГО КОМПЛЕКСУ 51
2.1 Загальна архітектура системи моніторингу 51
2.2 Вибір та конфігурація інструментів 59
2.3 Алгоритм роботи системи моніторингу 65
2.4 Побудова мінімального MVP 70
РОЗДІЛ 3. ОПТИМІЗАЦІЯ, АНАЛІЗ ТА ПЕРСПЕКТИВИ РОЗВИТКУ 77
3.1 Методи покращення ефективності системи 77
3.2 Порівняльний аналіз із іншими системами 83
3.3 Перспективи розвитку комплексу 93
ВИСНОВКИ 99
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 100
Додаток А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 102

ВСТУП

Сучасні комп’ютерні системи — від домашніх ПК до серверів і кластерів — потребують постійного контролю за станом апаратних ресурсів. Зростання обчислювального навантаження, поява нових задач (віртуалізація, контейнери, ML) та високі вимоги до стабільності роботи зумовлюють потребу у своєчасному виявленні відхилень у роботі компонентів. Моніторинг стає критично важливим для забезпечення безперервної роботи, запобігання збоям і швидкого реагування на проблеми.

У теорії моніторинг охоплює збір, збереження, візуалізацію та аналіз метрик, таких як навантаження CPU, використання ОЗП, активність дисків, стан мережі, температури тощо. Для цього застосовуються як легковагові рішення з веб-інтерфейсом (Glances, Netdata), так і масштабовані системи рівня підприємств (Prometheus зі Grafana, Zabbix), здатні працювати з десятками вузлів і базами даних для історичного аналізу.

Кожен підхід має свої переваги та недоліки. Glances забезпечує простоту розгортання та зручність використання, але поступається у масштабованості. Натомість Prometheus чи Zabbix пропонують гнучкість і розширюваність, однак вимагають складної конфігурації та більших ресурсів. Вибір системи залежить не лише від функціональних потреб, а й від загальних вимог до інфраструктури та простоти підтримки.

Це зумовлює потребу у створенні простого у впровадженні апаратно-програмного комплексу для базового моніторингу, який би забезпечував швидке отримання основних метрик без навантаження на ресурси. Такий підхід є доцільним у домашньому, навчальному або офісному середовищі, де важлива стабільність і контроль, а не масштабування.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ СИСТЕМ МОНІТОРИНГУ КОМП'ЮТЕРА

1.1 Поняття та класифікація систем моніторингу

Моніторинг є невід'ємною складовою забезпечення стабільної роботи сучасних комп'ютерних систем, незалежно від їх масштабу чи призначення. Як процес спостереження за станом системи, моніторинг дозволяє отримувати інформацію про параметри функціонування апаратного та програмного забезпечення, виявляти потенційні загрози, а також реагувати на аномалії ще до того, як вони можуть призвести до серйозних збоїв або втрати працездатності, що стосується як персональних комп'ютерів, так і серверних інфраструктур, кластерів або віртуалізованих середовищ. У найпростішому вигляді моніторинг може полягати у візуальному контролі основних показників через інтерфейс операційної системи, однак зі зростанням складності систем зростає і потреба у гнучкіших і масштабованіших інструментах, які забезпечують не лише відображення поточного стану, але й накопичення історичних даних, аналіз трендів і сповіщення про критичні події [1-3].

Поняття моніторингу стану комп'ютера. Під поняттям моніторингу системи зазвичай мається на увазі процес збору та аналізу телеметричних даних, що відображають реальний стан окремих компонентів комп'ютера або іншого обчислювального пристрою. Джерелами таких даних можуть бути як апаратні елементи, так і програмні процеси, а самі показники охоплюють, зокрема, завантаження центрального процесора, температуру його ядер, середнє навантаження за певний період, обсяг використаної, вільної або зарезервованої оперативної пам'яті, заповненість накопичувачів, інтенсивність операцій читання та запису, а також стан SMART-датчиків, що вбудовані в сучасні дискові пристрої. До цієї ж групи метрик належать характеристики мережевих інтерфейсів, такі як

швидкість передавання даних, кількість активних підключень, втрати пакетів або мережеві помилки. Також до уваги можуть братись процеси операційної системи, активні служби, запущені додатки, а в деяких випадках — і стан зовнішніх баз даних або програмних серверів (див. рис. 1.1) [1].

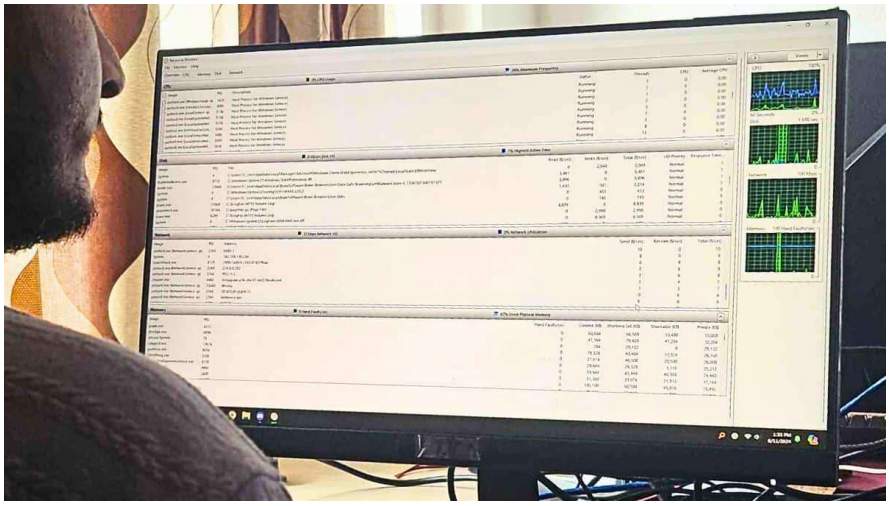


Рисунок 1.1 – Зовнішній вигляд системи моніторингу ресурсів ПК

Моніторинг може здійснюватися як локально, безпосередньо на самому пристрої, так і централізовано, коли телеметрія збирається з великої кількості вузлів у єдиному середовищі. У локальному випадку можуть використовуватися вбудовані засоби операційної системи або спеціальні інструменти з простим графічним інтерфейсом, які не потребують складного налаштування. Централізований підхід передбачає використання агентів, які встановлюються на цільові пристрої, або ж реалізацію безагентного збору даних, наприклад, за допомогою протоколів SNMP, WMI, IPMI, NetFlow тощо. В обох випадках системи моніторингу забезпечують ключову властивість — спостережливість, що дозволяє не лише фіксувати поточні параметри роботи, а й будувати загальну картину стану інфраструктури, оцінювати ефективність використання ресурсів, прогнозувати проблеми і завчасно адаптувати конфігурацію до змін у навантаженні [1].

Завдяки такому підходу створюється основа для надійного функціонування навіть простих або обмежених за ресурсами комп’ютерних систем, адже сам моніторинг не є самоціллю, а лише засобом для організації своєчасного реагування,

підвищення стабільності роботи, пошуку вузьких місць та оптимізації архітектури. У підсумку моніторинг не обмежується короткочасним переглядом окремих параметрів, а реалізується як безперервний процес нагляду за об’єктами, в межах якого формується база для прийняття технічних рішень та розвитку системи в цілому.

Класифікація систем моніторингу. Класифікація систем моніторингу може здійснюватися за різними ознаками, показниками та принципами, що відображають їхню функціональність, масштаб і спосіб впровадження. Загальна система моніторингу може мати різноманітні можливості — від ізольованих локальних варіантів до розподілених, інтегрованих у великі мережеві інфраструктури. У контексті розподілу за обсягом охоплення вирізняються локальні системи, що орієнтовані на моніторинг одного вузла, наприклад комп’ютера або сервера. У таких випадках відповідне програмне забезпечення встановлюється безпосередньо на цільовий пристрій, що дозволяє збирати й аналізувати дані в межах цього вузла. Прикладами таких систем можуть слугувати Glances — кросплатформенна утиліта, яку можна встановити різними способами, та htop або top — інструменти для Linux, які вже є частиною багатьох дистрибутивів і надають базову інформацію про стан системи [3].

У противагу локальним, мережеві або розподілені системи охоплюють кілька вузлів і забезпечують централізоване збирання інформації. Вони здатні передавати метрики з окремих джерел до центрального вузла, що дозволяє масштабувати моніторинг на великі інфраструктури. У приклад можна навести використання комбінації Prometheus та Node Exporter, де перший відповідає за збирання, обробку та зберігання метрик, а другий — за їхнє надання з окремих вузлів [3].

З огляду на рівень доступу до системних ресурсів, системи моніторингу поділяються на ті, що мають прямий доступ до апаратного рівня, де можлива взаємодія з фізичними компонентами, такими як диски, датчики, драйвери, інтерфейси управління живленням та інші, й ті, що працюють на рівні операційної системи, використовуючи системні виклики, API або бібліотеки, на кшталт psutil у Python. Окремо слід згадати системи, що функціонують у середовищах

віртуалізації або контейнеризації, які можуть реалізовувати моніторинг усередині гостьових ОС або контейнерів із передачею даних до зовнішнього облікового вузла [3].

За способом збору даних виділяють агентські системи, безагентські та гібридні. У першому випадку встановлюється окремий агент на кожен вузол, що відповідає за збирання метрик. До прикладів відносяться Zabbix Agent, Node Exporter та інші подібні сервіси. Безагентські системи покладаються на стандартні протоколи (наприклад, SSH або SNMP), що дозволяє уникати встановлення додаткового ПЗ, зменшуючи загальне навантаження на систему. Гібридні підходи поєднують обидві моделі, надаючи можливість комбінувати методи в залежності від особливостей мережі або потреб інфраструктури (див. табл. 1.1) [3].

Таблиця 1.1 – Класифікація систем моніторингу комп’ютера

Критерій	**Приклад**	**Призначення**
За об’єктом моніторингу	Системні ресурси / мережа / програми	Специфіка контролю
За способом збору даних	Агентські / безагентські	Особливості впровадження
За способом подання	CLI / GUI / Web	Зручність користування

Якщо звернути увагу на часову характеристику збору даних, то моніторинг може бути реалізований у режимі реального часу, коли інформація оновлюється щосекундно, що зручно для динамічних сценаріїв — прикладом може бути використання веб-інтерфейсу Glances. Також можливий періодичний моніторинг із певним інтервалом, наприклад, кожні 5–10 хвилин, що знижує навантаження на систему й дозволяє раціонально використовувати ресурси.

Цілі та задачі систем моніторингу. Основне призначення систем моніторингу полягає у забезпеченні стабільної роботи комп’ютерних систем за рахунок виявлення, локалізації й запобігання збоїв. Вони дозволяють виконувати низку задач, серед яких — спостереження за ресурсами, зокрема CPU, RAM,

дисковим простором, мережевими інтерфейсами та іншими компонентами. Крім того, важливим є аналіз поведінки системи, що охоплює виявлення аномальних сценаріїв, відхилень у споживанні ресурсів або інших нестандартних подій, що потенційно свідчать про збої чи атаки. Не менш суттєвою є функція сповіщення: за допомогою e-mail, Telegram, SMS або інших каналів відбувається повідомлення відповідальних осіб про критичні події, що дозволяє оперативно реагувати на проблеми навіть у разі відсутності постійного контролю за системою.

До важливих задач належить і зберігання історичних даних, що дозволяє накопичувати статистику, аналізувати динаміку роботи системи, формувати звіти, будувати графіки та виявляти закономірності. Це, у свою чергу, пов’язано з плануванням ресурсів, яке забезпечується завдяки можливості прогнозування майбутніх навантажень та ухвалення рішень щодо масштабування або оптимізації. Наприклад, у випадках, коли очевидним є недоцільність закупівлі надлишкових ресурсів, що не будуть використовуватися, застосування систем моніторингу дозволяє приймати обґрунтовані рішення, спрямовані на економічну ефективність. Водночас, у перспективі, вони допомагають оцінити необхідність розширення інфраструктури, коли очікується зростання навантаження або кількості обслуговуваних пристроїв. Саме тому інтеграція систем моніторингу є логічним кроком для забезпечення прозорості, передбачуваності та контрольованості у процесах функціонування сучасних IT-систем.

Роль моніторингу в IT-інфраструктурі. В умовах сучасного цифрового середовища IT-інфраструктура пронизує практично всі сфери діяльності, виступаючи критичним елементом функціонування організацій будь-якого типу — від невеликих підприємств і приватних компаній до транснаціональних корпорацій та державних установ. Її безперервна робота залежить від здатності своєчасно виявляти, реагувати й усувати потенційні збої або відхилення в системах, і саме з цим безпосередньо пов’язана важливість постійного моніторингу, який забезпечує стабільність, надійність і загальну ефективність функціонування всіх компонентів інфраструктури. У середовищах з великою кількістю користувачів і технічних засобів моніторинг стає єдиною ефективною відповіддю на зростаючу складність

управління системами, адже дозволяє не лише запобігати проблемам, а й оптимізувати використання ресурсів [2].

На рівні робочих станцій моніторинг дозволяє відстежувати продуктивність пристроїв, контролювати температуру, навантаження на процесор, стабільність роботи операційної системи та наявність помилок, що впливають на якість виконання завдань користувачами. Велика кількість таких пристроїв у рамках однієї організації зумовлює потребу в централізованому підході до діагностики та контролю, адже ручне виявлення несправностей було б вкрай неефективним, а час на вирішення інцидентів значно зростав би, що негативно позначалося б на загальній продуктивності [2].

У серверному сегменті інфраструктури безперервне спостереження є основою для попередження перенавантаження, відмов у роботі, втрати даних, деградації сервісів або порушення доступності. Моніторинг забезпечує контроль за роботою критично важливих служб, серверів додатків, файлових систем і мережевих підключень, завдяки чому можна оперативно локалізувати джерело проблеми та здійснити відповідні дії до того, як це стане причиною простою або збоїв для кінцевих користувачів [2].

У віртуалізованому середовищі, де фізичні ресурси розподіляються між численними віртуальними машинами, постає завдання ефективного управління цими ресурсами, що вимагає ретельного моніторингу. Нестача ресурсів або неправильний розподіл навантаження між гіпервізором і віртуальними інстанціями може призвести до деградації продуктивності не лише окремої служби, а й усієї системи в цілому. В умовах активного використання хмарних платформ спостереження поширюється й на кластеризовані сервіси, API, мікросервіси, бази даних, функції безсерверної обробки, а також аналіз затримок і показників доступності, що є критично важливим для побудови масштабованих і високодоступних архітектур [2].

Особливе значення моніторинг набуває в контексті DevOps-практик, де він є складовою безперервної інтеграції та розгортання (CI/CD), забезпечуючи негайну зворотну реакцію на зміни в системі, що розгортається. Рівень спостережуваності

після кожного етапу оновлення або деплою стає ключем до підтримки якості програмного забезпечення та скорочення часу на усунення дефектів. Інтеграція моніторингу з системами логування, трасування (tracing) та аналізу подій формує єдине інформаційне поле, у межах якого реалізується концепція спостереження (observability) як повноцінної екосистеми для відстеження та розуміння внутрішніх процесів системи за зовнішніми метриками (див. рис. 1.2).



Рисунок 1.2 – Взаємозв'язок моніторингу з IT-інфраструктурою

Таке інтегроване середовище створює основу для впровадження підходів Site Reliability Engineering (SRE), які, у свою чергу, дозволяють підтримувати угоди про рівень обслуговування (SLA), забезпечуючи відповідність очікуванням кінцевих користувачів та бізнесу. У результаті моніторинг не є просто допоміжним інструментом, а перетворюється на стратегічну складову цифрової інфраструктури, без якої неможливо забезпечити безперервність та ефективність сучасних IT-систем [2].

1.2 Протоколи та технології збору телеметрії

У контексті сучасної IT-інфраструктури моніторинг безпосередньо пов’язаний із механізмами збору телеметрії, які забезпечують спостереження за станом систем, сервісів, обладнання та програмного забезпечення. Загалом, система моніторингу базується на різноманітних компонентах, що залежать від конкретного середовища, архітектури, масштабу й задач, і тому може включати локальні, агентські, безагентські або гібридні методи. Така різноманітність підходів вимагає глибшого розуміння відмінностей між ними, їхніх переваг, обмежень та сумісності з тими чи іншими телеметричними протоколами й технологіями збору даних. У межах інфраструктурного моніторингу, телеметрія є механізмом збору, передачі, зберігання та аналізу інформації щодо параметрів функціонування систем на різних рівнях — від апаратного забезпечення до програмних сервісів, включно з мережевими показниками та станом операційних систем. Підхід до збору телеметричних даних може варіюватися від найпростіших локальних засобів до складних централізованих рішень на базі корпоративних стандартів, що застосовуються в дата-центрах, хмарних середовищах, DevOps-циклах та на виробничих платформах [4-6].

Загальні підходи до збору телеметричних даних. Локальний моніторинг передбачає збирання даних безпосередньо на самому пристрої без потреби в зовнішньому з’єднанні, використовуючи читання системних файлів або API операційної системи. Типовими прикладами є вбудовані або сторонні інструменти командного рядка типу top, htop, vmstat, glances, iostat, які зазвичай не потребують мережевих ресурсів, забезпечуючи мінімальну затримку. Цей підхід особливо ефективний у випадках, коли важливо зменшити навантаження на мережу або коли дані потрібні лише локально. Однак він обмежений у масштабуванні та не підходить для централізованого аналізу, що створює труднощі в середовищах із великою кількістю вузлів [4].

Агентський моніторинг базується на встановленні спеціальних програмних агентів на вузлах, з яких треба зібрати метрики. Ці агенти (наприклад, Node Exporter для Prometheus або Zabbix Agent) періодично передають зібрані дані до централізованої системи обробки — Prometheus, Zabbix, InfluxDB, Telegraf та інших. Такий підхід забезпечує високу точність, широку підтримку різних типів метрик, гнучкість у конфігурації та розширюваність. Проте недоліком є додаткове навантаження на ресурси системи, що може стати критичним на старому або обмеженому обладнанні, а також необхідність встановлення і конфігурування агентів, що ускладнює впровадження у великих інфраструктурах без автоматизації [4].

Безагентський моніторинг або agentless-модель функціонує на основі стандартних мережевих протоколів, які дозволяють отримувати інформацію без встановлення додаткового ПЗ. До таких протоколів належать SNMP, WMI, SSH, IPMI, REST API — вони дають змогу отримувати базову або проміжну телеметрію зі значною універсальністю та мінімальним втручанням у системи, що моніторяться. Цей метод забезпечує простоту впровадження, не вимагає обслуговування агента, часто працює з «коробки» навіть на застарілих системах. Але водночас має обмежену деталізацію зібраних метрик, залежність від підтримки необхідних протоколів у системі, а також не завжди дозволяє досягнути високої частоти оновлення або точності (див. рис. 1.3) [4].

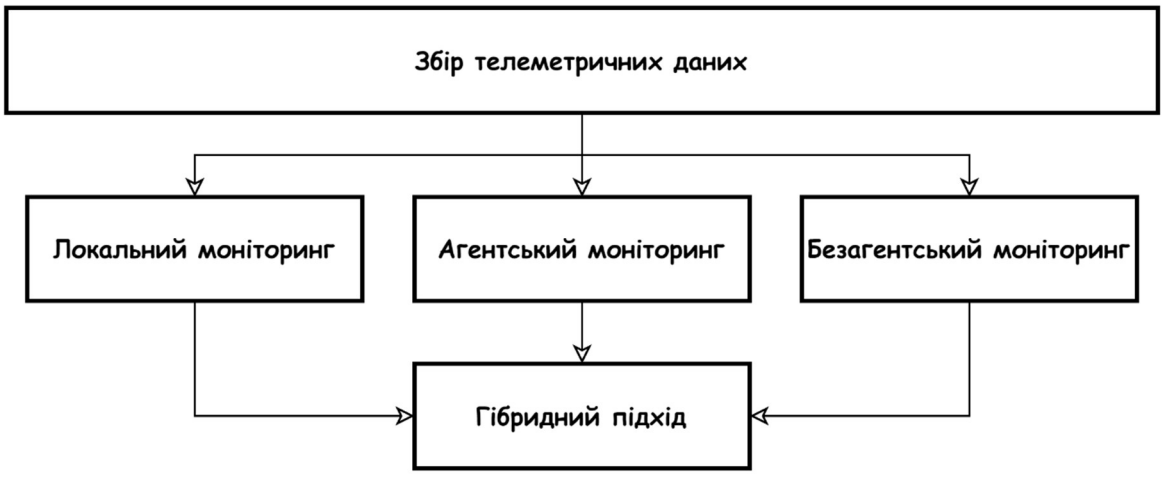


Рисунок 1.3 – Загальна схема збору телеметричних даних

Існує також гібридний моніторинг, який поєднує в собі характеристики локального, агентського й безагентського методів, забезпечуючи максимальну гнучкість у різних середовищах, дозволяючи обирати найбільш доцільний інструмент для кожного вузла окремо. Така модель може бути зручною для організацій із різнотипними інфраструктурами, де, наприклад, старіші системи можна моніторити за допомогою SNMP, а сучасні сервери з агентами інтегрувати у централізовану телеметрійну платформу, підтримуючи масштабованість та глибину збору інформації на високому рівні [4].

Основні протоколи телеметрії. Для ефективного збору інформації з різноманітних IT-середовищ необхідно використовувати стандартизовані протоколи, адже вузькоспеціалізовані варіанти можуть бути несумісні з тими чи іншими операційними системами або конкретними типами обладнання. Чим поширеніший і загальніший протокол застосовується, тим більше гарантована його підтримка на різних платформах. Найвідомішим серед таких є SNMP (Simple Network Management Protocol) — інтернет-протокол, який широко використовується для моніторингу мережевого обладнання, серверів та периферійних пристроїв. Його робота ґрунтується на моделі «менеджер–агент», де застосовуються UDP-порти 161 і 162 та підтримується як читання, так і запис даних через операції Get і Set. Застосовується він у комутаторах, маршрутизаторах, принтерах, серверах, сумісний навіть зі старими системами, що робить його універсальним рішенням, хоча варто врахувати обмежену деталізацію телеметрії порівняно з іншими методами збору та певну складність налаштування сучасної версії SNMPv3, яка хоч і значно безпечніша, але рідко підтримується застарілими платформами [5].

Ще одним важливим протоколом є WMI (Windows Management Instrumentation), розроблений компанією Microsoft для управління та моніторингу Windows-систем. Ця технологія надає повний доступ до параметрів ОС, процесів, служб, журналів та багатьох інших аспектів і часто інтегрується з Zabbix, Nagios або PowerShell-скриптами. Її недоліком є відсутність сумісності з іншими операційними системами, тому вона застосовна виключно в межах Windows-

середовища. Запити в межах WMI складніші, ніж прості Get/Set у SNMP, проте функціональність значно ширша [5].

Ще одним прогресивним підходом є використання REST API — інтерфейсу, що базується на HTTP-запитах і дозволяє інтегруватися з численними сучасними платформами моніторингу. Цей метод забезпечує зручність, швидкість та масштабованість, особливо у випадках, коли йдеться про сервіси на базі хмари чи контейнеризацію. Типовими прикладами таких API є Prometheus, Netdata Cloud чи Docker Engine API, які дозволяють збирати дані у форматі JSON або XML залежно від реалізації. REST API є кросплатформенним рішенням (див. рис. 1.4).

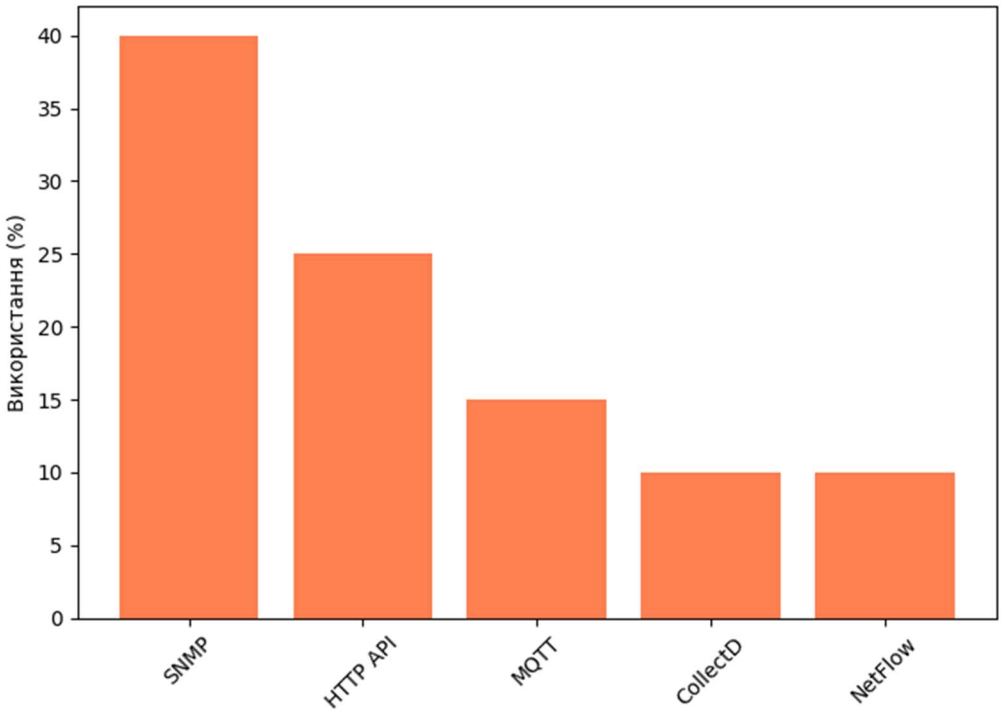


Рисунок 1.4 – Популярність протоколів збору телеметрії

Все це робить його універсальним інструментом у сучасних системах, хоча він може вимагати ручного опрацювання відповідей або налаштування автоматизації цього процесу (див. табл. 1.2) [5].

Таблиця 1.2 – Порівняння основних протоколів збору телеметрії

Протокол	**Порт**	**Підтримка в Linux**	**Переваги**	**Недоліки**
SNMP	161	Так	Стандартизований	Старий формат
Netdata	19999	Так	Простота	Важкий клієнт
REST API	-	Так	Гнучкість	Складність реалізації

Окрім цього, популярністю користується Python-бібліотека psutil (Python System and Process Utilities), яка дозволяє зчитувати дані про стан системи — процеси, завантаження CPU, пам’ять, мережу, диски та інші параметри. Вона підтримується на Windows, Linux та macOS, забезпечує зручність у кастомізованих скриптах моніторингу та відзначається простим синтаксисом, хоча для її використання необхідне базове розуміння командного рядка чи Python-коду (див. рис. 1.5).

```
import psutil
print(psutil.cpu_percent(interval=1))
print(psutil.virtual_memory().percent)
```

Рисунок 1.5 – Фрагмент коду для використання бібліотеки psutil

Даний інструмент дозволяє реалізувати локальний моніторинг і інтегрується у більші рішення завдяки своїй гнучкості [5].

Інструменти збору інформації в дистрибутивах GNU/Linux. У контексті дистрибутивів GNU/Linux, які надзвичайно поширені як серед серверів, так і в безкоштовних інфраструктурах з відкритим кодом, вже у базових дистрибутивах доступний широкий набір інструментів для збору інформації. У цьому середовищі значно більше штатних та сторонніх засобів моніторингу порівняно з Windows. Наприклад, стандартні утиліти top, htop дозволяють переглядати активні процеси і навантаження на систему, vmstat надає статистику пам’яті та введення/виведення,

iostat — продуктивність дискової підсистеми, df — інформацію про файлові системи, free — про доступну пам’ять, netstat або ss — про мережеві з’єднання, uptime, dmesg, journalctl — про загальний стан системи та журнал подій (див. рис. 1.6).

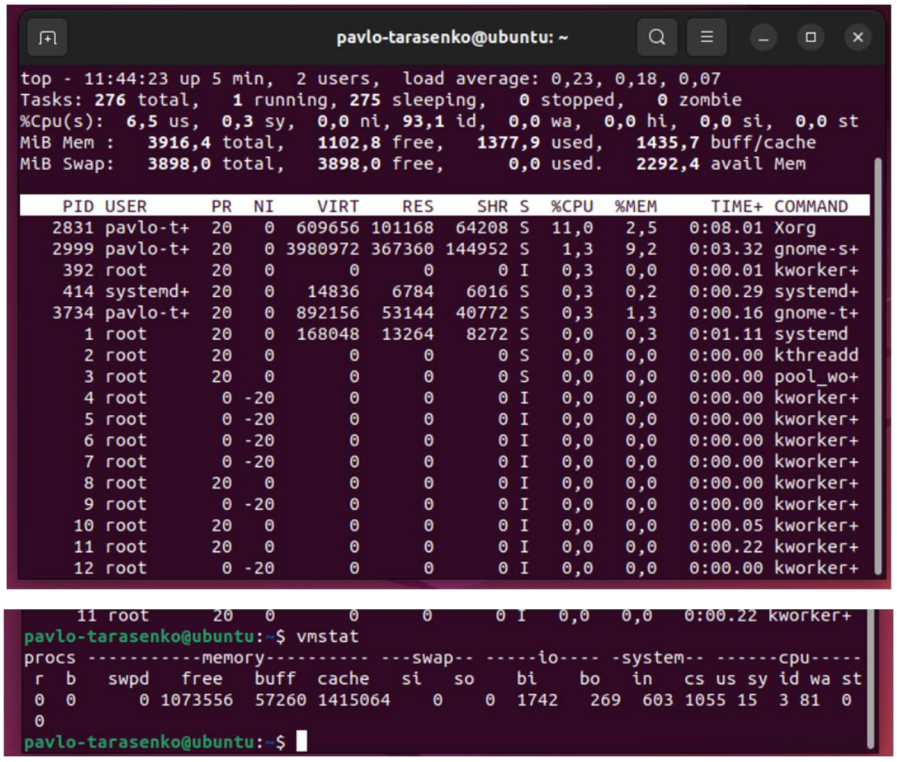


Рисунок 1.6 – Приклад збору метрик через top та vmstat у Linux

Проте, коли виникає потреба в централізованому або стандартизованому моніторингу, варто звернути увагу на сучасніші інструменти. Одним із найпоширеніших є Node Exporter — агент для експортування метрик у форматі Prometheus через HTTP-протокол, який ідеально підходить для інтеграції у великі системи. Іншим потужним рішенням є Netdata — сучасна платформа з візуальним веб-інтерфейсом, що дозволяє отримувати дані в реальному часі з будь-якого вузла, використовуючи як локальний агент, так і хмарний сервіс з можливістю підписки. Також широко використовується Glances — універсальна кросплатформна Python-утиліта для моніторингу, яка підтримує веб-інтерфейс та API, а також може експортувати зібрані дані в InfluxDB, що дозволяє гнучко інтегрувати її в різні архітектури моніторингу.

У системах на базі GNU/Linux важливу роль також відіграють засоби журналювання. У випадках, коли неможливо в реальному часі відстежувати телеметрію або необхідно накопичувати історичні дані для аналізу, системні журнали стають основним джерелом інформації про події. Найбільш поширеними інструментами є rsyslog та systemd-journald, які відповідають за збирання, зберігання та обробку логів. Додатково використовується logrotate, що дозволяє автоматично керувати обсягом логів шляхом архівації та видалення старих записів, що допомагає уникати перевантаження диску. Завдяки цим інструментам забезпечується як ручний, так і автоматизований збір, обробка і збереження телеметричних даних у складі більших систем на кшталт Prometheus з Node Exporter, Netdata або Glances, що розглядатимуться в межах даної роботи.

Бібліотеки та стандарти для збору даних. Ще одним способом побудови моніторингового рішення є інтеграція механізмів збору телеметрії безпосередньо у власні програмні продукти або створення індивідуальних скриптів автоматизації, які здатні взаємодіяти з інфраструктурою на низькому рівні, отримуючи специфічні метрики, що важко або неможливо дістати за допомогою готових засобів. У цьому контексті широке поширення отримали різноманітні бібліотеки, які надають розробникам гнучкий інструментарій для програмної реалізації збору даних. Наприклад, однією з найвідоміших утиліт є psutil — бібліотека для Python, яка завдяки своєму простому синтаксису та безпосередньому доступу до основних апаратних і системних параметрів дозволяє отримувати інформацію про процеси, навантаження на CPU, використання пам’яті, дискову активність, мережеві інтерфейси тощо. Саме завдяки своїй універсальності вона часто використовується в кастомних рішеннях, де необхідно створити власні сценарії моніторингу або автоматичного реагування, а також реалізувати сервери, що формують власні API для збору метрик чи навіть підміни поведінки у рамках тестування систем (див. рис. 1.7) [6].



Рисунок 1.7 – Візуальна інтерпретація бібліотеки psutil для моніторингу

Окрему увагу варто приділити бібліотеці prometheus_client, яка дозволяє створювати користувацькі метрики у форматі, сумісному з Prometheus, та експортувати їх за допомогою HTTP-інтерфейсу, що робить можливим спостереження за станом програмних компонентів безпосередньо у режимі реального часу. Така бібліотека підтримується на різних мовах, зокрема Python, Go та Java, й активно використовується у веб-додатках, мікросервісах і backend-системах, де необхідна глибока інтеграція моніторингу в архітектуру. Для спрощення інтеграції з існуючою інфраструктурою часто застосовуються утиліти на кшталт node_exporter, який передає низькорівневі метрики операційної системи в Prometheus, однак за необхідності більш специфічного контролю — наприклад, за подіями певного рівня в бізнес-логіці чи нестандартними джерелами даних — необхідне саме програмне розширення за допомогою відповідних клієнтів і бібліотек [6].

У системах на базі Linux також доступна низка вбудованих служб і утиліт, таких як os, subprocess або sh, які дозволяють отримувати необхідну інформацію

безпосередньо через командну оболонку, зчитуючи вміст спеціалізованих файлів або викликаючи стандартні команди. Водночас сучасна практика збору метрик передбачає також використання сторонніх бібліотек, таких як pymongo для моніторингу MongoDB, MySQL Connector для MySQL, python-docker для контейнеризованих середовищ, які значно спрощують доступ до внутрішніх лічильників баз даних або контейнерів та дозволяють вибудовувати комплексну систему спостереження за поведінкою системи з урахуванням бізнес-логіки [6].

Застосування таких бібліотек дозволяє адаптувати рішення до специфіки задачі, особливо якщо готові інструменти не забезпечують достатню точність, є занадто ресурсоємними або мають обмеження за ліцензуванням чи вартістю. Узагальнюючи зазначене, можна констатувати, що збір телеметрії є критично важливим етапом для забезпечення ефективної роботи будь-якої системи моніторингу, включно з інфраструктурами, орієнтованими на високу доступність, безперервність обслуговування та масштабованість [6].

У залежності від вимог до точності, безпеки, обсягу даних і обмежень щодо ресурсів можуть застосовуватись різні архітектури: агентські, безагентські або комбіновані, якщо інфраструктура складається з неоднорідних компонентів або є значною за масштабами. При цьому вибір протоколів і засобів інтеграції визначається як середовищем виконання, так і технічними обмеженнями. Наприклад, SNMP, REST API або gRPC можуть стати основою передачі даних, а у середовищах Linux штатні засоби доступу до системних параметрів можуть бути використані як окремо, так і у зв’язці з більшими системами спостереження. У випадку, коли потрібна максимальна гнучкість, бібліотека psutil залишається одним з найзручніших засобів для створення програмного збору метрик в рамках індивідуальних рішень із власною логікою моніторингу, що надає широкі можливості для кастомізації та глибокої інтеграції з прикладною логікою [6].

1.3 Методи візуалізації та подання даних

У системах моніторингу візуалізація даних є важливим елементом, який забезпечує значно ефективніше сприйняття інформації порівняно з простим відображенням числових показників у терміналі або командному рядку, оскільки саме через графічні інтерфейси з'являється можливість взаємодії з телеметричними даними, аналізу аномалій, оперативного прийняття рішень і своєчасного реагування на позаштатні ситуації, що можуть виникати. Рівень якості та ефективності цієї візуалізації значною мірою визначає функціональні можливості впровадженої системи моніторингу залежно від конкретного середовища застосування, типу інфраструктури й поставлених завдань [7-9].

Підходи до побудови Web-інтерфейсів. З огляду на те, що більшість сучасних систем моніторингу передбачають використання веб-інтерфейсів як основного інструменту представлення інформації, можна виділити кілька підходів до побудови таких інтерфейсів, які охоплюють класичні багатосторінкові рішення, односторінкові застосунки та дашборди як сервіс. У класичному багатосторінковому підході (MPA — Multi Page Application) реалізація полягає в тому, що при кожному переході між сторінками відбувається повне перезавантаження вмісту, а дані формуються безпосередньо на стороні серверного бека, що генерує HTML-вміст і надсилає його клієнту. Подібний підхід використовується у таких рішеннях, як Zabbix чи Cacti, які забезпечують простоту реалізації, чітку роботу з шаблонами та стандартну архітектуру, однак обмеженням є повільна реакція інтерфейсу, значне навантаження на мережу та менш зручна інтерактивність при роботі з динамічними даними [8].

Альтернативою є односторінкові застосунки (SPA — Single Page Application), де вся структура інтерфейсу завантажується лише один раз, а подальша взаємодія відбувається без повторного оновлення сторінки, завдяки чому забезпечується швидке реагування, динамічне оновлення інформації та значно більша інтерактивність. У таких рішеннях дані зазвичай передаються з бекенду

асинхронно через REST API, GraphQL або інші формати API, а серед популярних технологій, що застосовуються для фронтенду, переважають React, Vue.js і Angular. Попри суттєві переваги, ці рішення потребують складнішої реалізації, розробки окремих серверних компонентів, що не завжди є прийнятним у всіх проєктах, особливо коли необхідна швидка інтеграція або обмежені ресурси [8].

У випадках, коли потрібно досягти швидкого розгортання та мінімальних витрат на розробку, використовуються дашборди як сервіс, що являють собою готові платформи для візуалізації, з вбудованими шаблонами та конекторами до різних джерел даних. Такі рішення, як Grafana, Kibana та NetData Cloud, дозволяють без значних зусиль інтегруватися з популярними системами моніторингу, використовувати авторизацію та готові шаблони для різних типів даних. Основною перевагою є легкість впровадження, підтримка великої кількості джерел і адаптація до широкого спектра завдань без необхідності розробки нових інструментів вручну, хоча обмеження кастомізації, залежність від структури даних і підтримуваних форматів (наприклад, JSON або YAML) можуть впливати на гнучкість рішення [8].

Grafana, наприклад, функціонує як компонент у більшій системі моніторингу, отримуючи метрики з Prometheus або інших джерел, при цьому Kibana здебільшого потребує використання стеку ELK для повноцінної роботи, а NetData Cloud, у локальному варіанті, хоч і не зберігає телеметрію, дозволяє усунути це обмеження шляхом підключення до хмарного середовища (див. рис. 1.8) [8].



Рисунок 1.8 – Web-інтерфейс Grafana для візуалізації моніторингових даних

Таким чином, вибір методу візуалізації, як і самої архітектури інтерфейсу, повинен визначатися вимогами до масштабованості, швидкодії, складності інтеграції та підтримуваними джерелами телеметрії в конкретному середовищі розгортання.

Способи подання даних – графіки, таблички, віджети. Важливим аспектом у побудові системи моніторингу є не лише збір телеметричних даних, а й спосіб їх подання, оскільки саме форма візуалізації визначає зручність сприйняття, аналізу та оперативної реакції на події. Дані можуть бути представлені у вигляді графіків, таблиць або віджетів, і вибір тієї чи іншої форми залежить від характеру інформації, контексту її застосування, а також від потреб цільової аудиторії. Наприклад, графіки дають змогу візуально оцінити динаміку змін, побачити тенденції, пікові навантаження або відхилення в роботі системи, і можуть бути представлені у формі лінійних діаграм, гістограм, кругових графіків або часових шкал. Їх перевагою є наочність, компактність та зручність для побудови комплексних візуалізацій, таких як дешборди, однак вони менш придатні для точкового аналізу окремих значень. До типових прикладів використання графіків належать: графік навантаження CPU за часом, який зазвичай є лінійним; кругова

діаграма розподілу трафіку між інтерфейсами; гістограма кількості помилок за днями або часом [7].

Коли виникає потреба у детальному перегляді даних, які важко або неможливо зобразити графічно, доцільно використовувати табличне представлення. Таблиці забезпечують точний перегляд числових значень, подій, журналів, параметрів системи з можливістю сортування і фільтрації, що робить їх незамінними при роботі з великими обсягами інформації. Наприклад, таблиця може містити список останніх помилок, поточні метрики із зазначенням одиниць виміру, інформацію про диски, мережеві інтерфейси або процеси. Водночас значною вадою табличного способу є можливість перевантаження інтерфейсу при надмірній кількості рядків, що іноді призводить до зниження продуктивності або незручностей у сприйнятті [7].

Ще одним поширеним форматом візуалізації є віджети — невеликі компактні елементи, що дозволяють миттєво оцінити загальний стан системи завдяки візуальним індикаторам, кольорам або піктограмам. Вони можуть бути реалізовані у вигляді спідометрів, кругових шкал, кольорових міток. Наприклад, зелений колір може позначати нормальний стан, жовтий — попередження, а червоний — критичне навантаження. Такий підхід особливо ефективний для поверхневого моніторингу в реальному часі, хоча й не дозволяє глибоко аналізувати причини або структуру подій. Серед прикладів можна назвати: віджет завантаження CPU у вигляді спідометра, індикатор статусу підключення системи (онлайн/офлайн), круговий індикатор поточного рівня використання оперативної пам’яті або дискового простору. Важливо, що в сучасних системах моніторингу практикується комбіноване використання всіх трьох форматів — графіків, таблиць і віджетів — оскільки жоден з них не може повноцінно замінити інші (див. рис. 1.9).

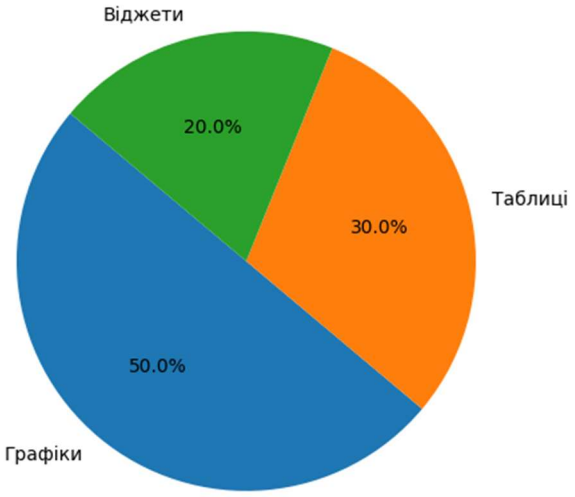


Рисунок 1.9 – Типи візуалізації у співвідношені до використання: графіки, таблиці, віджети

Разом вони створюють цілісну систему візуалізації, яка забезпечує як загальний огляд, так і детальне заглиблення в критичні аспекти роботи системи [7].

Вплив інтерфейсу користувача на зручність сприйняття. Зручність сприйняття даних безпосередньо залежить від інтерфейсу користувача, який є посередником між оператором та інформаційною системою. Інтерфейс, побудований за принципами ефективного графічного дизайну, дозволяє не лише скоротити час на аналіз інформації, а й мінімізувати ризики помилкового трактування даних. Добре реалізований UI допомагає гармонізувати відображення інформації, що значною мірою покращує оперативність реагування у разі виявлення аномалій. Зважаючи на сучасні вимоги до гнучкості та зручності, ефективний інтерфейс повинен бути побудований на основі кількох ключових принципів: консистентність, колірне кодування, ієрархічна структура, мінімалізм та адаптивність.

Консистентність полягає в однаковому оформленні всіх елементів інтерфейсу, що дозволяє користувачеві легко орієнтуватися у системі, незалежно від того, яку частину UI він відкриває. Колірна система є не менш важливою — зазвичай зелений означає норму, жовтий — попередження, а червоний — критичний стан. Ця система дозволяє миттєво оцінити загальний стан без потреби

у детальному аналізі. Ієрархічна побудова забезпечує можливість спочатку ознайомитись з узагальненою інформацією, а за потреби — перейти до деталізації окремих компонентів або підсистем. Наприклад, спочатку може відображатися лише загальний стан серверів, а при натисканні — розгортатись детальні параметри кожного з них.

Мінімалізм як принцип полягає у відмові від зайвих елементів, що лише перевантажують інтерфейс і ускладнюють сприйняття. Людина, на відміну від комп'ютера, краще інтерпретує інформацію у вигляді простих, логічно впорядкованих і візуально комфортних об'єктів, тому надлишок декоративних елементів чи непотрібних даних заважає швидко ухвалювати рішення. Адаптивність дозволяє інтерфейсу залишатися зручним як на десктопах, так і на мобільних пристроях з різною роздільною здатністю, що передбачає окремі реалізації інтерфейсу для різних типів екранів, аби, наприклад, дешборд, зручний на великому моніторі, мав не менш зручну версію для смартфона або планшета.

Інтерфейс сучасних систем моніторингу зазвичай реалізується з використанням frontend-технологій, таких як HTML, CSS або JavaScript-фреймворки, і водночас повинен базуватись на принципах UX-дизайну, які включають можливість здійснення фільтрації, пошуку, налаштування порогів, тригерів та збереження параметрів візуалізації. Наприклад, у разі якщо навантаження на процесор перевищує 90%, індикатор автоматично змінює колір на червоний, а інтерфейс генерує спливаюче попередження — це дозволяє оператору миттєво вжити заходів, не аналізуючи вручну весь графік (див. рис. 1.10).



Рисунок 1.10 – UX-порівняння простого та адаптивного інтерфейсу

Такий підхід поєднує точність, швидкість та інтуїтивність взаємодії, що є основою ефективної роботи з телеметричними даними в реальному часі.

SPA та інтеграція з бекендом. Сучасні системи моніторингу дедалі частіше реалізуються у форматі односторінкових застосунків — Single Page Application (SPA), які стали практично стандартом у розробці інтерфейсів для таких рішень. У SPA уся логіка користувацького інтерфейсу зосереджена на клієнтській стороні, зазвичай за допомогою JavaScript або фреймворків на його основі, таких як React чи Vue.js. Дані при цьому надходять у форматі JSON із REST API або через WebSocket-з'єднання. Для візуалізації застосовуються потужні бібліотеки, наприклад Chart.js, D3.js, ApexCharts та інші, що забезпечують інтерактивність графіків, динамічне оновлення таблиць, індикаторів і панелей управління [9].

Прикладом такої інтеграції може слугувати архітектура, де бекенд, реалізований на Flask у поєднанні з системою збору метрик Prometheus, надає доступ до даних через API типу /api/metrics. Зі сторони фронтенду SPA, створений на React, здійснює регулярні запити, наприклад кожні 5 секунд, до цього API для оновлення графіків у режимі реального часу. Третім важливим елементом є графічний інтерфейс, в якому розміщуються індикатори, дашборди, таблиці з фільтрами подій та інші інтерактивні елементи. Така реалізація дозволяє безперервно отримувати нову інформацію, не перевантажуючи сторінку та зберігаючи високу швидкість взаємодії з інтерфейсом [9].

Окрім технічної ефективності, SPA-рішення забезпечують гнучку структуру компонентів і можливість розгортання на будь-якому сервері. Серед типових сторінок, які реалізуються в подібних інтерфейсах, можна виокремити Dashboard — оглядову панель з ключовими метриками, Metrics Page — детальну сторінку з інформацією про процесор, пам'ять, дискову підсистему тощо, та Events Table — таблицю подій із можливістю фільтрації та аналізу окремих інцидентів. Усі ці елементи виконують важливу функцію оперативного спостереження за станом систем [9].

Методи візуалізації, що використовуються у таких системах, повинні відповідати вимогам зручності, масштабованості, швидкодії та адаптивності, що

робить SPA найефективнішим підходом для реалізації сучасного моніторингового інтерфейсу. Вибір між типами відображення — графіки, таблиці, індикатори, віджети — залежить від конкретної мети моніторингової панелі, її функціонального призначення та очікуваної поведінки користувачів. Якісний UI/UX дозволяє швидко оцінити поточний стан об’єктів моніторингу, виявити аномалії або потенційні проблеми, а також забезпечує можливість негайного реагування (див. рис. 1.11) [9].

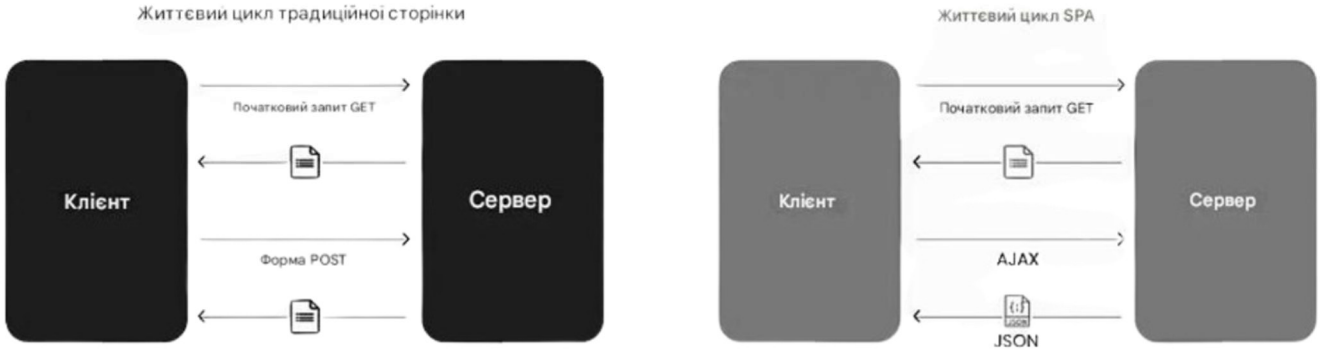


Рисунок 1.11 – Архітектура SPA з підключенням до бекенду

Інтеграція фронтенду з бекендом через API у такому підході виступає обов’язковою умовою створення гнучкого, адаптивного, динамічного рішення. Можна використовувати як повністю кастомні реалізації, так і готові платформи, які вже мають попередньо реалізовані ключові компоненти, адаптовані під потреби моніторингу комп’ютерних систем, що дозволяє зручно аналізувати стан як апаратних, так і програмних ресурсів, збирати телеметричні дані, систематизувати їх і приймати обґрунтовані рішення, ґрунтуючись на зручному та інформативному представленні даних.

1.4 Огляд готових рішень для моніторингу

У процесі розробки системи моніторингу одним із важливих завдань стає вибір готових інструментів або платформ, які здатні забезпечити повний цикл роботи з телеметричними даними — від збору, обробки й збереження до подальшої візуалізації. Готові рішення найчастіше використовуються замість створення всього з нуля, адже не кожен здатен побудувати повноцінну систему з відповідною функціональністю, яку вже забезпечують перевірені й підтримувані інструменти. На основі таких платформ можна налаштовувати власні графіки, таблиці, індикатори й інші елементи дашбордів, адаптуючи їх під конкретну кількість вузлів інфраструктури — це можуть бути як окремі сервери, так і численні робочі станції в межах IT-мережі [10-12].

Короткий опис кожного з рішень. Серед поширених рішень, які активно застосовуються в сучасних моніторингових системах, можна виокремити три приклади: умовно безкоштовний стек Prometheus зі Grafana, комерційне середовище Zabbix та легке утилітарне рішення Glances, яке розглядатиметься як базове в даному випадку. Кожне з цих рішень має власний набір функцій, специфіку інтерфейсу, сценарії використання й вимоги до розгортання, тому порівняння їх в контексті створення MVP дозволяє краще оцінити доцільність інтеграції того чи іншого інструменту в конкретну систему [10].

Glances — це кросплатформна консольна утиліта моніторингу, яка дозволяє в реальному часі відстежувати широкий спектр системних метрик. Реалізована на Python, вона підтримує як CLI-інтерфейс, так і веб-режим, а також може працювати в API-режимі з вихідними даними у форматі JSON. Серед ресурсів, які можна моніторити, доступні процесор, пам’ять, диски, мережеві інтерфейси, активні процеси тощо. Платформа дозволяє здійснювати як локальний, так і віддалений моніторинг через HTTP або SSH, а також підтримує роботу в середовищі Docker, що спрощує інтеграцію в сучасні інфраструктури на базі контейнеризації. Завдяки підтримці Linux, macOS, Windows і можливості запуску у вигляді сервісу, Glances

вважається універсальним рішенням для реалізації базової моніторингової логіки [10].

Prometheus у свою чергу зосереджений на зборі метрик і використовується переважно в архітектурах, орієнтованих на мікросервіси. Використовуючи механізм опитування (pull), Prometheus збирає дані через HTTP/HTTPS та обробляє їх за допомогою власної мови запитів PromQL. Його функціональність доповнюється можливостями побудови алертів та алгоритмів виявлення аномалій. Однак для повноцінного візуального представлення даних Prometheus зазвичай інтегрується з Grafana — потужним інструментом створення інтерфейсів моніторингу, який дозволяє будувати кастомізовані дашборди з графіками, таблицями та віджетами, а також підтримує численні джерела даних, автентифікацію, систему плагінів та інтеграцій. Такий підхід дозволяє зручно представляти телеметрію користувачеві через адаптивний UI і створювати гнучкі візуальні панелі контролю [10].

Zabbix відрізняється своєю орієнтацією на корпоративний сегмент і пропонує більш широкі можливості завдяки підтримці як агентного, так і безагентного моніторингу (див. рис. 1.12).

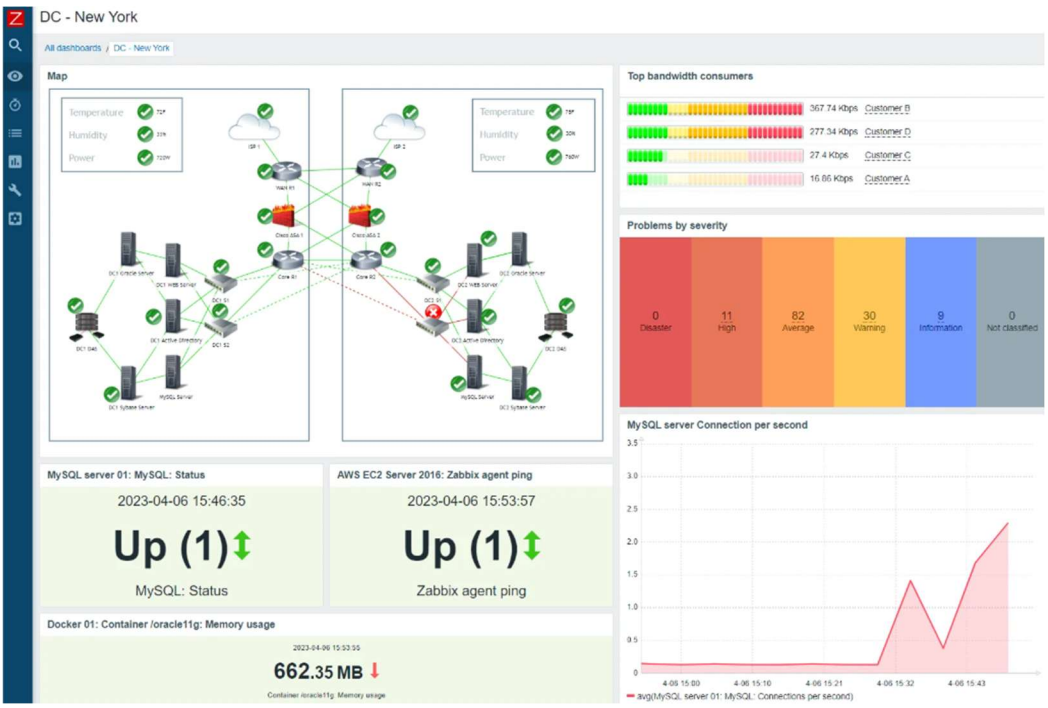


Рисунок 1.12 – Комерційне рішення для візуалізації моніторингових даних: Zabbix

Вбудована СУБД, система шаблонів, розширений веб-інтерфейс і вбудована система сповіщень дозволяють автоматизувати керування великими інфраструктурами, де важлива масштабованість і централізований контроль. Інструмент також забезпечує моніторинг серверів, додатків, мережевого обладнання, підтримку залежностей між подіями, тригери, алерти та кастомізовані сценарії інтеграції, що робить його зручним для впровадження у великих організаціях з критичними вимогами до стабільності й надійності [10].

Таблиця 1.3 – Порівняння популярних готових рішень для моніторингу

Назва	**Вимоги до ОС**	**Споживання ресурсів**	**Простота налаштування**	**Наявність Web UI**
Glances	Linux / Windows	Низьке	Висока	Так
Netdata	Linux	Середнє	Середня	Так
Zabbix	Linux	Високе	Низька	Так

Усі три рішення (див. табл. 1.3) дозволяють реалізувати повноцінну систему моніторингу, проте вибір конкретного варіанту має базуватися на вимогах до простоти, функціональності, доступності та адаптації під існуюче середовище. Glances може стати оптимальним варіантом для швидкого розгортання MVP і забезпечення базових потреб у моніторингу, тоді як Prometheus із Grafana або Zabbix дозволяють побудувати більш складні масштабовані рішення з розширеною візуалізацією, автоматизацією сповіщень і централізованим керуванням даними [10].

Порівняння вимог до встановлення та використання. Порівнюючи вимоги до встановлення та використання кожного з рішень, необхідно враховувати, що кожне з них створене з урахуванням певного контексту застосування, тому одне може бути кращим у конкретному випадку, ніж інше. Хоча зовні вони можуть здаватися схожими, за своєю суттю мають суттєві відмінності. Однією з основних характеристик, яка впливає на вибір, є мова реалізації: Glances створено на Python, що забезпечує добру гнучкість і простоту інтеграції, хоча й поступається за

продуктивністю системам, розробленим на таких мовах як C чи C++. Prometheus реалізовано на Go, що дозволяє йому працювати з високою ефективністю при обробці великої кількості метрик, а Grafana – на TypeScript, оскільки її основна функція полягає у візуалізації дашбордів. У випадку Zabbix реалізація здійснена за допомогою C та PHP, що дозволяє забезпечити стабільність і гнучкість на рівні великих корпоративних середовищ [11].

Якщо розглядати вимоги до ресурсів, Glances має низькі вимоги, що і стало однією з причин його вибору для простих або локальних рішень. Prometheus із Grafana, будучи двома окремими складовими, потребують середніх ресурсів для повноцінної роботи, особливо при значному обсязі метрик. Zabbix, як комерційне рішення корпоративного рівня, має середні або навіть високі вимоги до ресурсів, адже для його ефективної роботи потрібна вже налаштована та стабільна інфраструктура [11].

Щодо зберігання даних, Glances за замовчуванням не має вбудованої бази, що означає обробку інформації переважно в реальному часі без збереження історії, хоча при необхідності її можна додати через додаткові модулі. Prometheus має власну time-series базу (TSDB), яка дозволяє ефективно зберігати та обробляти метрики. Zabbix у свою чергу використовує повноцінні СУБД, як-от MySQL або PostgreSQL, що дає змогу створювати більш масштабовану та довготривалу аналітику [11].

Інтерфейс Glances поєднує командний рядок та веб-інтерфейс, забезпечуючи базовий рівень керування як через термінал, так і через браузер. Prometheus з Grafana здебільшого використовують веб-інтерфейс у вигляді SPA (Single Page Application), який є одним з найзручніших і найпопулярніших рішень для побудови інтерактивних дашбордів. Інтерфейс Zabbix, натомість, реалізований як MPA (Multi Page Application), що відповідає його рівню як повноцінної корпоративної платформи з великою кількістю сторінок, модулів і функцій [11].

Налаштування Glances займає мінімальний час і не потребує складної конфігурації, що робить його ідеальним для швидкого старту. Prometheus з Grafana мають середню складність налаштування, особливо у випадках, коли потрібно

налаштувати взаємодію з іншими сервісами, проте це компенсується великою кількістю документації. У випадку Zabbix налаштування набагато складніше, особливо через необхідність використання шаблонів, SNMP-агентів та інтеграції з різними джерелами даних, що вимагає значного досвіду та планування [11].

У контексті контейнеризації, всі три рішення підтримують Docker, однак реалізація відрізняється за рівнем зручності: Glances можна запустити практично миттєво, Prometheus з Grafana потребують створення docker-compose або окремих конфігурацій, а Zabbix хоч і має готові Docker-образи, проте їх налаштування значно складніше через необхідність організувати роботу кількох взаємодіючих компонентів [11].

З погляду кривої навчання, Glances дуже легко освоїти, що дозволяє навіть новачкам швидко розпочати моніторинг ресурсів. Prometheus з Grafana вимагають базових знань про систему моніторингу, проте мають значно ширший функціонал і масштабованість. Zabbix потребує значного часу для вивчення всіх його функцій, налаштувань та шаблонів, що природно для рішень корпоративного рівня [11].

Таким чином, Glances — це легке у використанні рішення для швидкого розгортання локального моніторингу, Prometheus з Grafana — більш гнучка та масштабована система, яка вимагає певної інтеграції компонентів, а Zabbix орієнтований на великі інфраструктури з багатьма серверами, пристроями та необхідністю централізованого керування.

Сценарії застосування кожного рішення. У кожного з розглянутих рішень є свої сценарії застосування, що зумовлені їх архітектурою, функціональністю та складністю реалізації (див. рис. 1.13).



Рисунок 1.13 – Сценарії використання систем моніторингу

Glances є ідеальним для локального або точкового моніторингу, зокрема для відслідковування основних показників на одному вузлі або віртуальній машині без збереження історичних даних. Такий підхід часто використовується розробниками для швидкої перевірки навантаження на CPU, RAM, мережу під час тестування чи налагодження програм [12].

Prometheus з Grafana краще підходять для аналізу продуктивності у хмарних середовищах, Kubernetes, Docker-кластерах, а також у IoT-проєктах, де важливо агрегувати та візуалізувати дані з численних сенсорів. Система дозволяє збирати кастомні метрики, будувати графіки навантаження, затримок, кількості запитів до API тощо, що особливо корисно для DevOps-команд та аналітики сервісів у реальному часі [12].

Zabbix же створено для масштабованого моніторингу великої кількості компонентів корпоративної IT-інфраструктури: серверів, баз даних, мережевих пристроїв, служб. Система підтримує складні тригери, автоматичне сповіщення, побудову шаблонів моніторингу для різних типів обладнання, що дозволяє створити цілісну картину стану всіх елементів у межах великої організації. Наприклад, можна побудувати комплексну систему сповіщення про збої мережі,

виявлення аномалій у роботі сервісів чи оцінювання доступності ключових компонентів на рівні SLA [12].

Оцінка з точки зору простоти розгортання MVP. Оцінюючи технології з точки зору простоти розгортання MVP, особливу увагу варто приділяти тим рішенням, які дозволяють швидко створити функціонуючий прототип без необхідності глибокого конфігурування або складної інфраструктури, що дає змогу мінімізувати затрати часу на налаштування та зосередитися безпосередньо на реалізації функціональності, необхідної для демонстрації або початкового тестування концепції. Одним із прикладів такого підходу є використання Glances, який має низку суттєвих переваг у цьому контексті. Його інсталяція відбувається максимально просто: у випадку Python достатньо виконати команду pip install glances, а в середовищі Linux встановлення можливе через пакетний менеджер командою sudo apt install glances. Завдяки наявності вбудованого веб-сервера, який активується за допомогою параметра --web-server, можна одразу отримати доступ до базового інтерфейсу моніторингу через браузер без потреби в додатковому ПЗ. Крім того, Glances підтримує модель клієнт-сервер, що дозволяє моніторити віддалені вузли, що особливо корисно навіть у простих топологіях.

Однак варто зважати й на обмеження цього рішення — його масштабованість є дещо обмеженою, довготривале зберігання даних не реалізоване повною мірою, а отже, Glances доцільніше використовувати як засіб для оперативного огляду стану системи, а не як основу для тривалої аналітики. Проте як рішення для MVP Glances чудово виконує свою роль: дозволяє швидко зібрати і показати базову функціональність системи моніторингу, яку згодом можна поглиблювати або замінювати на більш масштабовані варіанти.

Іншим популярним підходом до реалізації MVP-рішення в сфері моніторингу є використання зв’язки Prometheus і Grafana, де Prometheus відповідає за збір метрик, а Grafana — за їх візуалізацію, що дає чіткий поділ обов’язків та дозволяє будувати більш складні, кастомізовані дашборди. Розгортання цієї зв’язки відбувається також доволі просто — не вручну, як у випадку з Glances, а через docker-compose, де після написання конфігураційного файлу можна запустити всю

систему одною командою. Також важливою перевагою є підтримка шаблонів, API-запитів і гнучких механізмів збору та обробки телеметрії. Проте на відміну від Glances, ця зв’язка вимагає певного досвіду: необхідне попереднє конфігурування джерел, налаштування дашбордів, розуміння мови запитів PromQL. Все це робить Prometheus зі Grafana менш придатними для демонстраційних цілей без попередньої підготовки, але водночас більш придатними для MVP у багатовузлових проектах, де важливими є гнучкість, аналітика та масштабованість.

У свою чергу, Zabbix — це рішення вже корпоративного рівня, яке постачається з великим набором функціональності «з коробки». Воно підтримує SNMP, безагентний моніторинг, шаблони, події, а також має потужну систему оповіщень. Але ці переваги мають свою ціну: встановлення Zabbix є складнішим, початкова конфігурація потребує більше часу, а сам процес масштабування — менш очевидний у порівнянні з попередніми рішеннями. Крім того, щоб повноцінно працювати з Zabbix, потрібно витратити більше часу на навчання, розуміння його логіки, інтерфейсів та правил роботи з тригерами і шаблонами. Саме тому Zabbix рідко розглядається як платформа для створення MVP, адже він більше підходить для повноцінного впровадження у стабільному, довготривалому середовищі, де вже сформульовані конкретні вимоги до масштабування, аналітики та безперебійності (див. рис. 1.14).

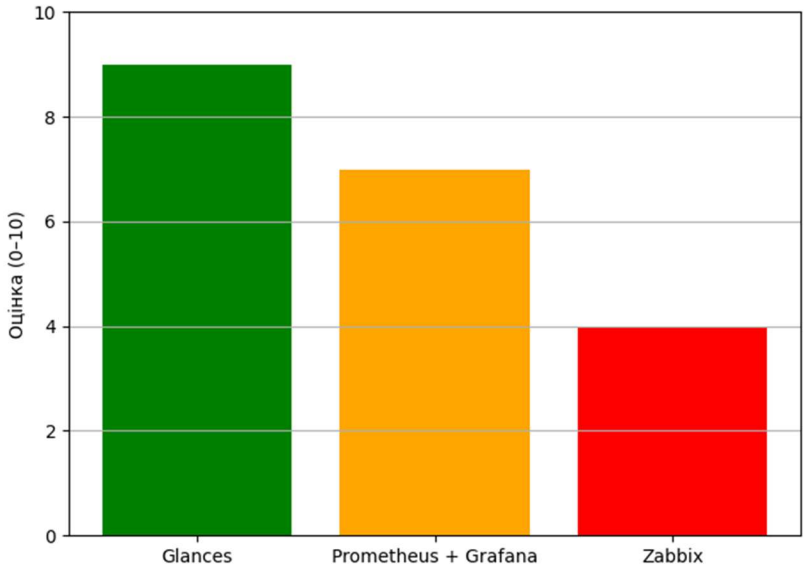


Рисунок 1.14 – Оцінка рішень за критерієм швидкості розгортання

Загалом, вибір платформи для реалізації системи моніторингу у вигляді MVP безпосередньо залежить від масштабів проєкту, доступних ресурсів, вимог до швидкості розгортання, складності аналізу даних та рівня готовності команди. Якщо головне завдання — швидко реалізувати базову функціональність і продемонструвати життєздатність ідеї, Glances буде оптимальним вибором, особливо для невеликих систем і початкової перевірки концепції. У випадку, коли потрібен баланс між функціональністю та складністю, і водночас є потреба у візуалізації з кількох джерел — зв'язка Prometheus зі Grafana підходить ідеально. Натомість Zabbix має сенс розгортати вже на більш зрілому етапі розвитку продукту, коли важливими стають стабільність, масштабування та повна інтеграція в інфраструктуру.

У рамках розглядуваної роботи було обрано саме Glances, оскільки це рішення дозволяє реалізувати мінімальну, але показову систему моніторингу, яка демонструє стан комп'ютера та окремих елементів IT-інфраструктури, будучи при цьому простим у використанні й легким для презентації. Згодом, за потреби, систему можна буде масштабувати або перевести на іншу платформу, з урахуванням майбутніх вимог до функціональності та продуктивності.

1.5 Вимоги до апаратно-програмного комплексу моніторингу

Проєктування системи моніторингу потребує глибокого розуміння вимог до апаратно-програмного комплексу, які визначають її ефективність у конкретному цільовому середовищі. Врахування таких вимог починається з формування чіткого технічного завдання, яке охоплює як функціональні, так і нефункціональні аспекти, а також апаратні обмеження і потенційні можливості системи. Таке завдання дозволяє спрогнозувати, які задачі зможе вирішувати майбутня система, на якій інфраструктурі її можна буде розгорнути та які обсяги даних вона зможе обробляти. Загальна структура вимог формується не лише з точки зору роботи

окремих компонентів, але й з урахуванням здатності до масштабування, гнучкості інтеграцій та адаптивності до змін середовища [13-14].

Функціональні вимоги до системи моніторингу. Функціональні вимоги безпосередньо визначають перелік задач, які повинна виконувати система моніторингу, та способи її взаємодії з користувачем і зовнішніми сервісами. Основою функціональності виступає підтримка збору телеметричних даних, що включає інформацію про завантаження центрального процесора, об'єм використаної оперативної пам'яті, дискову активність, швидкість читання-запису (IOPS), мережевий трафік та інші базові параметри стану системи. Крім цього, важливо забезпечити інтеграцію з додатковими джерелами даних, зокрема брокерами повідомлень або IP-пристроями, а також підтримку як пулінгових, так і пуш-механізмів збору інформації для забезпечення гнучкості у роботі з різними джерелами телеметрії [14].

Зібрані дані потребують відповідного зберігання та обробки, що реалізується через використання сховищ часових рядів (TSDB), де здійснюється агрегація метрик за заданими інтервалами — наприклад, середнє навантаження за хвилину — та реалізується автоматична ротація або архівація даних, що втратили актуальність. Після етапу збору та обробки метрик переходять до візуалізації даних, що реалізується у вигляді інтерактивного веб-інтерфейсу. Він повинен містити гнучкі дешборди, графіки, таблиці, індикатори стану, віджети та інші елементи, що забезпечують наочне представлення інформації. Особливої уваги потребує можливість масштабування часової шкали, фільтрації даних та налаштування зовнішнього вигляду елементів інтерфейсу відповідно до потреб користувача або адміністратора системи [14].

Одним із важливих функціональних блоків є підтримка системи сповіщення (алертингу), що забезпечує оперативне реагування на аномалії або критичні стани. В рамках цієї функції формується набір правил для створення тригерів, які активуються при досягненні визначених порогових значень. Повідомлення можуть надсилатися через різні канали — електронну пошту, Telegram, Slack або інші платформи — відповідно до обраної конфігурації. Також підтримується

можливість налаштування умов спрацювання та рівнів пріоритетності для більш гнучкого контролю подій у системі [14].

Інтеграційна здатність системи моніторингу реалізується через підтримку взаємодії з іншими сервісами, наприклад, за допомогою REST API, що дозволяє як приймати, так і віддавати дані зовнішнім платформам. Крім того, важливо забезпечити можливість експорту зібраних метрик у сторонні сховища типу Elasticsearch або Kafka, що може бути необхідно у разі масштабованих рішень або побудови складних аналітичних ланцюгів (див. рис. 1.15).

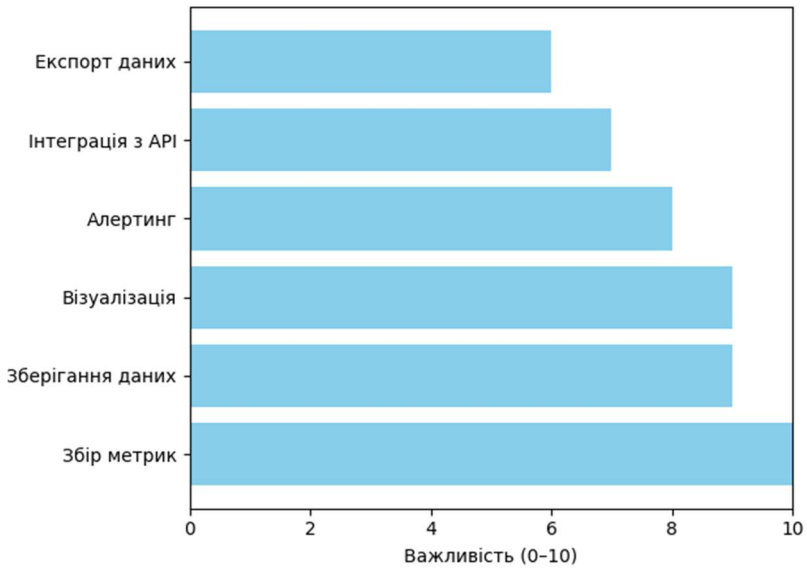


Рисунок 1.15 – Співвідношення важливості функціональних вимог до системи

Потенційна модульність системи моніторингу, яка дає змогу поступово розширювати її функціональність через додаткові плагіни або інтеграції, також належить до кола функціональних вимог, оскільки забезпечує її довготривалу життєздатність та здатність до адаптації в умовах зміни вимог або середовища [14].

Нефункціональні вимоги. Нефункціональні вимоги стосуються характеристик системи, що не пов’язані безпосередньо з її функціональністю, але мають критичне значення для її надійності, ефективності та загального користувацького досвіду. Вони охоплюють аспекти продуктивності, безпеки, надійності, масштабованості та зручності обслуговування. Продуктивність системи, зокрема, визначається такими критеріями, як час відгуку веб-інтерфейсу,

що має не перевищувати 500 мілісекунд для основних операцій, і здатність обробляти не менше 1000 повідомлень на секунду без втрати даних. У таких умовах важливо передбачити підтримку багатопотокової обробки та реалізацію розподіленої архітектури для забезпечення ефективного масштабування системи в разі зростання навантаження [14].

Крім продуктивності, однією з ключових вимог є безпека, що включає аутентифікацію пристроїв за допомогою таких механізмів, як локальні облікові записи, LDAP чи OAuth, а також обов'язкове шифрування трафіку між вузлами через HTTPS або TLS. Додатково необхідне впровадження ролей доступу — наприклад, для адміністратора, користувача та спостерігача — з метою чіткої організації прав і уникнення надлишкових дозволів. У цьому контексті не менш важливим є і журналювання дій користувачів та системних подій для контролю змін і виявлення потенційних загроз [14].

Серед інших нефункціональних вимог можна виділити надійність і доступність, що охоплюють можливість резервного копіювання баз даних для запобігання втраті критичної інформації та забезпечення відновлення без повного перезапуску системи. Виявлення збоїв компонентів через використання сервісів типу Watchdog дозволяє автоматично реагувати на критичні помилки, а підтримка механізмів автоматичного перезапуску або міграції сервісів у кластері забезпечує безперервність роботи навіть у випадку фізичних чи програмних збоїв [14].

Масштабованість реалізується через горизонтальне додавання нових вузлів і розділення ролей, коли функції збору, зберігання та візуалізації розносяться між різними серверами для підвищення гнучкості та стабільності системи. Серед відповідних механізмів важливе місце посідає кешування метрик і агрегація даних, що дозволяє ефективно обробляти великі обсяги інформації без надмірного навантаження на систему (див. рис. 1.16).

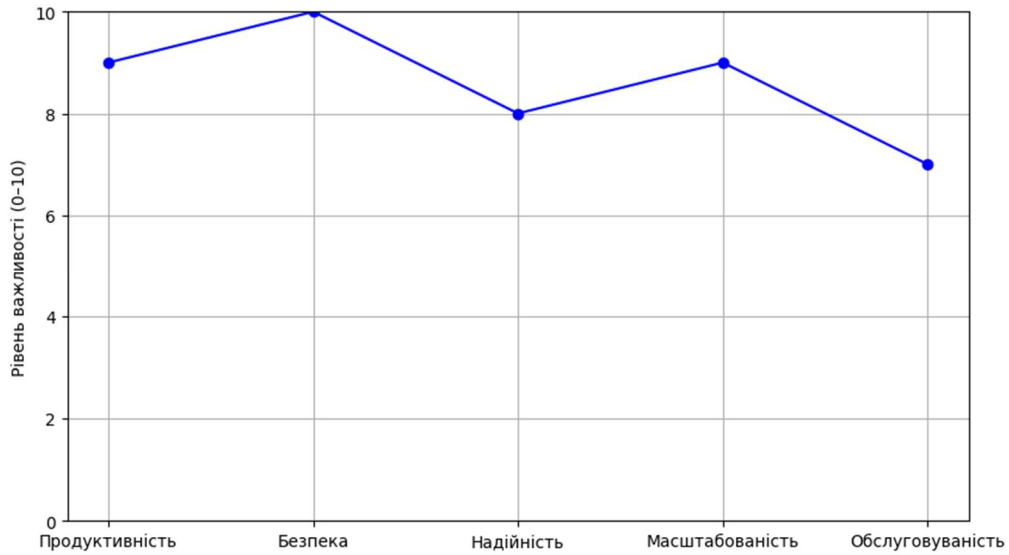


Рисунок 1.16 – Графік відповідності нефункціональних вимог

До зручності обслуговування, як до важливої нефункціональної вимоги, належать наявність офіційної документації, що описує процеси розгортання, налаштування й оновлення. Для цього можуть використовуватись Docker, Ansible або Helm — інструменти, що вже мають детальні керівництва в офіційних репозиторіях. Зручність також забезпечується через графічні інтерфейси для конфігурації та моніторингу стану системи, найчастіше у вигляді веб-інтерфейсів, а також через підтримку автоматичних оновлень компонентів або можливість ручного оновлення з мінімальними зусиллями з боку обслуговуючого персоналу [14].

Апаратні обмеження та припущення. При реалізації рішення у вигляді MVP або в рамках обмеженого середовища для тестування критично важливо враховувати апаратні обмеження та припущення. Зокрема, необхідно мати уявлення про мінімальні вимоги до ресурсів — процесора, оперативної пам’яті, мережі та дискового простору. Наприклад, використання агентів збору даних, таких як Node Exporter або Glances, створює невелике навантаження на процесор, приблизно 5%, тоді як Glances вимагає мінімум 256 МБ оперативної пам’яті. Prometheus оптимально працює при наявності 2 ГБ, хоча мінімально можливий запуск із 512 МБ, а Grafana — від 512 МБ. Таким чином, поєднане рішення Prometheus зі Grafana потребує близько 1–3 ГБ оперативної пам’яті, а Zabbix-сервер — від 2 до 4 ГБ, залежно від навантаження (див. рис. 1.17).

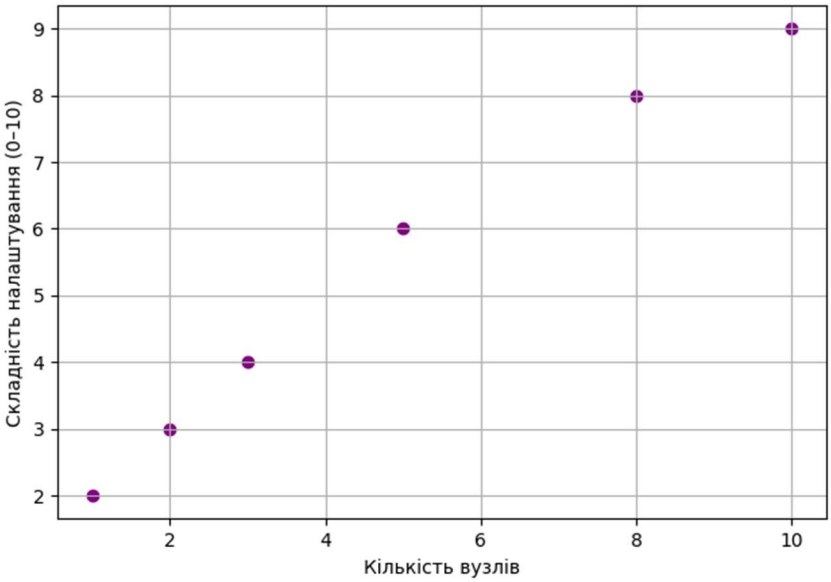


Рисунок 1.17 – Візуалізація взаємозв’язку між кількістю вузлів та складністю до налаштування

Обмеження також стосуються мережевої складової, оскільки для агентів і безагентного моніторингу за допомогою HTTP, HTTPS або SNMP потрібно забезпечити стабільне мережеве з’єднання, навіть якщо система працює в ізольованому локальному середовищі. У цьому ж контексті важливо передбачити можливість кешування метрик у разі втрати зв’язку з центральним сервером, що дозволяє уникнути втрати даних, забезпечивши тимчасове зберігання інформації на вузлах до моменту відновлення з’єднання [13].

Ще одним аспектом є вимоги до обсягу зберігання. Системи на кшталт Prometheus або Zabbix, які використовують Time Series Databases (TSDB), потребують суттєвого простору для зберігання історичних метрик. Залежно від навантаження, це може бути від 2 до 10 ГБ на місяць на кожен сервер збору. У такому випадку бажано використовувати SSD-накопичувачі, оскільки вони забезпечують вищу швидкість доступу до даних, порівняно з HDD, що позитивно впливає на загальну продуктивність системи [13].

Окрему увагу варто приділити сумісності, яка визначається тим, на яких операційних системах або платформах може бути розгорнуте рішення. Найпоширенішими є дистрибутиви Linux, зокрема Ubuntu та Debian, а також

Windows Server. Крім цього, важливою є можливість запуску компонентів у контейнерному середовищі (наприклад, Docker) або у віртуальних машинах (KVM, VirtualBox, VMware), що надає додаткову гнучкість при розгортанні та масштабуванні системи під наявну інфраструктуру [13].

Потенційний функціонал системи. До вищезазначених вимог, які висуваються до системи моніторингу, необхідно також враховувати потенційний функціонал, який формується як набір конкретних можливостей, здатних забезпечити її повноцінну роботу в умовах реального середовища. Якщо подивитися на це з практичного боку та умовно класифікувати за категоріями, першочергово розглядається збір даних — фактично це означає, що система повинна мати змогу здійснювати моніторинг вузлів, сервісів, датчиків, точок доступу, а також IoT-пристроїв, з яких надходить інформація. Ці дані, будучи зібраними, мають бути збережені в локальній базі, що у випадку використання Prometheus реалізується через вбудовану time-series базу, а в Zabbix — через підтримку баз на зразок MySQL чи PostgreSQL, або ж у кеші, якщо мова йде про полегшені реалізації без повноцінного сховища. Наявність локального зберігання метрик дає змогу не лише зберігати історичні дані, але й виконувати подальшу обробку без потреби в зовнішніх залежностях.

Наступним етапом логічно є візуалізація зібраної інформації, яка реалізується через дашборди. Вони можуть включати карти топології, гістограми, графіки навантаження, таблиці стану та інші віджети. Найбільш поширеним прикладом такої реалізації є Grafana, яка дозволяє створювати зручні візуальні панелі як для миттєвого аналізу, так і для виявлення довгострокових трендів. Утім, подібний функціонал притаманний також системам на кшталт Zabbix, Netdata, VictoriaMetrics та інших сумісних рішень, які мають відповідні інтерфейси виводу.

Окремою важливою функціональною складовою є система сповіщень, яка забезпечує оперативне інформування у випадку виявлення аномалій або перевищення заданих порогових значень. Сповіщення можуть надсилатися електронною поштою, через месенджери на зразок Telegram або Slack, або у вигляді SMS-оповіщення, залежно від критичності події. Така логіка реалізується

через тригери, що автоматично реагують на умови, визначені адміністратором чи системним архітектором, дозволяючи уникати постійного ручного моніторингу.

Функціонал безпеки в системі моніторингу полягає у впровадженні засобів автентифікації, авторизації, використанні HTTPS-протоколів для шифрування, а також у реалізації рольового доступу до елементів конфігурації та перегляду даних. Додатково варто враховувати можливість ведення журналу дій користувачів (audit log), що дозволяє забезпечити простежуваність змін і підвищити загальний рівень прозорості та контролю над процесами адміністрування.

Ще одним напрямом, що розширює потенційний функціонал, є можливість керування системою через веб-інтерфейс — такий підхід спрощує конфігурацію шаблонів, перегляд логів, додавання нових вузлів до моніторингу та оновлення параметрів. У випадку необхідності інтеграції з іншими програмними або апаратними рішеннями важливо передбачити підтримку API, наприклад REST API, що забезпечує взаємодію з зовнішніми системами керування, автоматизації або обробки подій. Такий функціонал особливо важливий при створенні гібридних або масштабованих інфраструктур із наявністю декількох типів платформ та різнорідних компонентів.

Таблиця 1.4 – Порівняння апаратних вимог для рішень моніторингу

Категорія	**Glances**	**Prometheus**	**Grafana**	**Zabbix-сервер**
Процесор	~5% використання	~5–10% при зборі/обробці	Низьке навантаження	Середнє (залежить від кількості вузлів/метрик)
Оперативна пам’ять	≥ 256 МБ	Мін. 512 МБ (рек. 2 ГБ)	Мін. 512 МБ	2–4 ГБ
Дисковий простір	Мінімальний (текстові логи)	2–10 ГБ/місяць (TSDB)	~100–300 МБ (без великих дашбордів)	2–10 ГБ/місяць (TSDB)

Продовження таблиці 1.4 – Порівняння апаратних вимог для рішень моніторингу

Категорія	**Glances**	**Prometheus**	**Grafana**	**Zabbix-сервер**
Мережеві вимоги	HTTP/HTTPS	HTTP / remote write	HTTP	SNMP, HTTP, агентні методи
Сумісність ОС	Linux, Windows	Linux (Ubuntu, Debian), Docker	Linux, Windows, Docker	Linux, Windows
Підтримка контейнерів	Так (Docker)	Так (Docker)	Так (Docker)	Частково (Docker + Agent)
Можливість роботи в ізоляції	Так	Так (із локальним retention)	Так (як інтерфейс до локального TSDB)	Так
Кешування при втраті зв’язку	Так (локальні метрики)	Частково (буфер збереження)	Залежить від TSDB	Так (агенти з буфером)

Узагальнюючи зазначене, формування чітких і раціональних вимог до такого апаратно-програмного комплексу (див. табл. 1.4) визначає загальний вектор проєктування та є критично важливим для успішної реалізації системи моніторингу будь-якого рівня складності. Функціональні можливості, які закладаються на етапі аналізу, мають забезпечити здатність системи не лише збирати, а й обробляти, зберігати, аналізувати й візуалізувати інформацію в режимі реального часу або з використанням історичних даних. На противагу їм, нефункціональні вимоги визначають рівень надійності, продуктивності, безпеки, масштабованості та

зручності використання системи в умовах обраної IT-інфраструктури, яка може мати певні апаратні обмеження. Такі обмеження, в свою чергу, впливають на архітектурні рішення і дозволяють обрати оптимальний набір інструментів та технологій відповідно до доступних ресурсів і очікуваної ефективності.

Формування переліку потенційного функціоналу, як і нефункціональних характеристик, допомагає створити цілісне бачення розвитку системи — від початкового MVP-продукту, що реалізує базові потреби, до повнофункціонального рішення, придатного до масштабування та застосування в реальних умовах. Аналіз, який проводиться на етапі підготовки, дозволяє аргументовано обґрунтувати вибір компонентів, передбачити витрати, визначити слабкі місця та сформувати стратегію розгортання з урахуванням потенційних оновлень або переходу на вищий рівень навантаження у майбутньому.

РОЗДІЛ 2 РОЗРОБКА АРХІТЕКТУРИ АПАРАТНО-ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Загальна архітектура системи моніторингу

Побудова ефективної системи моніторингу передбачає створення архітектурної моделі, що чітко визначає компоненти, їх функції та засоби взаємодії між собою, формуючи логічну структуру, яка забезпечує цілісність, узгодженість та гнучкість у розгортанні. В такій структурі мають бути детально описані ключові рівні — збір, обробка, зберігання та візуалізація, а також механізми обміну даними між елементами. Демонстрацію реалізації доцільно здійснювати на прикладі практичного рішення, зокрема Glances, який реалізується у вигляді агента, розгорнутого у контейнері Docker на відповідній платформі, що дозволяє створити продукт з мінімальною життєздатною реалізацією, придатний до подальшого розвитку [15-16].

Логічна структура системи моніторингу. Логічна структура типової системи моніторингу передбачає поділ на три функціональні рівні. Перший — рівень збору даних (Collection Layer), відповідає за відстеження та фіксацію параметрів із вузлів або фізичних і віртуальних пристроїв, що формує основу для наступних обчислень. Другий рівень — обробки та зберігання (Processing and Storage Layer), де зібрана телеметрія агрегується, аналізується та зберігається в системах збереження, таких як локальні бази даних або вбудовані сховища. Третій рівень — візуалізації та керування (Presentation and Management Layer), забезпечує користувацький інтерфейс для перегляду отриманих метрик, побудови візуальних уявлень, таких як графіки, таблиці або дашборди, а також інструменти адміністрування системи (див. рис. 2.1).



Рисунок 2.1 – Логічна структура IoT-системи моніторингу

Вибір Glances як агента збору метрик обумовлений його здатністю надавати зведену інформацію про ресурси системи у режимі реального часу, при цьому контейнерна реалізація через Docker забезпечує ізольоване та кероване середовище. Система розгортається в середовищі Docker Desktop на платформі Windows 11, яка підтримує Linux-підсистему WSL2 з базовим дистрибутивом Alpine, що дозволяє здійснювати доступ до контейнерів, керувати ними через командний рядок та виконувати необхідні налаштування. Дане середовище формує базу для повноцінної демонстрації всієї логіки взаємодії між компонентами системи моніторингу [15].

Рішення Glances, виконане у вигляді агента в контейнері, дозволяє безпосередньо реалізувати функції збору телеметрії, її обробки, доступу до веб-інтерфейсу для керування та перегляду метрик, а також інтеграції з іншими сервісами. Такий підхід забезпечує наочну реалізацію базової моделі, яка ілюструє повний життєвий цикл моніторингу — від збору до візуалізації, включаючи можливість створення власних дашбордів для зручного спостереження та аналізу стану системи. У результаті формується мінімальна, але функціональна модель, на основі якої можливо розширення та масштабування під конкретні потреби IT-інфраструктури [15].

Основні компоненти системи. Побудова моніторингової інфраструктури передбачає поетапне формування її архітектури з урахуванням логіки функціонування та взаємодії внутрішніх складових, де ключову роль виконує поділ на компоненти, кожен з яких реалізує окрему частину повного циклу моніторингу. Реалізація обрана на основі інструменту Glances, який розгортається у середовищі Docker Desktop на базі Windows 11, що використовує підсистему Windows

Subsystem for Linux (WSL) для емульованого запуску Linux-середовища, де безпосередньо розгортається контейнер із образом Alpine, всередині якого працює Glances. Таким чином, рішення Glances не працює як звичайний процес, що взаємодіє з операційною системою хосту, а виконується в ізольованому середовищі контейнера, що імітує автономний вузол із власними ресурсами, дозволяючи імітувати повноцінну роботу системи моніторингу у межах однієї машини.

Першим елементом у такій архітектурі є апаратна частина, яка в цьому випадку умовно представлена системою Windows 11, що виступає хостом для WSL та Docker Desktop, створюючи віртуальне середовище для запуску інструментів моніторингу. Наступним компонентом стає Агент збору даних, що втілений у вигляді Glances, розгорнутого в контейнері зі образом Alpine. Основна функція агента — це збирання телеметрії, включно з навантаженням на CPU, використанням оперативної пам’яті, інформацією про файлові системи, мережеву активність, активні процеси та загальний стан ОС, що дозволяє комплексно оцінювати поведінку об’єкта моніторингу. Glances у такій конфігурації працює у фоновому режимі як демон усередині контейнера, надаючи телеметрію через вбудований API або веб-інтерфейс.

Наступний логічний компонент — це сервер збору та зберігання, який у демонстраційній реалізації не представлений окремим модулем, оскільки його функції також виконує Glances, що одночасно виступає агентом, сервером збору та інтерфейсом візуалізації. Такий підхід реалізує концепцію MVP (продукту з мінімально необхідним функціоналом), що дозволяє у межах одного контейнера демонструвати повноцінну систему моніторингу без потреби у побудові розподіленої інфраструктури. Збереження даних у Glances реалізується через внутрішні механізми, які можуть агрегувати показники, зберігати часові ряди, обчислювати похідні метрики та надавати доступ до них через API для подальшого аналізу чи інтеграції з іншими платформами.

Останнім елементом архітектури є інтерфейс користувача, що в цьому випадку реалізований за допомогою вбудованого веб-інтерфейсу Glances (див. рис. 2.2).

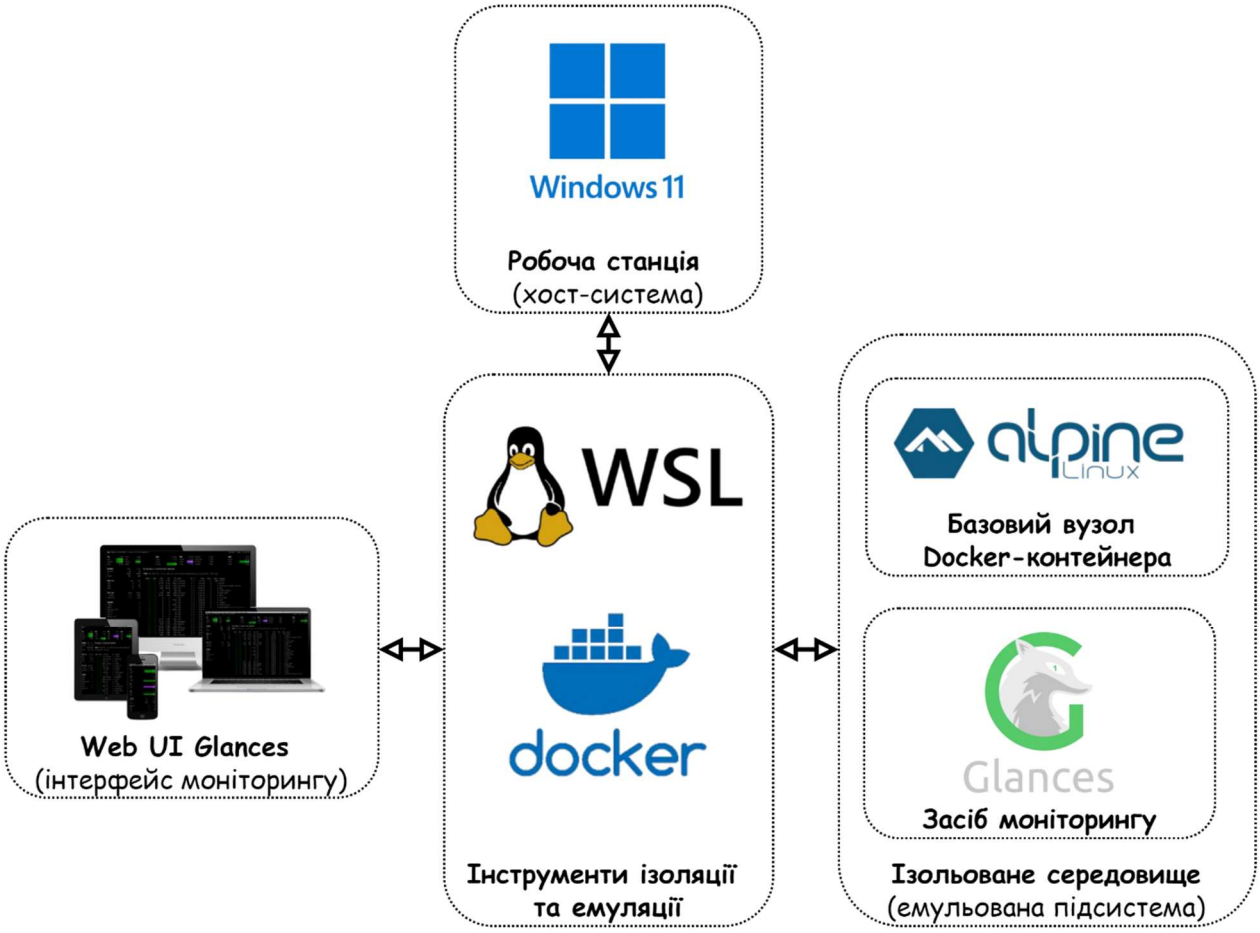


Рисунок 2.2 – Архітектура системи моніторингу комп’ютерних систем на базі Glances у Docker

Цей інтерфейс забезпечує доступ до телеметрії у зручному графічному вигляді, дозволяє будувати прості дешборди, переглядати графіки, отримувати табличні звіти та загалом відстежувати динаміку змін стану системи у реальному часі. Окрім перегляду, інтерфейс дає змогу налаштовувати параметри моніторингу, керувати параметрами доступу та, у розширених сценаріях, налаштовувати алерти (див. табл. 2.1).

Таблиця 2.1 – Основні компоненти архітектури системи моніторингу на базі Glances

Компонент	**Призначення**	**Технологія / інструмент**
Апаратна платформа	Хост-система, яка забезпечує фізичне або віртуальне середовище для запуску інфраструктури	Windows 11
Віртуалізаційне середовище	Запуск ізольованого середовища для контейнерів Linux та сервісів моніторингу	WSL2 зі Docker Desktop
Контейнерне середовище	Ізольований вузол моніторингу, що містить Glances та інші залежності	Docker (образ Alpine)
Агент збору даних	Збір телеметрії з ОС: CPU, RAM, диски, мережа, процеси	Glances (фоновий режим)
Збір даних	Отримання низькорівневої системної телеметрії	psutil (Python-бібліотека)
Передача даних	Обмін даними між модулями системи, доступ до API	Вбудований HTTP API (Glances RESTful API)
Зберігання даних	Локальне збереження та агрегування метрик, обчислення похідних значень	Внутрішні механізми Glances
Візуалізація	Графічне подання телеметрії, створення дешбордів	Веб-інтерфейс на базі Bottle Web Server
Інтерфейс користувача	Доступ до даних у браузері, налаштування моніторингу, перегляд графіків і таблиць у реальному часі	Web UI Glances (порт 61208/tcp)

Взаємодія між компонентами. Вся взаємодія між компонентами системи відбувається за схемою, де агент Glances періодично зчитує стан внутрішнього

середовища та надсилає метрики до вбудованого інтерфейсу, який відображає ці дані у реальному часі. Передача інформації може здійснюватись через REST, UDP, TCP або MQTT, однак у локальній конфігурації контейнера використовується вбудоване REST API без зовнішнього мережевого обміну. При перевищенні граничних значень або появі аномалій Glances може генерувати алерти, які будуть відображені у вебінтерфейсі, забезпечуючи мінімально необхідний рівень реакції на потенційні проблеми (див. рис. 2.3) [16].

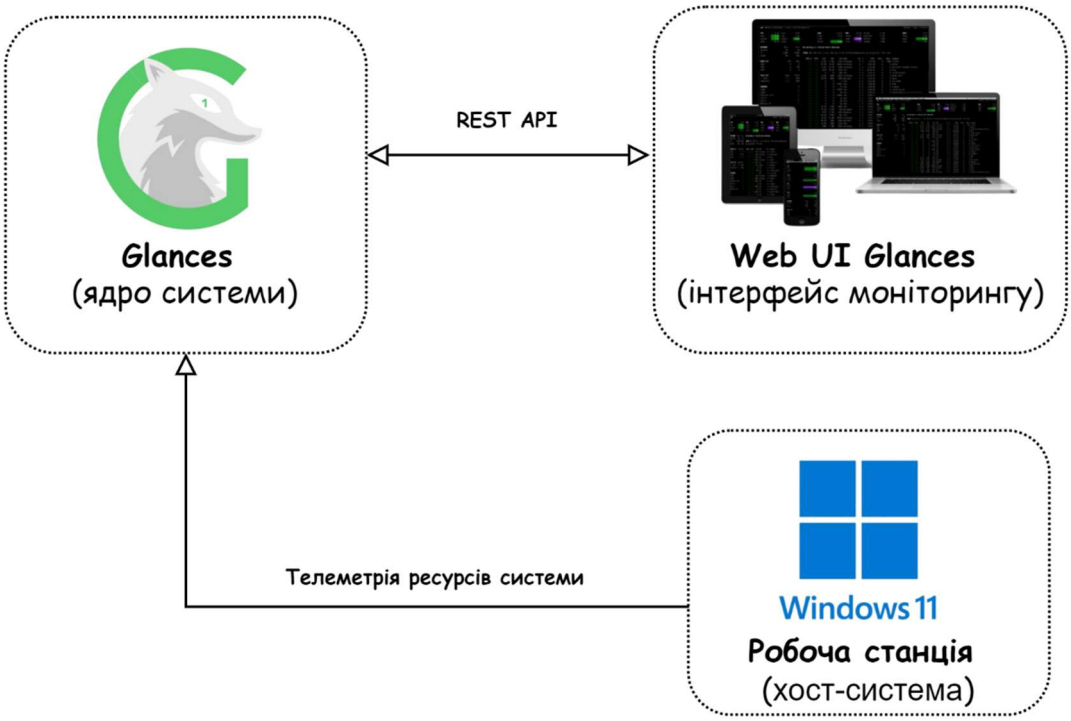


Рисунок 2.3 – Взаємодія між модулями системи

Узагальнюючи, всі основні компоненти — агент, сервер, інтерфейс — поєднані в одному контейнері, що дозволяє ефективно продемонструвати цикл моніторингу в умовах обмеженого середовища. Така структура підходить для ознайомлення з принципами роботи систем моніторингу, апробації базових функцій і подальшого масштабування в розподілене середовище із залученням повноцінних інструментів зберігання, візуалізації та реакції на події. Реалізація на базі Glances у контейнері Alpine створює універсальне, мобільне середовище для

перевірки працездатності концепції моніторингу, її гнучкості та потенціалу до подальшої інтеграції у більш складні архітектурні моделі [16].

Приклад реалізації архітектури з Glances у Docker. У розглянутому прикладі реалізації архітектури системи моніторингу із застосуванням Glances у Docker базовою платформою виступає хостова система, яка може бути будь-якою, зокрема в даному випадку використовується Windows 11. Управління здійснюється за допомогою кросплатформенного інструмента Glances, що функціонує як моніторинговий сервіс із можливістю роботи в режимах клієнт-сервер, термінального інтерфейсу та веб-інтерфейсу, підтримує REST API, має розширення для графічного виводу метрик через InfluxDB, однак у межах поточної реалізації ці функції не задіюються, а акцент робиться на базовій CLI-візуалізації, веб-доступі та серверній моделі. Обраний інструмент відповідає мінімальним вимогам щодо системних ресурсів, що важливо для розгортання на слабких вузлах або при обмеженнях інфраструктури. Сам процес реалізації виконується за допомогою контейнеризації, що дозволяє створити легку, швидко придатну до запуску конфігурацію, описану у вигляді YAML-файлу, в якому визначено образ останньої версії Glances, назву контейнера, політику перезапуску, мережевий режим (host), підключення до сокета Docker для збору додаткових даних про контейнери, а також середовище запуску з параметром GLANCES_OPT=-w, що активує веб-режим.

Таким чином, створюється контейнер, який збирає метрики з хост-системи, відкриває доступ до веб-інтерфейсу на порту 61208 і забезпечує простий браузерний перегляд даних про стан системи. Така реалізація є прикладом архітектури MVP (Minimum Viable Product), тобто найпростішого варіанту робочого рішення, що включає у собі одночасно і агента збору, і сервер обробки, і клієнтський інтерфейс в одному контейнері. Подібна спрощена модель дозволяє максимально швидко та ефективно розгорнути систему моніторингу для демонстрації або первинного тестування без необхідності налаштування зовнішніх баз даних чи окремих інтерфейсів. Дані не зберігаються довготривало, однак за потреби можуть бути експортовані у зовнішні системи для подальшого аналізу. Завдяки малій вимогливості до ресурсів Glances здатен працювати навіть на

системах із 256–512 МБ оперативної пам’яті, що робить його зручним для впровадження на віддалених або малопотужних вузлах.

Переваги та обмеження такої архітектури. Розглядаючи переваги та обмеження такої архітектури, варто зазначити, що її головна сильна сторона полягає в надзвичайно швидкому розгортанні за рахунок використання контейнерного підходу, що дозволяє підняти систему моніторингу буквально за хвилини без складної конфігурації. Однак такий підхід водночас і обмежує масштабованість: хоча Glances підтримує інтеграцію з InfluxDB або Prometheus для збереження історичних метрик, поточна реалізація їх не використовує, що робить її менш придатною для великих розподілених систем. У класичній інфраструктурі Prometheus зазвичай використовує node_exporter як основного агента збору, тоді як Glances краще поєднується з InfluxDB. Попри це, інтеграція з будь-якими з цих компонентів можлива, хоча й вимагає додаткового налаштування та виходить за межі демонстраційної реалізації. Веб-інтерфейс Glances є легким, мобільним, зручним у використанні, не створює надмірного навантаження, і може запускатись на практично будь-якому комп’ютері навіть із мінімальними апаратними характеристиками.

Підсумовуючи, така архітектура моніторингу системи на основі Glances у Docker становить собою компактне, просте та ефективне рішення з трьома ключовими компонентами – агентом збору, сервером обробки та інтерфейсом користувача – реалізованими у межах одного контейнера, що ілюструє ідею програмного продукту з мінімальним життєвим циклом. Завдяки цьому її можна швидко запустити в середовищах із обмеженими ресурсами, використовувати як демонстраційний прототип або MVP-рішення, а при необхідності розширити функціональність за рахунок зовнішніх сервісів та баз даних, перетворивши на повноцінну систему моніторингу. Такий підхід особливо цінний в умовах обмежених інфраструктурних можливостей, польових розгортань або при початковому проектуванні майбутніх розподілених систем.

2.2 Вибір та конфігурація інструментів

Для побудови мінімально життєздатного прототипу (MVP) системи моніторингу було обрано інструмент Glances, який раніше вже розглядався як засіб, що дозволяє швидко та ефективно отримувати основні метрики стану системи. Його функціональність включає можливість моніторингу як хостової системи, так і внутрішніх Docker-контейнерів завдяки використанню змонтованого сокету Docker. Такий вибір обумовлений потребою у демонстрації базових можливостей контролю комп'ютерних систем у спрощеній, але наочній формі без зайвої складності в розгортанні або налаштуванні. Враховуючи вимоги до портативності, ізольованості середовища та мінімального впливу на систему, запуск Glances у вигляді Docker-контейнера виявився найбільш придатним варіантом. Дане рішення забезпечує незалежність від операційної системи, легкість перенесення між середовищами та простоту ініціалізації, що критично важливо для експериментальних і демонстраційних цілей [17-19].

Встановлення та запуск Glances у контейнері Docker. Glances підтримує кілька способів розгортання, зокрема встановлення через pip, використання у вигляді системної служби або розміщення в контейнері. З огляду на потребу в простоті реалізації без глибокого втручання у середовище хоста, було застосовано саме контейнерний підхід. Запуск Glances у Docker відбувається за допомогою однієї команди, яка створює і конфігурує контейнер (див. рис. 2.4).

```
docker run -d \
  --name glances \
  --restart always \
  --pid host \
  --network host \
  -v /var/run/docker.sock:/var/run/docker.sock:ro \
  nicolargo/glances:latest \
  -w
```

Рисунок 2.4 – Ручна команда запуску Glances у Docker

Ця команда забезпечує відображення всіх процесів на хості за рахунок використання --pid host, прямий доступ до мережі через --network host, а параметр -v /var/run/docker.sock:/var/run/docker.sock:ro дозволяє Glances отримувати інформацію про всі запущені контейнери Docker. Додавання -w активує вбудований веб-інтерфейс, який можна відкрити у браузері за IP-адресою хосту з портом 61208. Така конфігурація не потребує попередньої підготовки або наявності окремого конфігураційного файлу, що спрощує використання в умовах обмеженого часу або ресурсів [17].

Як альтернативу ручному запуску через docker run, можливо скористатись структурованим підходом із використанням docker-compose.yml, що дозволяє зберігати опис контейнера у вигляді конфігураційного файлу. У такому випадку запускається Glances у схожому режимі з усіма потрібними параметрами (див. рис. 2.5) [17].

```yaml
version: '3'
services:
  glances:
    image: nicolargo/glances:latest
    container_name: glances
    restart: always
    pid: "host"
    network_mode: "host"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
    environment:
      - GLANCES_OPT=-w
```

Рисунок 2.5 – Файл конфігурації для автоматичного розгортання Glances у Docker

Конфігурація порту для Web-доступу. Для реалізації демонстраційної версії мінімально життєздатного прототипу (MVP) системи моніторингу було обрано Glances — інструмент, що дозволяє оперативно отримувати базові метрики системного стану, включаючи навантаження на процесор, пам’ять, мережу, використання дисків і навіть активність Docker-контейнерів. Обґрунтування вибору саме Glances ґрунтується на його гнучкості, швидкості налаштування, доступності веб-інтерфейсу та можливості запуску в контейнеризованому

середовищі, що забезпечує зручну ізоляцію та портативність. Замість встановлення як системної служби або через пакетний менеджер, було застосовано варіант розгортання Glances у Docker-контейнері, що дозволяє уникнути прив'язки до конкретної ОС хоста та полегшує перенесення рішення на інші системи. Запуск контейнера Glances здійснюється однією командою, яка включає параметри доступу до системних процесів, мережевих інтерфейсів та сокету Docker, що забезпечує прямий моніторинг і доступ до інформації про всі активні процеси як хостової системи, так і запущених у ній контейнерів. Параметр --pid host відкриває доступ до процесів хоста, --network host забезпечує прямий мережевий доступ без додаткових перенаправлень портів, а змонтований у режимі тільки для читання сокет /var/run/docker.sock дозволяє Glances бачити інтерфейси та статус кожного контейнера (див. рис. 2.6) [18].

```
-p 61208:61208

environment:
  - GLANCES_OPT=-w -p 8080
```

Рисунок 2.6 – Команда для налаштування відкритого порту для Web-доступу

Альтернативно до ручного запуску також використовується підхід з використанням конфігураційного файлу docker-compose.yml, що дозволяє краще структурувати параметри запуску. У такій конфігурації задається образ Glances, ім'я контейнера, параметри перезапуску, мережевий режим, змонтовані томи та середовище виконання, зокрема опція GLANCES_OPT=-w, яка активує веб-інтерфейс. Запуск у цьому випадку також здійснюється однією командою — docker-compose up -d, після чого інтерфейс Glances стає доступним у браузері за IP-адресою хоста та портом 61208, що використовується за замовченням. Якщо Docker працює у bridge-режимі, для доступу до веб-інтерфейсу необхідно явно пробросити порт за допомогою параметра -p 61208:61208, але в більшості випадків зручніше використовувати режим host, що спрощує доступ до всіх мережевих інтерфейсів системи без потреби в додаткових налаштуваннях, особливо коли Glances

використовується у внутрішній мережі або на сервері з фіксованою IP-адресою. У разі потреби порт також можна змінити, задавши нове значення через змінну середовища, наприклад GLANCES_OPT=-w -p 8080, після чого інтерфейс буде доступний на відповідному порту (див. табл. 2.2) [18].

Таблиця 2.2 – Налаштування Glances у контейнері Docker

Параметр	**Значення**	**Призначення**
PORT	61208	Порт Web UI для доступу до інтерфейсу Glances
--docker	true	Увімкнення моніторингу Docker-контейнерів через Docker socket
--export	csv	Експорт метрик у форматі CSV
--pid	host	Доступ до процесів хост-системи
--network	host	Прямий доступ до мережевих інтерфейсів без пробросу портів
-v /var/run/docker.sock:/...:ro	/var/run/docker.sock:/var/run/docker.sock:ro	Монтаж Docker socket у режимі "тільки для читання"
-w	активовано	Увімкнення вбудованого веб-інтерфейсу Glances
GLANCES_OPT	-w -p 8080 (за потреби)	Альтернативне задання параметрів запуску через змінну середовища
image	nicolargo/glances:latest	Docker-образ для запуску Glances
restart	always (у docker-compose)	Автоматичне відновлення роботи контейнера після перезавантаження
PORT	61208	Порт Web UI для доступу до інтерфейсу Glances

Моніторинг контейнерів через Docker socket. Інструмент також підтримує моніторинг Docker-контейнерів через зчитування інформації з Docker API, що доступний через сокет /var/run/docker.sock. При монтуванні цього сокета в контейнер Glances отримує повний доступ до API Docker, що дозволяє без встановлення додаткових агентів у кожен окремий контейнер збирати інформацію про їхній статус, використання процесора та оперативної пам'яті, мережеву активність тощо. У цьому випадку система отримує єдину точку збору даних, з якої можна бачити як сам хост, так і всі контейнери одночасно, без потреби в ручній конфігурації кожного компонента окремо. Такий підхід дає змогу централізовано контролювати всю інфраструктуру та знижує ризики безпеки, оскільки доступ до сокета Docker обмежується режимом лише для читання, що виключає можливість втручання у роботу контейнерів. Відповідні дані про активні контейнери, їх ресурси та ідентифікатори виводяться у вкладці Docker у веб-інтерфейсі Glances, що робить рішення зручним для демонстрації можливостей моніторингу як на рівні хост-системи, так і всередині ізольованих контейнерів без зайвих ускладнень [19].

Обґрунтування вибору Glances. Вибір Glances як базового інструмента моніторингу для прототипу був зумовлений низкою технічних, організаційних і практичних міркувань, частково розглянутих раніше, однак потребує окремого уточнення. Головною перевагою цього інструмента стала швидкість розгортання: Glances не вимагає складної конфігурації чи тривалих попередніх налаштувань, а достатньо виконання лише однієї команди docker run або запуску попередньо підготовленого конфігураційного файлу через docker compose up -d, що фактично забезпечує миттєвий запуск системи без додаткових дій. Така властивість особливо цінна в умовах тестування або розгортання мінімального життєздатного продукту (MVP) на різних вузлах без значних витрат часу та ресурсів.

Glances також демонструє надзвичайно низькі вимоги до апаратного забезпечення, споживаючи суттєво менше ресурсів, ніж більшість аналогічних систем, що дозволяє ефективно працювати навіть на застарілих пристроях або віртуальних машинах з обмеженим обсягом оперативної пам'яті, де інші сучасні моніторингові рішення можуть просто не запускатися або спричиняти

нестабільність. Додатковою перевагою є наявність вбудованого вебінтерфейсу, який, на відміну від Prometheus чи Zabbix, не потребує налаштування зовнішніх візуалізаційних панелей типу Grafana, оскільки має інтерактивну вебпанель з автооновленням даних, доступну одразу після запуску без необхідності додаткових модулів чи конфігурацій.

Серед ключових аргументів на користь Glances вирізняється вбудована підтримка моніторингу Docker-контейнерів, яка дозволяє здійснювати нагляд за всіма контейнерами та хостом одночасно, без необхідності в окремих рішеннях на кшталт cAdvisor або Netdata. Достатньо надати доступ до сокета Docker (/var/run/docker.sock), щоб Glances автоматично зчитував інформацію про всі запущені контейнери, їх навантаження на CPU, споживання пам’яті, мережеву активність та інші метрики, і при цьому не потребував встановлення жодних агентів у кожному контейнері окремо, що значно спрощує реалізацію централізованого моніторингу.

Додатковим чинником стала гнучкість Glances у частині масштабування: підтримка експорту метрик у такі системи як InfluxDB, Prometheus, MQTT та інші дозволяє у подальшому нарощувати можливості без повної перебудови архітектури рішення. Така інтеграційна спроможність відкриває шлях до побудови складніших систем моніторингу, заснованих на вже напрацьованій базі, і не вимагає відмови від існуючих підходів у разі масштабування.

Інтерфейс Glances також інтуїтивно зрозумілий, а логіка його роботи — прозора, що забезпечує зручність взаємодії навіть для тих, хто лише починає знайомство з моніторинговими системами, і дозволяє без зайвих труднощів здійснювати демонстрацію функціональності в рамках експериментального або дослідницького середовища. У підсумку це рішення виявляється оптимальним для швидкого запуску, мінімального конфігурування, зручного перегляду телеметрії в режимі реального часу і тестування базових функцій моніторингу навіть у середовищах з обмеженими ресурсами.

Саме ця сукупність переваг — простота розгортання, нативний вебінтерфейс, гнучкість, підтримка Docker, відсутність потреби у додаткових агентах, сумісність

з системами збереження метрик — і дозволила ефективно реалізувати систему спостереження за станом вузлів у даному проєкті, зокрема в умовах використання контейнерної інфраструктури, де збір телеметрії можливий з усіх вузлів одразу, фактично за лічені хвилини після розгортання.

2.3 Алгоритм роботи системи моніторингу

Обрана система моніторингу Glances забезпечує постійне спостереження за станом хосту та контейнерів, що працюють у його середовищі, автоматично збираючи, обробляючи та відображаючи інформацію в інтерактивному веб-інтерфейсі. У цьому процесі важливо простежити, як відбувається повний цикл її роботи: починаючи з ініціалізації всіх необхідних компонентів після запуску, через послідовний процес збору та оновлення метрик, аж до їх представлення у зручному графічному середовищі прямо в браузері [20-21].

Послідовність дій з моменту запуску системи. Після запуску Glances у режимі веб-сервера через команду glances -w система проходить низку послідовних етапів ініціалізації. Найпершим кроком є ініціалізація ядра програми: активуються основні модулі, включаючи збирач системних метрик glances_stats, веб-сервер на основі Bottle або Tornado, а також таймери для періодичного оновлення даних. Далі система автоматично виявляє та активує наявні сенсори й плагіни відповідно до особливостей поточного середовища. Наприклад, при наявності доступу до сокета Docker автоматично активується плагін для роботи з контейнерами, що дозволяє здійснювати моніторинг у реальному часі без додаткових налаштувань або встановлення окремих агентів у самі контейнери (див. рис. 2.7).

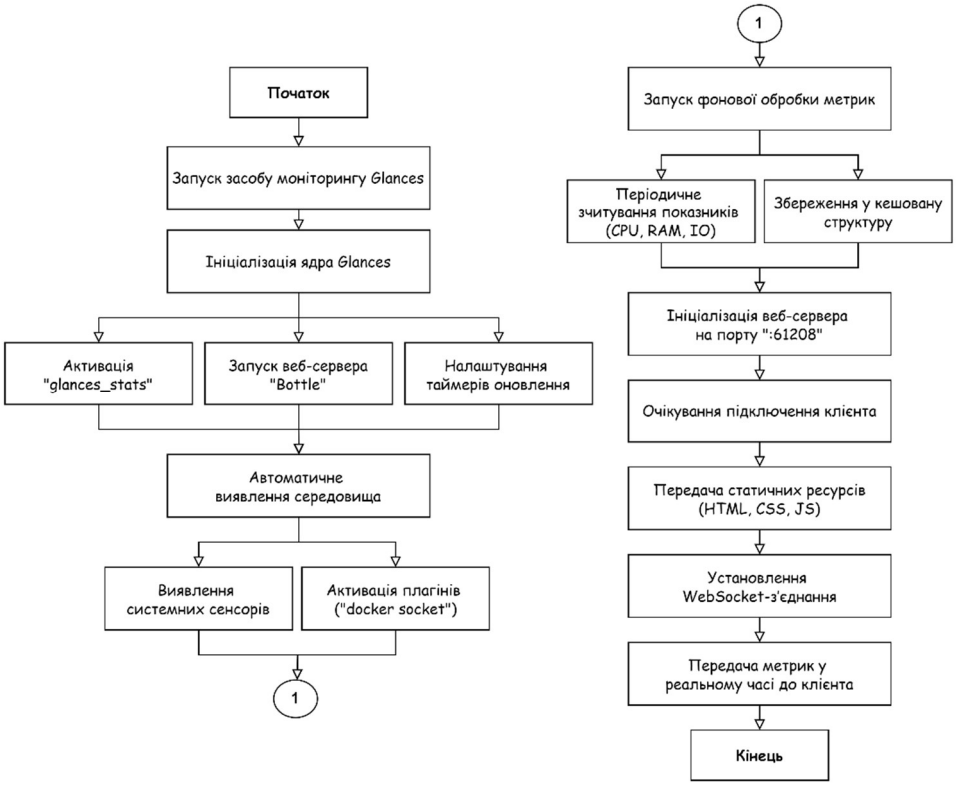


Рисунок 2.7 – Алгоритм запуску та ініціалізації системи

У наступному кроці запускається фоновий цикл збору даних, у межах якого з певним інтервалом, за замовчуванням — кожну секунду, відбувається опитування всіх активованих плагінів, зчитування поточних метрик та їх збереження у кешовану область пам’яті. Паралельно із цим ініціалізується веб-сервер, який починає слухати HTTP-запити на стандартному порту 61208, очікуючи підключення з боку клієнта — браузера на тій самій або іншій машині в мережі.

Після встановлення з’єднання між клієнтом і сервером система надсилає у браузер необхідні статичні файли — HTML-розмітку, стилі та скрипти JavaScript — і переходить у режим активної взаємодії через WebSocket. Саме через цей канал у реальному часі надходять оновлені метрики, що формуються на основі поточного стану хосту та всіх доступних контейнерів. За такої побудови архітектури відпадає потреба в дублюванні функціоналу або встановленні додаткових інструментів для збору телеметрії на кожен елемент інфраструктури, що робить реалізацію ефективною навіть у межах обмежених обчислювальних ресурсів.

Логіка зчитування даних через psutil. Одним із базових інструментів для збору системних метрик у межах реалізації моніторингового рішення на основі Glances є бібліотека psutil, яка забезпечує кросплатформену взаємодію з ресурсами операційної системи та дозволяє отримувати в реальному часі детальні дані про стан процесора, пам’яті, дисків, мережевих інтерфейсів, запущених процесів тощо. У межах такої взаємодії з системним середовищем можна, наприклад, зчитувати рівень завантаження процесорних ядер як окремо, так і сумарно, отримувати обсяг використаної оперативної пам’яті або її кешованої частини, а також дані про стан дискових систем, включаючи кількість операцій читання й запису, активність у байтах та загальне навантаження на файлові системи. Крім того, доступними є і метрики мережі, такі як обсяг отриманого та переданого трафіку, IP-адреси, назви інтерфейсів і швидкість з’єднання, а також інформація про окремі процеси — список найактивніших із них за використанням CPU чи RAM, PID, статуси та ієрархія. Усе це дозволяє Glances формувати повноцінну картину навантаження системи без встановлення окремих драйверів або залежностей [20].

У випадку реалізації інтеграції безпосередньо через Python можна виконати виклики до psutil, які повертають відповідні об’єкти для подальшого аналізу або кешування (див. рис. 2.8).

```
import psutil
cpu_percent = psutil.cpu_percent(interval=None)
mem_info = psutil.virtual_memory()
disk_io = psutil.disk_io_counters()
net_io = psutil.net_io_counters()
```

Рисунок 2.8 – Фрагмент коду для читання системних метрик через psutil

Ці об’єкти в подальшому можуть бути перетворені у звичні словники Python, оновлювані у фоновому циклі Glances і доступні в рамках веб-інтерфейсу без додаткового звернення до системних API при кожному запиті, що оптимізує час реакції та зменшує навантаження на хост [20].

Формування та передача даних у Web UI. Після отримання і кешування актуальних даних наступним етапом є формування структури для передачі у веб-інтерфейс. В основі обміну між серверною частиною Glances і браузером лежить

WebSocket-з’єднання, що дозволяє надсилати динамічні оновлення метрик у режимі реального часу без потреби оновлення всієї сторінки. Кожні одну-дві секунди сервер формує JSON-структуру, яка містить поточні значення метрик (див. рис. 2.9).

```
{
  "cpu": {"total": 17.3},
  "mem": {"used": 2048, "total": 8192},
  "diskio": {"read_bytes": 12345678},
  "net": {"eth0": {"rx": 234567, "tx": 345678}}
}
```

Рисунок 2.9 – Формування JSON-даних для Web-відображення

Ця структура передається у клієнтську частину безпосередньо через постійне WebSocket-з’єднання, що суттєво знижує затримки та прибирає необхідність повторних HTTP-запитів чи оновлень HTML-сторінки [21].

На стороні браузера працює JavaScript-клієнт, який отримує повідомлення, розпарсює JSON, інтерпретує його вміст і оновлює відповідні DOM-елементи на сторінці, що дозволяє досягти динамічної реактивної візуалізації, де всі зміни стану системи одразу відображаються у відповідних візуальних блоках — графіках, панелях, табличках — без помітних затримок або навантаження на користувацький пристрій. Завдяки такій архітектурі забезпечується стабільна, постійна передача актуальних даних із системи у веб-інтерфейс, що дозволяє виявляти проблеми на ранньому етапі, оперативно реагувати на зміну навантаження та тримати під контролем як фізичні, так і віртуалізовані ресурси [21].

Життєвий цикл даних – збір, обробка, візуалізація. Загальний життєвий цикл даних у межах реалізації базового MVP-рішення з моніторингу системних ресурсів розпочинається зі збору метрик, ініціалізованого безпосередньо Glances, який викликає відповідні функції бібліотеки psutil, зчитуючи значення центрального процесора, оперативної пам’яті, дискової підсистеми, мережевих інтерфейсів та активних процесів. Після цього зібрані значення переходять до наступного етапу, пов’язаного з обробкою — у ході цієї фази дані можуть округлюватися, фільтруватися відповідно до пріоритетів, нормалізуватися або конвертуватися до

уніфікованих типів, після чого формується структурований словник, що містить лише актуальні активні метрики. Цей словник зберігається у проміжному кеш-об’єкті в оперативній пам’яті, що дозволяє уникнути повторного зчитування та забезпечує миттєвий доступ до даних без додаткових затримок або системних викликів.

На наступному етапі формується відповідь для кожного клієнта — створюється окрема JSON-структура, яка включає повний набір або лише запитані категорії метрик, залежно від контексту запиту. Така відповідь передається клієнтам через WebSocket-з’єднання, що дозволяє уникнути традиційного пулінгу, характерного для REST-архітектури, і забезпечує стабільну доставку оновлень у реальному часі з мінімальними затримками. Одразу після передачі відбувається візуалізація — браузерний інтерфейс приймає дані через встановлене з’єднання, обробляє отримані JSON-пакети засобами JavaScript і миттєво оновлює DOM-елементи: відображаються графіки навантаження на CPU, RAM, трафік мережевих інтерфейсів, таблиці процесів і стовпчики, що відображають використання дисків, кожні дві секунди оновлюючи стан усього інтерфейсу без повного перезавантаження сторінки (див. рис. 2.10).

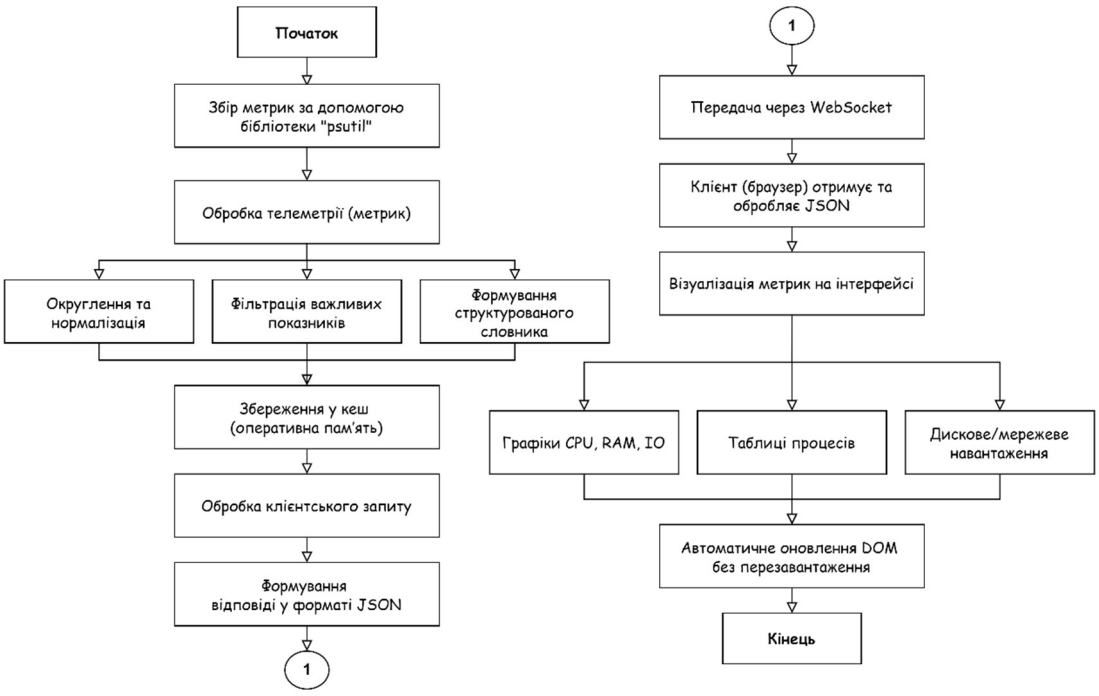


Рисунок 2.10 – Алгоритм обробки даних у системі

Цикл виконується безперервно з мінімальним інтервалом між ітераціями, створюючи ілюзію повної синхронізації з поточним станом системи, що дозволяє розглядати навіть найдрібніші коливання навантаження. Такий підхід є надзвичайно ефективним для моніторингу у реальному часі, особливо в умовах обмежених ресурсів. Завдяки комбінації легкого контейнеризованого розгортання, зчитування метрик за допомогою psutil, передачі через WebSocket та фронтенду на HTML5 можлива реалізація продуктивної системи спостереження навіть у застарілій інфраструктурі, де використання важковагових рішень було б недоцільним або неможливим.

Загальна логіка життєвого циклу, сформована таким чином, забезпечує повний ланцюг — від низькорівневого збору до візуального представлення даних — і підходить не лише для демонстраційних або навчальних цілей, але й для спрощених промислових систем, де важлива стабільність, прозорість та відсутність надмірної залежності від складного програмного середовища. У випадках, коли мова йде про моніторинг у середовищах з обмеженим апаратним забезпеченням або в інфраструктурі, що не оновлювалася десятки років, така модель забезпечує ефективне рішення без необхідності в дорогих оновленнях або складному налаштуванні, залишаючись доступною й адаптивною навіть у нестабільному або фрагментованому IT-середовищі.

2.4 Побудова мінімального MVP

З метою практичного підтвердження працездатності запропонованої архітектури моніторингу для комп’ютерних систем було реалізовано мінімально життєздатний продукт (MVP), побудований на основі інструменту Glances, який розгорнуто в середовищі контейнеризації Docker. Такий підхід дозволяє швидко створити функціональний прототип із повною підтримкою базових функцій моніторингу без складних налаштувань, встановлення додаткових пакетів чи

залежностей, що властиві багатьом традиційним рішенням. Розгортання контейнера з Glances дозволяє зосередитися на демонстрації ключових можливостей — зчитуванні та візуалізації метрик у реальному часі через вбудований веб-інтерфейс, а також відкриває перспективу подальшого масштабування системи, включно з підключенням додаткових апаратних сенсорів, наприклад температурних датчиків або сенсорів вологості, що дозволяє здійснювати не лише моніторинг програмно-апаратного стану системи, а й отримувати дані фізичного середовища, що у звичайних інструментах на зразок Prometheus або Zabbix часто потребує складного окремого налаштування або додаткових вузлів збору [22-23].

Запуск Glances у Docker. В основі реалізації MVP використовується офіційний Docker-образ nicolargo/glances:latest, доступний на Docker Hub, який вже включає всі необхідні залежності, включно з бібліотекою psutil для збору системних метрик, а також модулем для інтеграції з контейнерним середовищем. Запуск Glances у режимі веб-сервера виконується через просту команду Docker, яка створює контейнер у фоновому режимі, надає йому ім’я, пробросує порт 61208 для доступу до веб-інтерфейсу, монтує сокет Docker для читання інформації про активні контейнери на хості, а також дозволяє доступ до PID-простору хоста для перегляду всіх активних процесів. Така команда забезпечує миттєве розгортання повністю функціонального моніторингового вузла з інтеграцією на рівні операційної системи без встановлення Glances безпосередньо на саму хост-машину (див. рис. 2.11) [23].

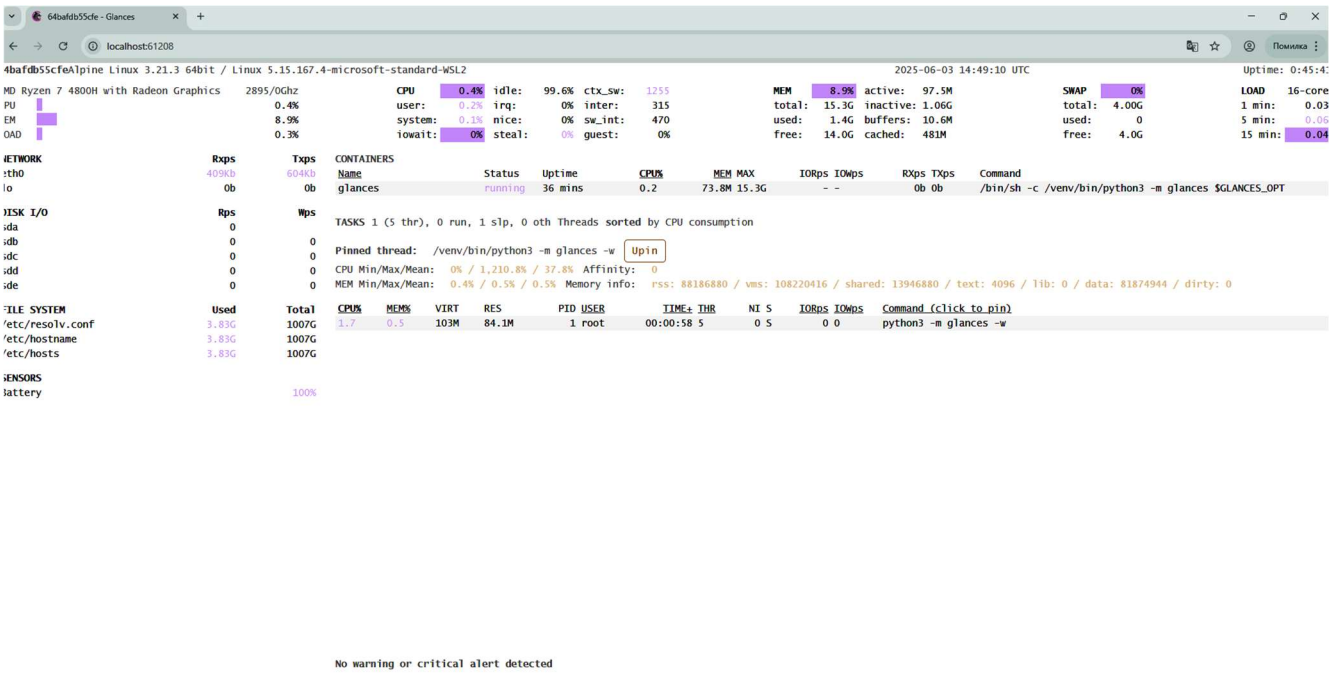


Рисунок 2.11 – Інтерфейс Glances після запуску

У випадку необхідності більш гнучкого управління конфігурацією або підготовки до розгортання на кількох хостах можливо використання docker-compose.yml — конфігураційного файлу, який дозволяє централізовано керувати параметрами контейнера, обсягом доступу, портами та змінними середовища. Такий підхід було реалізовано в рамках демонстраційного проекту, що дало змогу забезпечити швидкий запуск MVP за допомогою однієї команди docker compose up -d, яка активує весь сервіс у фоновому режимі [23].

Особливу увагу слід приділяти аспекту безпеки, оскільки надання контейнеру доступу до PID-простору хоста та монтування Docker-сокету, хоча й необхідне для повноцінного моніторингу всіх процесів та контейнерів, водночас створює потенційно небезпечну ситуацію з точки зору прав доступу. У продакшн-середовищах необхідне ретельне обмеження привілеїв, можливо, за допомогою механізмів AppArmor, SELinux або обмеження доступу через proxy-контейнери [23].

Відкриття Web-інтерфейсу. Після успішного запуску Glances у контейнері веб-інтерфейс автоматично стає доступним у браузері за адресою, що складається з IP-адреси хоста та порту 61208, а у випадку локального запуску його відкриття

можливе навіть без зазначення IP-адреси, достатньо звернутись до http://localhost:61208, або ж скористатися IP-адресою віртуальної машини у межах локальної мережі з тим самим портом, що дає змогу без додаткових дій одразу перейти до панелі моніторингу, яка оновлюється в режимі реального часу (див. рис. 2.12).

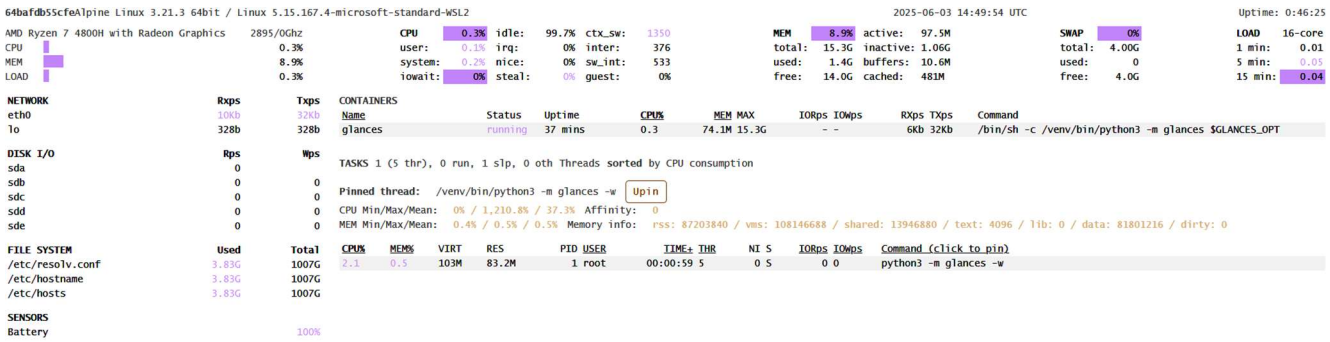


Рисунок 2.12 – Веб-інтерфейс з основними метриками

Відображення даних реалізовано у вигляді кольорових блоків з актуальними показниками, які оновлюються автоматично, демонструючи загальний стан комп’ютерної системи.

Демонстрація основної функціональності. Після відкриття інтерфейсу Glances стає доступним огляд основних функцій системи, що у форматі MVP дозволяє оцінити її базову ефективність без потреби повноцінного розгортання складних інфраструктур. Інтерфейс, хоч і компактний, демонструє достатньо інформативну панель, яка охоплює ключові ресурси системи, зокрема моніторинг завантаження процесора із загальним навантаженням у відсотках, відображенням активності кожного ядра окремо (за наявності підтримки багатоядерності), а також середніми значеннями навантаження за 1, 5 та 15 хвилин (load average), що дозволяє оцінити стабільність роботи системи протягом часу. Оперативна пам’ять (RAM) представлена в розрізі обсягу, що використовується (used), загального обсягу (total), вільного обсягу (free), а також розподілених буферів та кешу, при цьому візуальний індикатор змінює колір відповідно до навантаження: зелений свідчить про нормальний стан, а червоний сигналізує про перенавантаження. Дискова

підсистема охоплює моніторинг файлових систем, наприклад / або /data, з відображенням залишку вільного місця, а також активності у вигляді кількості байтів, що зчитуються або записуються у поточний момент, що дозволяє оперативно оцінювати продуктивність зберігання даних (див. рис. 2.13) [22].

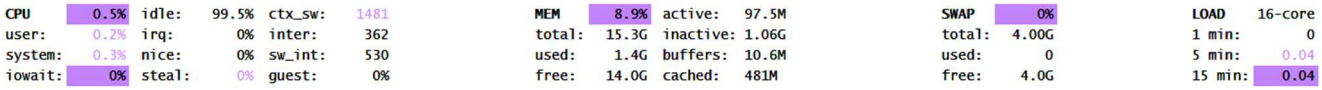


Рисунок 2.13 – Демонстрація роботи CPU, RAM, DISK моніторингу

Додатково доступна інформація про мережеві інтерфейси: швидкість прийому (RX) і передачі (TX) даних, IP-адреси, а також ідентифікація активних з'єднань, які, за потреби, можна активувати для повнішого розуміння стану мережі. При наявності контейнерів Docker, завдяки змонтованому сокету /var/run/docker.sock, Glances отримує можливість бачити всі активні контейнери на хості, що дозволяє переглядати не лише їх список, а й індивідуальне споживання ресурсів для кожного — таких як CPU, RAM та інші показники, що вже описувалися, забезпечуючи більш деталізовану телеметрію в рамках одного зручного інтерфейсу. Такий підхід дозволяє локально, без зовнішніх сервісів чи доступу до інтернету, здійснювати гнучкий контроль над виконанням процесів і станом системи, що особливо доцільно у випадках використання edge-рішень, де критичним є автономний моніторинг [22].

Розширення MVP – підтримка температурних датчиків. Рішення у межах впровадженого MVP, тобто мінімально життєздатного продукту, може бути розширене до більш повноцінного варіанту шляхом підключення температурних сенсорів, що відкриває додаткові можливості для моніторингу системи за умови наявності відповідних бібліотек та інтерфейсів, зокрема LM-sensors у Linux для отримання температур CPU, hddtemp для моніторингу жорстких дисків або використання бібліотек, сумісних із Python версії 3.4 і вище, таких як psutil або sensor-temperatures, які дозволяють отримувати дані про температуру апаратного забезпечення безпосередньо з користувацького простору. За умови реалізації такої

підтримки у правій частині веб-інтерфейсу з’являється секція "sensors", у якій можуть бути представлені значення температури центрального процесора, наприклад, як coretemp зі значенням 52 градуси, температури графічного процесора — якщо графічний адаптер підтримує таку телеметрію, а також температура жорсткого диска, що, наприклад, для пристрою /dev/sda може становити 38 градусів Цельсія, що дозволяє наочно оцінити теплове навантаження на фізичні компоненти системи (див. рис. 2.14).

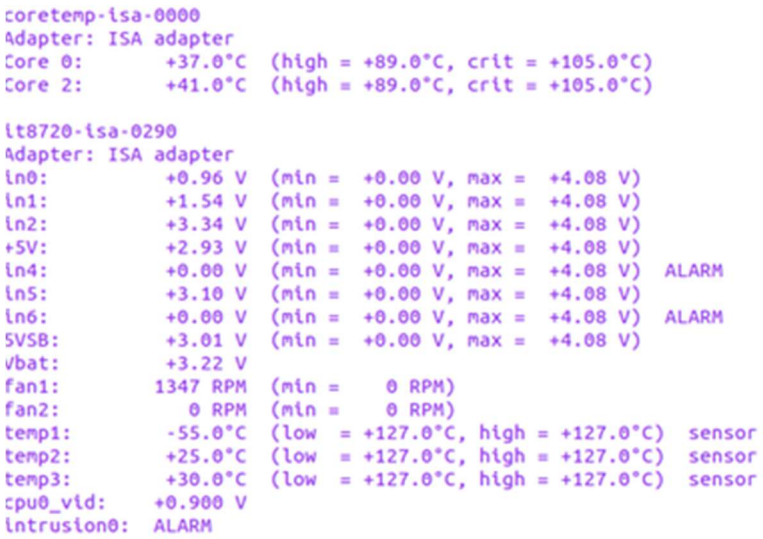


Рисунок 2.14 – Приклад збору фізичних властивостей системи: датчик температури

У контейнерному середовищі відображення температурних показників можливе лише за умови, що контейнер має доступ до апаратних сенсорів хоста, тобто через запуск у привілейованому режимі або за допомогою додаткового монтування директорій типу /sys/class/thermal, проте у демонстраційному прикладі це необов’язково, хоч така підтримка цілком може бути реалізована у подальшому. Розширення MVP у цьому напрямі дозволяє сформувати більш гнучкий прототип, орієнтований на моніторинг критичних температурних зон у різноманітних пристроях, включно з автоматичним сповіщенням про перегрів або побудовою температурних графіків за допомогою зв’язки Grafana та InfluxDB, якщо є потреба в історичному аналізі.

Таблиця 2.3 – Етапи запуску мінімального MVP з Glances

Крок	**Дія**	**Результат**
1	Запуск Docker-контейнера командою docker run або docker-compose up -d	Система моніторингу активована, контейнер Glances працює
2	Перевірка Web UI через браузер (http://host_ip:61208)	Відображення метрик у режимі реального часу через веб-інтерфейс
3	Імітація навантаження на CPU/пам’ять	Дані змінюються в реальному часі, підтверджуючи коректну роботу моніторингу
4	Перехід до вкладки Docker у веб-інтерфейсі	Відображення ресурсів і стану активних Docker-контейнерів
5	Аналіз CSV-логів (за потреби)	Отримання історичних даних про змінні навантаження системи
6	Зміна порту через GLANCES_OPT=-w -p 8080 (за потреби)	Інтерфейс Glances доступний на іншому порту (напр., http://host_ip:8080)

Таким чином, побудова мінімального функціонального рішення на базі Glances у середовищі Docker (див. табл. 2.3) дозволяє оперативно розгорнути систему моніторингу без складної інсталяції, забезпечивши базову візуалізацію ключових системних метрик із можливістю гнучкої інтеграції з контейнеризованими середовищами та поступовим додаванням підтримки апаратного моніторингу температур. Усе це створює відповідність вимогам до прототипу системи аналізу навантаження у форматі тестового впровадження, де пріоритетом залишається швидкість розгортання, модульність і гнучкість у подальшій адаптації до більш складних сценаріїв використання.

РОЗДІЛ 3 ОПТИМІЗАЦІЯ, АНАЛІЗ ТА ПЕРСПЕКТИВИ РОЗВИТКУ

3.1 Методи покращення ефективності системи

Побудова системи моніторингу передбачає не лише реалізацію базового функціоналу збору та відображення даних, а й необхідність забезпечення ефективної роботи у середовищах з обмеженими ресурсами, де може відбуватись її впровадження, включно з віртуальними машинами, вбудованими пристроями або робочими станціями, а також у випадках масштабного розгортання на кількох вузлах. В таких умовах постає потреба в застосуванні методів, що дозволяють суттєво зменшити споживання ресурсів, покращити швидкість реагування на події та підтримувати стабільну роботу без надмірного навантаження на систему. Найбільш доцільним напрямом удосконалення виступає оптимізація частоти оновлення даних, що безпосередньо впливає на продуктивність системи та її здатність працювати в реальному часі або у фоновому режимі за умови мінімального впливу на ресурси інфраструктури [24-26].

Оптимізація частоти оновлення даних. Занадто часте зчитування телеметрії, тобто системних метрик, призводить до надмірного використання процесора, підвищеного споживання оперативної пам’яті, а також створює навантаження на мережеві канали при віддаленому моніторингу, що особливо критично для слабких пристроїв. З іншого боку, надто рідкісне оновлення обмежує оперативність реагування та знижує загальну інформативність інтерфейсу, що вимагає пошуку оптимального балансу між точністю, швидкістю та ресурсним споживанням. У поточному рішенні на основі Glances ця задача вирішується через параметр -t, що визначає інтервал оновлення у секундах, наприклад, glances -w -t 10 встановлює оновлення метрик кожні 10 секунд. Такий інтервал є прийнятним для стандартного моніторингу серверів, тоді як у випадку розробки або тестування доцільнішими є

значення від 1 до 2 секунд, а при роботі на малопотужних пристроях — понад 10 секунд, що дозволяє уникнути перевантаження і зберегти стабільність [24].

У межах цього підходу можливо також розглянути реалізацію адаптивного регулювання частоти оновлення залежно від поточного навантаження системи, що дозволить підвищити ефективність у динамічних умовах без необхідності ручного налаштування. Наприклад, під час пікових навантажень інтервал може збільшуватись, а у випадках спокійного стану — зменшуватись для підвищення чутливості (див. рис. 3.1).

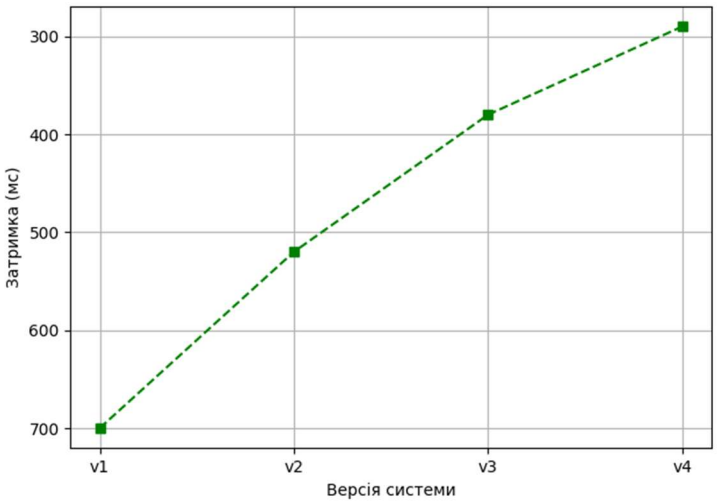


Рисунок 3.1 – Графік покращення частоти оновлення телеметрії (метрик) після оптимізації

Додатково до оптимізації з боку збору метрик важливим напрямом є покращення клієнтського інтерфейсу через механізми кешування або диференційованого оновлення лише окремих частин сторінки, що зменшує кількість запитів і навантаження на серверну частину, дозволяючи зберігати динамічність при значно меншому споживанні ресурсів. Такий підхід забезпечує не лише покращення ефективності, а й створює фундамент для майбутніх розширень, не змінюючи принципової архітектури системи [24].

Зменшення навантаження на сервер. Зменшення навантаження на сервер у системах моніторингу може бути досягнуте за рахунок впровадження певних архітектурних підходів, що дозволяють розподіляти обробку даних, знижуючи

інтенсивність взаємодії з центральним вузлом. Одним із найефективніших рішень у цьому напрямі є використання агентів, які виконують первинну обробку метрик безпосередньо на вузлах збору. У такому підході ще до передачі інформації до центрального сервера відбувається фільтрація зайвих даних, їхня агрегація, наприклад, усереднення або обчислення максимуму, а також передача лише змінених або критично важливих показників, що значно зменшує обсяг трафіку і навантаження на мережу та серверну частину. Цей механізм дозволяє уникати надсилання сирих телеметричних потоків, які за своєю природою є надлишковими у контексті обмежених ресурсів. Додатково можлива реалізація кешування результатів для зниження частоти повторних звернень до системних API або файлів у директорії /proc, що часто стає джерелом надмірної активності з боку інструментів моніторингу. У разі використання Glances цей підхід передбачає збереження раніше зчитаних метрик у пам’яті, дозволяючи уникати зайвого зчитування при кожному оновленні (див. рис. 3.2) [25].

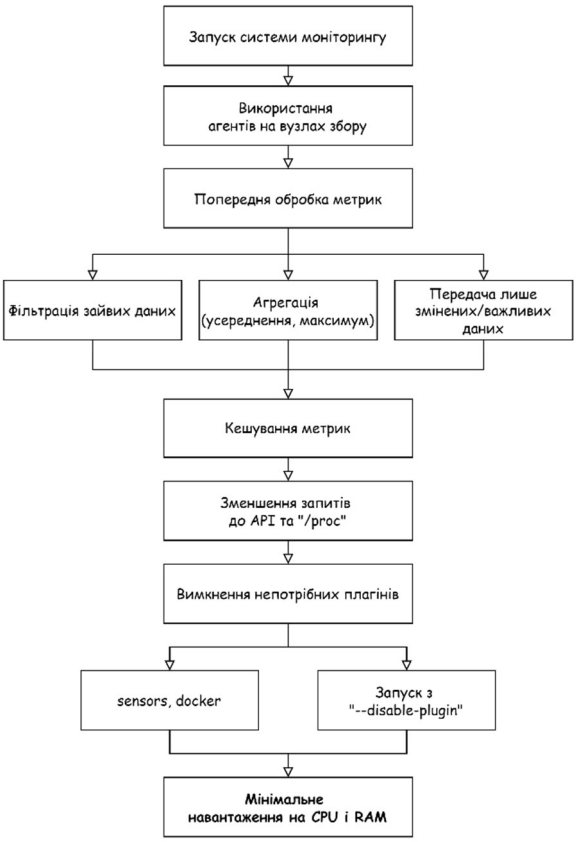


Рисунок 3.2 – Метод зменшення навантаження на сервер

Крім того, додатковим засобом зниження навантаження є відключення непотрібних плагінів або модулів, які не використовуються у поточному сценарії експлуатації. Наприклад, якщо у системі відсутні сенсори температури або контейнерна інфраструктура, можна відключити відповідні модулі sensors і docker, що дозволяє зменшити споживання ресурсів CPU та оперативної пам'яті. У Glances це реалізується через параметр --disable-plugin, що дозволяє запускати систему моніторингу у полегшеному режимі, зосередженому лише на тих показниках, які є дійсно релевантними. Таким чином формується більш керована та стабільна архітектура, в якій використовується мінімум обчислювальних потужностей без втрати інформативності [25].

Механізми сповіщень при перевищенні порогів. Поряд із цим доцільно реалізувати й механізми автоматичного сповіщення про перевищення встановлених порогів, що забезпечує раннє виявлення аномалій ще до того, як виникне критична ситуація. Своєчасне інформування адміністратора або оператора про навантаження на процесор, нестачу оперативної пам'яті, проблеми з дисковою підсистемою або мережевими з'єднаннями дає змогу швидше вжити заходів і уникнути простоїв або втрати даних. У Glances подібний функціонал реалізується через систему autoalert із підтримкою конфігурації порогових значень у файлі glances.conf, де кожна метрика може мати власні межі для сповіщення. У веб-інтерфейсі такі ситуації відображаються через кольорову індикацію: жовтий колір сигналізує про попередження, а червоний — про критичний стан. Крім візуального відображення, система також може бути налаштована на надсилання повідомлень за допомогою електронної пошти, Telegram, Slack або будь-яких інших каналів через зовнішні скрипти або інтеграцію зі сторонніми інструментами. Для сценаріїв, що передбачають використання Prometheus, Glances може виступати у ролі експортувальника даних, доступних за IP, після чого на базі Grafana створюються дашборди з відповідними порогами та тригерами для відправлення повідомлень у вигляді pop-up вікон, SMS або email, формуючи комплексну та гнучку систему реагування без порушення сумісності з Glances як основною платформою моніторингу (див. рис. 3.3).

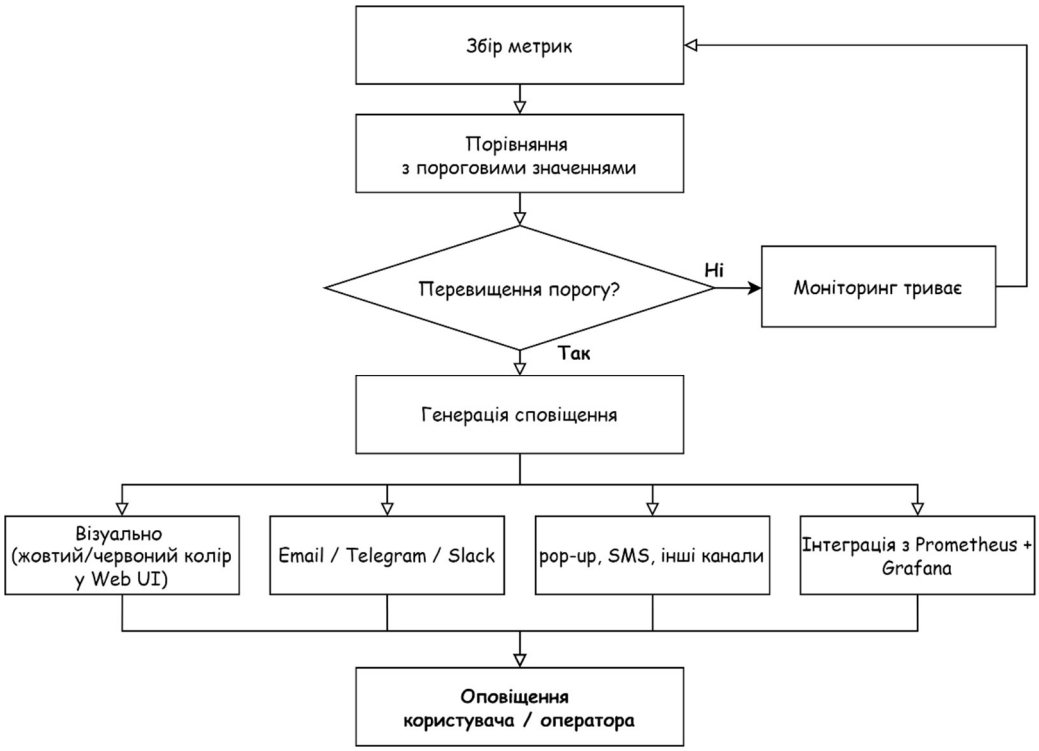


Рисунок 3.3 – Метод генерації сповіщень при перевищенні порогів

Такий підхід поєднує зниження навантаження, підвищення чутливості до подій і кращу масштабованість, особливо актуальну для систем із динамічно змінним середовищем і ресурсними обмеженнями [26].

Логування стану системи для подальшого аналізу. Логування стану системи для подальшого аналізу забезпечує можливість ретроспективного відстеження поведінки системи в різних умовах експлуатації, дозволяючи здійснювати аудит стабільності роботи, формувати уявлення про навантаження в часовому розрізі, а також встановлювати послідовність подій у випадку виникнення збоїв або аномалій, що особливо важливо при дослідженні причин виникнення інцидентів та формуванні стратегій їх недопущення в майбутньому. Реалізація логування в середовищі Glances дозволяє зберігати інформацію у форматах CSV або JSON за допомогою відповідних команд запуску, що відкриває шлях до інтеграції з базами даних часових рядів, такими як InfluxDB, у яких метрики зберігаються у вигляді структурованих даних із часовою прив’язкою. Така інтеграція, з одного боку, дозволяє автоматизувати процес збору статистики, а з іншого — забезпечує зручний канал для подальшого аналізу отриманої інформації за допомогою

аналітичних платформ на кшталт Grafana, яка здатна будувати динамічні дашборди, виводити тенденції зміни навантаження та визначати пікові стани. У тих випадках, коли впровадження повноцінної візуалізації не є необхідним, достатньо лише базового логування у файл, що дозволяє накопичувати історичні дані з мінімальними витратами ресурсів і зберігати їх для подальшого аналізу вручну або напівавтоматизовано (див. рис. 3.4) [26].



Рисунок 3.4 – Приклад структури журналу логування стану зі InfluxDB

Перспективним напрямом є зберігання логів не лише локально, а й на зовнішніх файлових системах, таких як S3-сумісні об’єктні сховища, що забезпечує відмовостійкість, масштабованість і централізований доступ до архівів телеметрії. В обробці таких логів можуть використовуватись скрипти на Python, які дозволяють проводити сортування, фільтрацію, обрахунок статистичних параметрів або ж побудову графіків у середовищах на кшталт Jupyter Notebook, що особливо корисно для дослідницького аналізу системних даних. Подібний підхід дозволяє не лише отримати оперативну оцінку поточного стану інфраструктури, а й виявити закономірності, які формуються з часом, включаючи тренди навантаження, періодичність пікових значень, часові лаги у відповіді системи на

зміну параметрів та інші аспекти, що критично важливі у середовищах з високими вимогами до стабільності та продуктивності [26].

Узагальнюючи вищезазначене, підвищення ефективності вже побудованої системи базового моніторингу досягається завдяки сукупності технічних і організаційних заходів, що включають оптимізацію частоти оновлення, зменшення ресурсного навантаження, впровадження механізмів автоматичного сповіщення та систематичне логування. Така комбінація дозволяє формувати не просто гнучке, а й надійне рішення, яке виходить за межі базового засобу спостереження та набуває ознак завершеної моніторингової платформи з передбачуваним життєвим циклом та потенціалом до масштабування. Завдяки відкритій архітектурі Glances можливе його розширення та інтеграція з більш масштабними інфраструктурними проєктами, які потребують високої доступності, поглибленого аналізу динаміки навантаження та автоматизованого реагування на події у режимі реального часу. У результаті формується основа для глибшої операційної аналітики та стабільного функціонування всієї інформаційної екосистеми незалежно від складності та динаміки її змін [24-26].

3.2 Порівняльний аналіз із іншими системами

Існує велика кількість інструментів, призначених для моніторингу IT-інфраструктури, кожен з яких має власні переваги, обмеження, особливості архітектури та сфери застосування, в яких його використання буде більш доцільним і ефективним. У контексті порівняння доцільно розглянути побудоване рішення на основі Glances у співвідношенні з двома найбільш поширеними альтернативами — системою на базі Prometheus із Grafana та інструментом Zabbix — із фокусом на таких аспектах, як складність встановлення, гнучкість конфігурації, масштабованість, підтримка різних джерел даних і здатність адаптуватися до різних типів мереж — починаючи з домашніх середовищ і

закінчуючи офісною чи промисловою інфраструктурою. Подібне порівняння дає змогу не лише оцінити поточний рівень зрілості рішення на базі Glances, а й сформувати уявлення про його перспективи у загальному ландшафті інструментів моніторингу [27-28].

Рівень складності встановлення. Якщо розглядати рівень складності встановлення, Glances демонструє значну простоту впровадження, оскільки може запускатися за допомогою одноразової команди в Docker або встановлюватися напряму через менеджер пакетів, наприклад pip, що дозволяє отримати повністю функціональний моніторинг без потреби у зовнішній базі даних, додаткових службах чи конфігураційних процедурах. У випадку з Prometheus і Grafana ситуація виглядає складнішою, адже впровадження включає інсталяцію двох окремих компонентів — власне Prometheus-сервера та Grafana-інтерфейсу, налаштування конфігураційних файлів для джерел даних, встановлення Node Exporter як агента на кожен з вузлів інфраструктури, інтеграцію компонентів та регулярне обслуговування сховища метрик. Попри це, система пропонує автоматизоване виявлення цілей моніторингу та широкі можливості через декларативну конфігурацію, що дозволяє створювати гнучкі та масштабовані рішення, проте вимагає знань мови запитів PromQL, без якої побудова складніших метрик стає ускладненою [27].

У свою чергу, Zabbix є найбільш комплексним із порівнюваних рішень і водночас найвибагливішим у плані інсталяції. Стандартна конфігурація потребує розгортання серверного компонента, бази даних (MySQL або PostgreSQL), вебінтерфейсу, агента на кожному вузлі, а також може включати SNMP або HTTP-моніторинг, що додатково ускладнює структуру системи. Окремо налаштовується робота з зовнішніми API, скриптами, користувацькими перевірками, а також доступ до розширеної функціональності вимагає попереднього створення шаблонів і тригерів. Така глибина можливостей дозволяє адаптувати Zabbix до складних корпоративних умов, але водночас потребує глибокого технічного розуміння його компонентів і принципів функціонування [27].

На цьому фоні Glances виглядає найбільш придатним до швидкого впровадження у середовищах, де не ставляться вимоги до побудови розгалуженої інфраструктури, проте необхідне швидке отримання ключових метрик з мінімальними ресурсними витратами. Prometheus із Grafana пропонують гнучкість, масштабованість і велику кількість інструментів інтеграції, але вимагають глибших технічних навичок і часу на розгортання. Zabbix залишається інструментом корпоративного класу з максимальним охопленням, орієнтованим на сценарії, де потрібен централізований контроль, розширене управління правами доступу, налаштування SLA та потужна подієва логіка (див. рис. 3.5).



Рисунок 3.5 – Діаграма складності розгортання різних систем

У підсумку вибір залежить від наявних ресурсів, досвіду команди, очікуваних навантажень та масштабів інфраструктури, однак Glances забезпечує оптимальну відправну точку для малих та середніх сценаріїв без складної адаптації, з можливістю поступового переходу до більш масштабованих платформ [27].

Гнучкість та масштабованість. У контексті порівняння гнучкості та масштабованості розглянутих систем, побудоване рішення на базі Glances виявляється досить обмеженим через свою орієнтацію переважно на локальний моніторинг і використання лише вбудованих плагінів, таких як CP, RAM, Docker

або базові сенсори, що обумовлює обмеження у функціональності щодо кастомізації та масштабування, що, в свою чергу, робить Glances доцільним переважно для моніторингу невеликої кількості систем, наприклад, у рамках демонстраційного середовища або нечисленної інфраструктури з однієї до трьох машин, де не виникає потреби у складному зборі, обробці та агрегації метрик [27].

На противагу цьому, поєднання Prometheus та Grafana демонструє високу гнучкість за рахунок підтримки створення кастомних дашбордів, складних запитів на основі PromQL, налаштування власних панелей візуалізації та динамічного конфігурування метрик, що дозволяє адаптувати систему практично до будь-якого сценарію застосування. Завдяки масштабованій архітектурі Prometheus може працювати з мільйонами метрик одночасно, підтримуючи реплікацію, горизонтальне шардирування даних та федерацію, що робить його надзвичайно ефективним у розподілених середовищах. Використання експортерів дозволяє автоматизувати виявлення цілей і значно полегшує розширення інфраструктури моніторингу [27].

Комерційне корпоративне рішення, представлене на прикладі Zabbix, має високу гнучкість завдяки підтримці макросів, шаблонів, тригерів, сценаріїв автоматизації та кастомізованих інтерфейсів. Масштабованість Zabbix реалізована через агентно-орієнтовану архітектуру, що забезпечує централізоване управління сотнями, а в окремих випадках і тисячами вузлів з єдиної точки, що є надзвичайно зручним у великих підприємствах або в умовах складних виробничих мереж (див. рис. 3.6) [27].

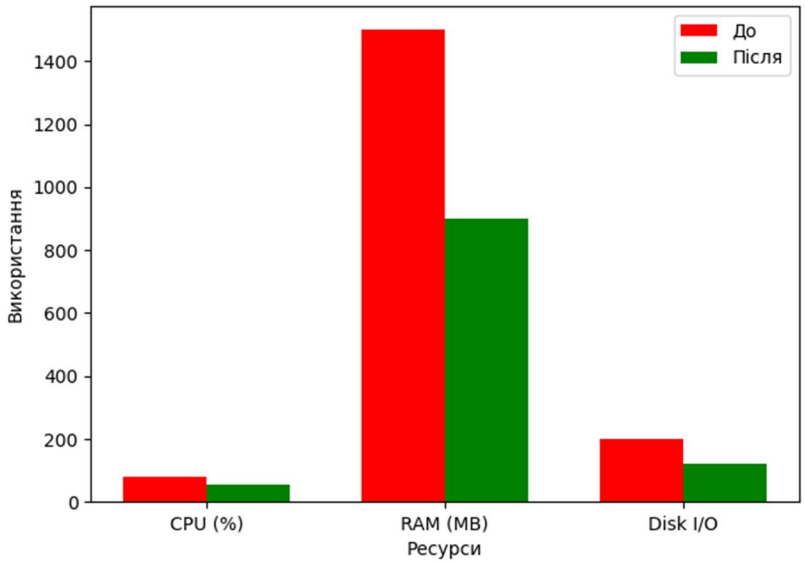


Рисунок 3.6 – Порівняння використання ресурсів до та після оптимізації системи зі Glances

Підтримка різних джерел даних. Щодо підтримки джерел даних, Glances обмежується вбудованою підтримкою стандартних системних метрик через утиліти типу ps, отримання інформації з Docker-контейнерів через сокети, а також базові сенсори температури чи навантаження, що є прийнятним лише в умовах мінімального середовища. Проте спроба інтегрувати сторонні протоколи або нестандартні джерела даних, як-от SNMP чи HTTP API, не є можливою без застосування сторонніх рішень, що обмежує гнучкість інтеграцій (див. рис. 3.7) [27].

Prometheus у зв’язці з Grafana має значно ширшу підтримку джерел даних, яка налічує понад п’ятдесят варіантів, охоплюючи системні агенти типу Node Exporter, бази даних, мережеві пристрої, хмарні сервіси на кшталт AWS або Azure, а також специфічні джерела на основі форматів OpenTelemetry, JSON, InfluxDB тощо. Така архітектура дозволяє побудувати гнучку модель збору інформації, яку легко масштабувати, реплікувати та візуалізувати за допомогою широкого спектра візуальних компонентів Grafana [27].

У випадку Zabbix забезпечується глибока підтримка різноманітних джерел даних завдяки використанню безагентного моніторингу через SNMP, HTTP, ICMP, IPMI, JMX, що дозволяє обійтися без встановлення агентів на кожен вузол.

Додатково передбачена підтримка кастомних скриптів на Bash, PowerShell або інших мовах, що дозволяє адаптувати моніторинг до специфічних завдань. Важливим компонентом є наявність готових шаблонів для моніторингу широкого спектру пристроїв і платформ, включаючи маршрутизатори, сервери, системи віртуалізації та хмарні сервіси, що значно пришвидшує розгортання та знижує поріг входу у великих інфраструктурах [27].

Застосовність у різних умовах. У межах розгляду застосовності різних систем моніторингу в умовах, що суттєво відрізняються за масштабом та функціональним призначенням, Glances виявляється найпридатнішим для використання в домашньому середовищі, де є потреба контролювати лише обмежену кількість пристроїв, таких як один або два хости, сервер або, наприклад, Raspberry Pi. Простота інтерфейсу, мінімальні вимоги до конфігурації та швидкість розгортання роблять його зручним інструментом для реалізації базового моніторингу, що охоплює лише найнеобхідніше, не перевантажуючи користувача складністю архітектури або надмірною деталізацією метрик [28].

Prometheus зі Grafana також можуть бути впроваджені в домашньому середовищі, однак через свою архітектуру, яка орієнтована на масштабованість, автоматизацію, гнучку візуалізацію та використання складних запитів до метрик, таке рішення виглядає надмірно складним для побутового застосування, хоча водночас відкриває більше можливостей у разі побудови домашніх лабораторій або експериментальних середовищ для підготовки до професійної роботи. У випадку Zabbix, як комерційного та корпоративного рішення, його застосування в домашніх умовах виявляється нераціональним через складність налаштування, необхідність у централізованій інфраструктурі, орієнтацію на тривале зберігання даних, великий набір інтеграцій, розгалужену систему сповіщень та високий ступінь автоматизації, що робить його ефективним лише за наявності відповідного масштабу задач і ресурсів для розгортання [28].

У рамках офісних середовищ Glances може використовуватись як тимчасове або допоміжне рішення, яке дозволяє швидко отримати базову інформацію щодо навантаження на машини або короткочасно перевірити працездатність невеликих

кластерів або окремих систем, проте у випадку постійної роботи в команді DevOps, QA чи системного адміністрування куди доцільніше застосовувати Prometheus разом із Grafana завдяки широкому діапазону інструментів для моніторингу, що дозволяють динамічно змінювати конфігурації, зручно масштабувати систему при зростанні кількості вузлів, будувати складні алерти та інтегруватися з CI/CD конвеєрами, а також хмарними сервісами різних постачальників. Така комбінація особливо вигідна за умови, якщо існує потреба у відстеженні зміни параметрів продуктивності в реальному часі, автоматичному масштабуванні або перевірці релізів у процесах розгортання застосунків [28].

Zabbix при цьому краще функціонує у великих корпоративних середовищах, де є потреба у централізованому моніторингу сотень або навіть тисяч пристроїв, серверів, маршрутизаторів, мережевої інфраструктури, включаючи фізичні й віртуальні машини, при чому важливо забезпечити не лише збір метрик, а й систему сповіщень, підтримку SLA, резервне копіювання конфігурацій, інтеграцію з CMDB та інші бізнес-критичні функції. У промисловому сегменті також спостерігається певна диференціація, коли Prometheus і Grafana демонструють себе ефективно в умовах побудови CI/CD, моніторингу клауд-середовищ, застосуванні у DevOps-ланцюжках, де потрібна гнучкість, інтеграція через API, підтримка нових форматів експорту, а також динамічний масштаб, що змінюється під навантаженням [28].

Zabbix же переважно використовується у класичних промислових мережах із протоколами SNMP, IPMI, JMX, SCADA, де більш важливо забезпечити стабільну роботу протягом тривалого часу, гарантовану підтримку старих пристроїв і широке використання готових шаблонів моніторингу (див. рис. 3.7).

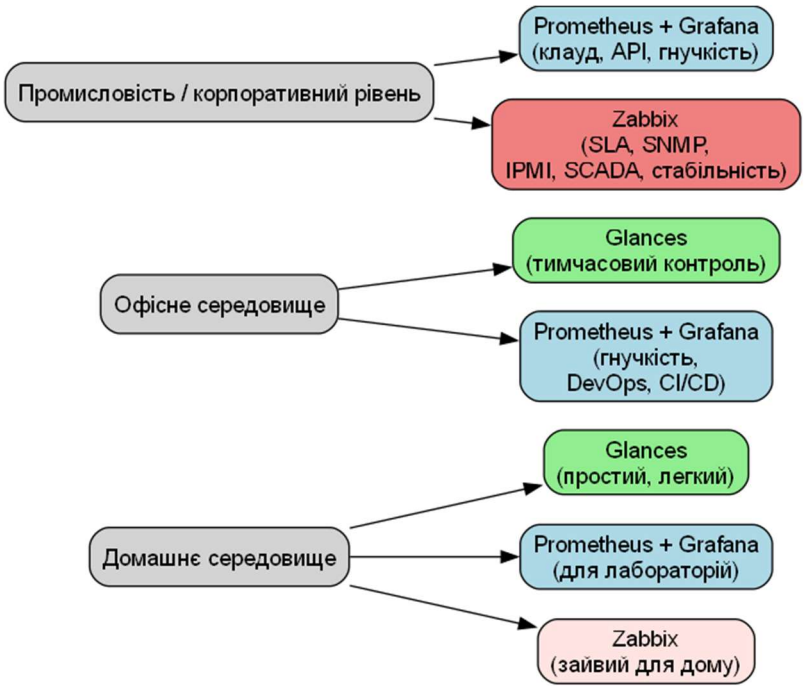


Рисунок 3.7 – Сценарії застосування систем в умовах реального середовища

Таким чином, Glances залишається ефективним у простих або навчальних середовищах, Prometheus із Grafana — у динамічних, хмарно-орієнтованих або лабораторних інфраструктурах, а Zabbix — у середовищах з акцентом на надійність, масштаб і безперервність [28].

Таблиця 3.1 – Порівняльна характеристика систем моніторингу Glances зі Prometheus + Grafana, Zabbix

Критерій	**Glances**	**Prometheus + Grafana**	**Zabbix**
Тип рішення	Локальне / агентське	Серверно-агентське з візуалізацією	Повноцінна серверно-агентська система
Принцип роботи	Реального часу, pull-запити	Pull-модель, збирання метрик через експортери	Переважно pull, з підтримкою trap-методу

Продовження таблиці 3.1 – Порівняльна характеристика систем моніторингу Glances зі Prometheus + Grafana, Zabbix

Критерій	**Glances**	**Prometheus + Grafana**	**Zabbix**
Інтерфейс користувача	CLI, Web (простий), API	Web-дашборди Grafana, API, Prometheus UI	Web, конфігурований dashboard
Легкість встановлення	Дуже проста, Docker/пакети	Середня складність: окремі компоненти, експортери, конфігурація	Складна: потребує сервер, БД, агенти
Підтримка платформ	Windows, Linux, macOS	Linux, macOS, частково Windows (через WSL/WSA)	Windows, Linux, Unix
Моніторинг системних ресурсів	Так (CPU, RAM, диск, мережа)	Так (через Node Exporter та інші)	Так (CPU, RAM, диск, мережа)
Підтримка контейнерів/Docker	Так (через Docker socket)	Так (через cAdvisor, Docker Exporter)	Так (з налаштуванням)
Можливість кастомізації	Обмежена	Висока (PromQL, власні дашборди Grafana)	Висока (тригери, шаблони, сценарії)
Візуалізація історичних даних	Ні (реальний час тільки)	Так (довготривале зберігання метрик)	Так (зберігається в БД, повна історія)

Продовження таблиці 3.1 – Порівняльна характеристика систем моніторингу Glances зі Prometheus + Grafana, Zabbix

Критерій	**Glances**	**Prometheus + Grafana**	**Zabbix**
Нотифікації/оповіщення	Ні (можливе через розширення)	Так (через Alertmanager, інтеграції)	Так (розвинена система тригерів та повідомлень)
Споживання ресурсів	Низьке	Середнє	Високе при великій кількості агентів
Масштабованість	Ні (локальний моніторинг)	Так (федерація, шардирування, автоматичне виявлення цілей)	Так (підтримує кластери, розподілену інфраструктуру)
Підтримка зовнішніх метрик (наприклад, SNMP)	Ні	Так (через SNMP Exporter, API, HTTP endpoints)	Так (повна підтримка SNMP, IPMI, API)
Інтеграції зі сторонніми сервісами	Ні	Так (Telegram, Slack, Email, Webhooks, API, Thanos, Loki)	Так (будь-які через API, webhooks)
Ліцензія	Open Source (GNU GPL v3)	Open Source (Apache 2.0)	Open Source (GPL v2), є комерційна версія

Підсумовуючи, обрання конкретного рішення як Glances (див. табл. 3.1), повинно базуватись не лише на технічних характеристиках і функціональних можливостях, але й на врахуванні середовища розгортання, ресурсної доступності, специфіки завдань та потреб у подальшій експлуатації, адже саме відповідність

рішення цим факторам визначає ефективність його впровадження у контексті будь-якої інформаційної інфраструктури.

3.3 Перспективи розвитку комплексу

Поточна реалізація програмно-апаратного комплексу моніторингу на базі Glances демонструє ефективність для контролю стану окремих машин або невеликої кількості вузлів, зокрема коли йдеться про обмежені за ресурсами або застарілі пристрої в локальних IT-інфраструктурах, однак зі зростанням масштабів систем і складності проектів зростає потреба в розширенні функціоналу, включаючи інтеграцію додаткових сервісів, централізоване збирання метрик, довготривале збереження історичних даних та впровадження механізмів швидкого сповіщення про критичні події з метою зниження часу реагування на інциденти і підвищення надійності всієї інфраструктури загалом [29-30].

Інтеграція зі службами сповіщення – Telegram-бот, email. У межах подальшого розвитку комплексу одним з очевидних кроків є впровадження механізмів сповіщення, які дозволяють оперативно отримувати критичні повідомлення про стан системи, наприклад у разі перевищення допустимої температури процесора, нестачі оперативної пам’яті, відмови накопичувача або критичного зростання навантаження на систему, найпростіше рішення у цьому напрямі — це інтеграція з Telegram-ботом, який забезпечує високу швидкість доставки алертів, незалежність від платформи користувача, можливість зворотного зв’язку шляхом введення команд (наприклад, top, uptime або status), а також легку реалізацію завдяки бібліотекам Python Telegram Bot або Aiogram (див. рис. 3.8) [29].



Рисунок 3.8 – Інтеграція Telegram-бота у загальну систему

Така інтеграція може бути реалізована через створення окремого контейнера із ботом, що забезпечить стабільну роботу та гнучкість у його конфігурації, а також підключення модуля apGlances або psutil, який дозволить отримувати поточну інформацію про стан системи безпосередньо в момент запиту; у випадку необхідності додатково можна реалізувати сценарії відправки автоматичних повідомлень на основі порогових значень ресурсів, при перевищенні яких бот самостійно сповіщатиме адміністратора.

Окрім Telegram, менш інтерактивним, але також ефективним засобом сповіщення може бути email, особливо у випадках, коли потрібна періодична звітність або архівування сповіщень; реалізація цього методу можлива через використання SMTP-сервісів, таких як Postfix або Sendmail, у поєднанні з популярними поштами на зразок Gmail, що дозволяє надсилати як агреговані добові метрики, так і термінові повідомлення про порушення встановлених порогів у використанні ресурсів.

Загалом підключення механізмів сповіщення до комплексу на базі Glances дозволяє підвищити інформованість відповідальних осіб про поточний стан системи та зменшити ймовірність тривалого простою або критичних збоїв, оскільки така інтеграція створює передумови для переходу до проактивної моделі адміністрування, коли реагування відбувається не постфактум, а в режимі реального часу, що особливо важливо в умовах розширення інфраструктури або підвищених вимог до надійності IT-систем.

Збір додаткових показників – GPU, температури чипсетів. Сучасні обчислювальні платформи дедалі частіше використовують дискретні графічні процесори для виконання ресурсоємних задач, таких як обробка відео, машинне навчання чи паралельні обчислення, однак у більшості базових систем моніторингу температура, споживання ресурсів та загальне навантаження на GPU залишаються поза полем зору, що створює потенційні ризики перегріву, деградації продуктивності або передчасного зношення обладнання. Для вирішення цієї проблеми можлива реалізація збору додаткових метрик, зокрема за допомогою інструментів nvidia-smi або py3nvml для GPU виробництва Nvidia, або AMD GPU Pro для пристроїв на базі AMD. В результаті стає доступним отримання критичних параметрів, таких як температура, рівень завантаження, енергоспоживання, стан пам’яті, а також телеметрія VR, що дозволяє відстежити роботу графічного адаптера з високою точністю.

Крім того, для контролю температури чипсетів материнської плати, таких як північний та південний мости, сенсори VRM або модулі живлення, можливо використовувати інструменти на кшталт lm-sensors, psensor або pySensors, які взаємодіють із системою через протоколи SMBus та I2C, за умови підтримки відповідних драйверів, таких як coretemp, k10temp або it87. Дані, отримані через такі модулі, можуть бути інтегровані до Glances за допомогою додаткових плагінів, окремого модуля на Python або шляхом запису кастомних метрик у форматі JSON, що дозволить візуалізувати критичні параметри поряд із класичними системними показниками. Таким чином, розширення збору метрик на рівень GPU і температури внутрішніх компонентів дозволяє краще розуміти теплові процеси в системі, що є

критично важливим для стабільної роботи середніх обчислювальних станцій, персональних комп'ютерів, орієнтованих на ігри чи обробку графіки, а також серверних платформ, де перегрів може стати причиною суттєвих втрат (див. рис. 3.9).

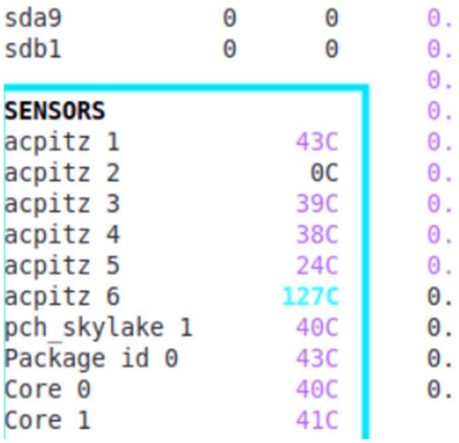


Рисунок 3.9 – Приклад додаткової телеметрії (метрик): GPU, чипсет

Поточна архітектура Glances за своєю суттю орієнтована на роботу в режимі локального агента, де кожен екземпляр програми встановлюється безпосередньо на машині, яка підлягає моніторингу, і надає інформацію лише про поточний вузол через веб-інтерфейс або API, що створює обмеження у випадку потреби централізованого спостереження за декількома серверами чи пристроями у складі більшої інфраструктури. Для вирішення цього питання можливо реалізувати кластерну топологію, у якій кожен вузол передає свої метрики на центральний сервер за допомогою REST API, MQTT або подібного каналу передачі даних, а центральний збирач агрегує цю інформацію для подальшої обробки або візуалізації.

Підтримка багатьох вузлів (кластерна топологія). Іншим ефективним підходом є поєднання Glances з системою збору метрик Prometheus, який може збирати дані за допомогою спеціального експортера, після чого ці дані виводяться на дашборди у Grafana, що дозволяє реалізувати масштабування моніторингу до десятків або навіть сотень вузлів. При цьому стає можливою побудова інтерактивних карт мережевої структури, групування вузлів за ролями (сервери,

клієнти, маршрутизатори, системи зберігання тощо), а також впровадження фільтрації та порогових алертів для різних груп пристроїв (див. рис. 3.10).

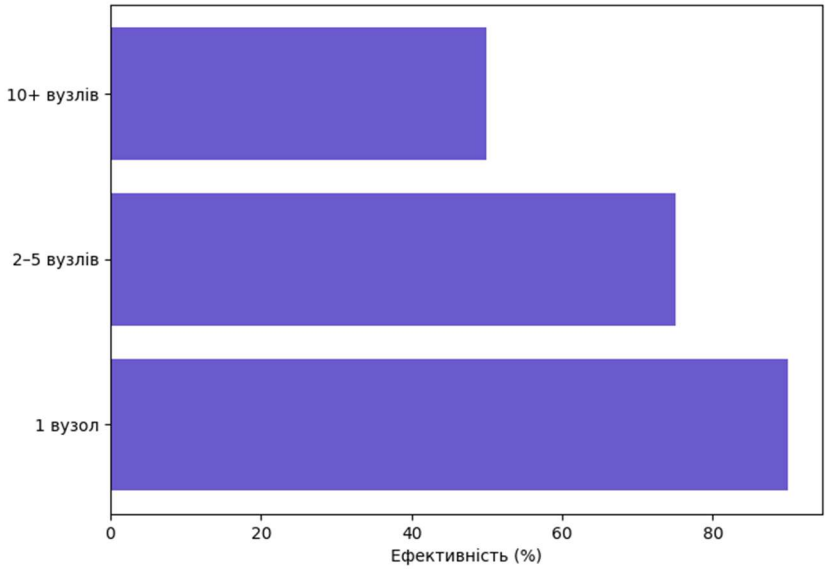


Рисунок 3.10 – Процентне співвідношення різності у використанні одного агента на всіх вузлах

Така структура забезпечує зручне централізоване керування великою кількістю вузлів, знижує складність масштабування системи моніторингу та дозволяє швидко ідентифікувати проблемні ділянки в інфраструктурі, що є важливим кроком у бік створення надійного та масштабованого середовища спостереження для будь-яких типів інформаційних систем.

Історичне зберігання даних у базі. Ще одним важливим аспектом функціонального розвитку моніторингового комплексу є проблема відсутності історичного зберігання даних, оскільки у поточній реалізації Glances відображається виключно поточний стан системи, що обмежує можливості проведення ретроспективного аналізу, побудови графіків змін, виявлення трендів або ж прогнозування майбутнього навантаження. Для подолання цього обмеження доцільним є впровадження механізмів збереження інформації у базі даних, що наразі не реалізовано. Як варіанти технологічного забезпечення можуть виступати такі рішення як InfluxDB, Cassandra (CQL Lite) або PostgreSQL, які дозволяють фіксувати метрики у вигляді часових рядів із заданими інтервалами, наприклад,

кожні 10–60 секунд. Формат збереження передбачає використання timestamp-позначок, де для кожного запису вказується тип метрики, її значення, час реєстрації та вузол, з якого отримано інформацію. Архітектурно це може виглядати як передача зібраних glances-агентом даних через REST або інші API-запити, які обробляються спеціальним Python-скриптом-збирачем, після чого дані записуються до відповідної бази, а інтерфейс перегляду на зразок Grafana забезпечує візуалізацію отриманої інформації [30].

Таким чином, реалізація збереження історичних метрик відкриває значно ширші можливості — побудову часових графіків змін параметрів навантаження, виявлення пікових аномалій у використанні ресурсів (таких як CPU, RAM, диск або мережеві інтерфейси), а також ретроспективне вивчення поведінки системи в конкретні проміжки часу, що дозволяє здійснювати причинно-наслідковий аналіз. Наприклад, у разі регулярного перевищення допустимого навантаження на один із ресурсів з’являється можливість точно визначити, коли саме це відбувалося і який саме компонент потребує оптимізації або заміни, що унеможливлюється при відсутності збереження історії. Інакше система залишається обмеженою до відображення лише поточного стану, що суттєво знижує її ефективність в умовах серйозних навантажень чи потреби довготривалого аудиту [30].

Узагальнюючи наведене, перспективи розвитку моніторингу комплексу на базі Glances демонструють значний потенціал для еволюції від базового MVP-рішення до повноцінної гнучкої та масштабованої системи. Реалізація додаткових функцій, зокрема підтримки температурних метрик GPU і чипсетів, забезпечення підтримки багатьох вузлів у кластерній топології та введення історичного зберігання метрик у базі даних, дозволяє адаптувати поточну архітектуру до складніших умов експлуатації. Такий розвиток дає змогу переходити від обмеженого локального моніторингу до повноцінного інструмента супроводу в умовах офісного середовища, хмарної інфраструктури або промислових платформ з високим навантаженням і складною структурою, задовольняючи зростаючі вимоги до стабільності, прозорості та передбачуваності системного функціонування [29-30].

ВИСНОВКИ

У процесі реалізації апаратно-програмного комплексу було обрано інструмент Glances, який дозволяє швидко та зручно отримувати інформацію про стан комп’ютера через веб-інтерфейс. Запуск сервісу за допомогою контейнеризації спростив розгортання системи та зробив її легкою для відтворення на будь-якому вузлі з підтримкою Docker. Отримані метрики охоплюють завантаження CPU, використання оперативної пам’яті, активність дискової підсистеми, параметри мережі та інші важливі показники.

Веб-інтерфейс Glances надав можливість візуалізувати дані в режимі реального часу, що спростило спостереження за змінами навантаження системи та оперативну діагностику. Продемонстровано функціонування комплексу в мінімальному варіанті (MVP), що може бути використаний у реальному середовищі без складного налаштування. Окремо розглянуто можливість підключення додаткових сенсорів, зокрема температурних датчиків, що розширює функціональність системи та дозволяє стежити за критичними параметрами охолодження.

Система виявилася ефективною в умовах обмежених ресурсів та мінімальної інфраструктури. Простота налаштування, швидкість розгортання та доступність через веб-браузер роблять її зручною для використання в освітніх або офісних умовах. У ході тестування система показала стабільну роботу та точне відображення основних метрик без значного навантаження на обчислювальні ресурси.

Додатково були запропоновані напрями покращення, зокрема оптимізація частоти оновлення даних, реалізація сповіщень при перевищенні критичних порогів, логування показників для подальшого аналізу, а також розширення системи до підтримки історичних даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Computer Monitoring Basics. Paessler AG. URL: https://www.paessler.com/it-explained/monitoring (дата звернення: 02.03.2025)
2. Monitoring and Observability. Red Hat. URL: https://www.redhat.com/en/topics/devops/what-is-observability (дата звернення: 28.02.2025)
3. Types of Monitoring Tools. ScienceSoft. URL: https://www.scnsoft.com/blog/types-of-monitoring-tools (дата звернення: 05.03.2025)
4. Telemetry Basics. Microsoft Learn. URL: https://learn.microsoft.com/en-us/azure/azure-monitor/overview (дата звернення: 08.03.2025)
5. SNMP vs. NetFlow vs. WMI. SolarWinds. URL: https://www.solarwinds.com/resources/it-glossary/snmp-vs-netflow-vs-wmi (дата звернення: 13.03.2025)
6. Prometheus Monitoring and Telemetry. Prometheus.io. URL: https://prometheus.io/docs/introduction/overview/ (дата звернення: 11.03.2025)
7. Data Visualization Best Practices. Chartio. URL: https://chartio.com/learn/charts/chart-design-best-practices/ (дата звернення: 17.03.2025)
8. Web UI Architecture. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn/Front-end_web_developer (дата звернення: 14.03.2025)
9. SPA vs MPA: What’s the Difference? Kinsta. URL: https://kinsta.com/blog/spa-vs-mpa/ (дата звернення: 19.03.2025)
10. Best Open Source Monitoring Tools. Comparitech. URL: https://www.comparitech.com/net-admin/best-open-source-network-monitoring-tools/ (дата звернення: 21.03.2025)
11. Nagios vs Zabbix vs Prometheus. Geekflare. URL: https://geekflare.com/network-monitoring-tools/ (дата звернення: 25.03.2025)
12. Top 10 Server Monitoring Tools. Hostinger. URL: https://www.hostinger.com/tutorials/server-monitoring-tools (дата звернення: 23.03.2025)
13. System Requirements for Monitoring Software. ManageEngine. URL: https://www.manageengine.com/network-monitoring/system-requirements.html (дата звернення: 29.03.2025)
14. Functional vs Non-Functional Requirements. IBM. URL: https://www.ibm.com/cloud/blog/functional-vs-nonfunctional-requirements (дата звернення: 31.03.2025)
15. System Monitoring Architecture. Datadog Docs. URL: https://docs.datadoghq.com/agent/basic_agent_usage/ (дата звернення: 02.04.2025)

16. Architecture Patterns for Monitoring. Azure Architecture Center. URL: https://learn.microsoft.com/en-us/azure/architecture/example-scenario/monitoring/log-analytics (дата звернення: 04.04.2025)
17. Glances Documentation. Glances Project. URL: https://nicolargo.github.io/glances/ (дата звернення: 06.04.2025)
18. Dockerizing Applications. Docker Docs. URL: https://docs.docker.com/get-started/overview/ (дата звернення: 09.04.2025)
19. Docker Socket Monitoring. ContainIQ. URL: https://www.containiq.com/post/docker-socket (дата звернення: 07.04.2025)
20. Python psutil library. PyPI. URL: https://pypi.org/project/psutil/ (дата звернення: 11.04.2025)
21. Client-Server Data Flow Models. Cloudflare Learning. URL: https://www.cloudflare.com/learning/ddos/glossary/client-server-model/ (дата звернення: 13.04.2025)
22. Minimal Viable Product for Monitoring. Devopedia. URL: https://devopedia.org/minimum-viable-product (дата звернення: 15.04.2025)
23. Running Glances in Docker. Techno Tim YouTube. URL: https://www.youtube.com/watch?v=dxk8bQvVSHU (дата звернення: 18.04.2025)
24. Optimizing Monitoring Frequency. Better Stack. URL: https://betterstack.com/community/guides/logging/monitoring-frequency/ (дата звернення: 20.04.2025)
25. Reducing Server Load in Monitoring. Site24x7. URL: https://www.site24x7.com/monitoring.html (дата звернення: 23.04.2025)
26. Alerting Best Practices. Grafana Labs. URL: https://grafana.com/docs/grafana/latest/alerting/ (дата звернення: 21.04.2025)
27. Comparison of Monitoring Tools. StackShare. URL: https://stackshare.io/stackups/nagios-vs-zabbix-vs-prometheus (дата звернення: 27.04.2025)
28. Scalability in Monitoring Systems. Dynatrace. URL: https://www.dynatrace.com/news/blog/scalable-observability/ (дата звернення: 29.04.2025)
29. Telegram Bot API for Alerting. Telegram Docs. URL: https://core.telegram.org/bots/api (дата звернення: 02.05.2025)
30. Historical Monitoring Data Storage. InfluxData. URL: https://www.influxdata.com/time-series-database/ (дата звернення: 05.05.2025)

Додаток А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра комп'ютерної інженерії



КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

НА ТЕМУ: «АПАРАТНО-ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ МОНІТОРИНГУ СТАНУ КОМП'ЮТЕРА»

Виконав студент групи КІД-42
Тарасенко Павло Русланович
Керівник дипломної роботи:
Асєєва Людмила Анатоліївна,
phD (д-р філософії), доц.

Київ – 2025

Мета, об’єкт, предмет дослідження

Мета роботи – спрощення впровадження базового моніторингу апаратних ресурсів комп’ютера за рахунок використання принципів модульності, контейнеризації та відкритих кросплатформених засобів візуалізації

Об’єкт дослідження – процес моніторингу стану апаратних компонентів комп’ютера в умовах обмежених ресурсів

Предмет дослідження – технології збору, обробки та представлення метрик системних ресурсів з використанням відкритих інструментів

Актуальність роботи

У роботі проаналізовано вимоги до мінімальної інфраструктури моніторингу комп'ютерної системи, охарактеризовано функціональні та нефункціональні вимоги до програмно-апаратного комплексу, а також представлено короткий огляд і порівняння популярних рішень: Glances, Prometheus зі Grafana, Zabbix.

Розроблено тестове середовище з використанням Glances у контейнері Docker, налагоджено Web-доступ до метрик і забезпечено базове візуальне представлення даних про навантаження на CPU, пам'ять, дискову підсистему, мережу та контейнери Docker. Наведено алгоритм обробки даних та логіку побудови MVP з можливістю подальшого розширення (сповіщення, підтримка кластерів, збереження історичних даних).

Потреба у локальному моніторингу комп’ютерних систем



Рис. 1.13 – Сценарії використання систем моніторингу

Табл. 1.1 – Класифікація систем моніторингу комп’ютера

Критерій	Приклад	Призначення
За об’єктом моніторингу	Системні ресурси / мережа / програми	Специфіка контролю
За способом збору даних	Агентські / безагентські	Особливості впровадження
За способом подання	CLI / GUI / Web	Зручність користування

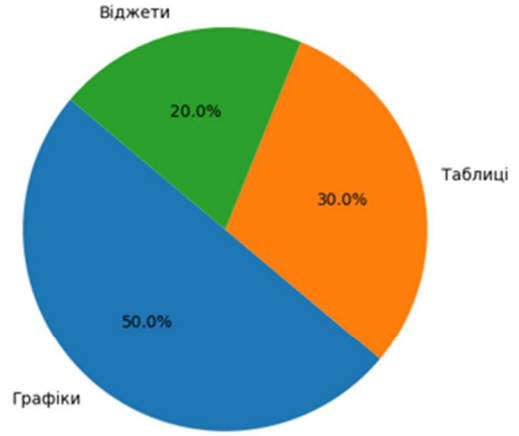


Рис. 1.9 – Типи візуалізації у співвідношені до використання: графіки, таблиці, віджети

Загальна архітектура запропонованої системи

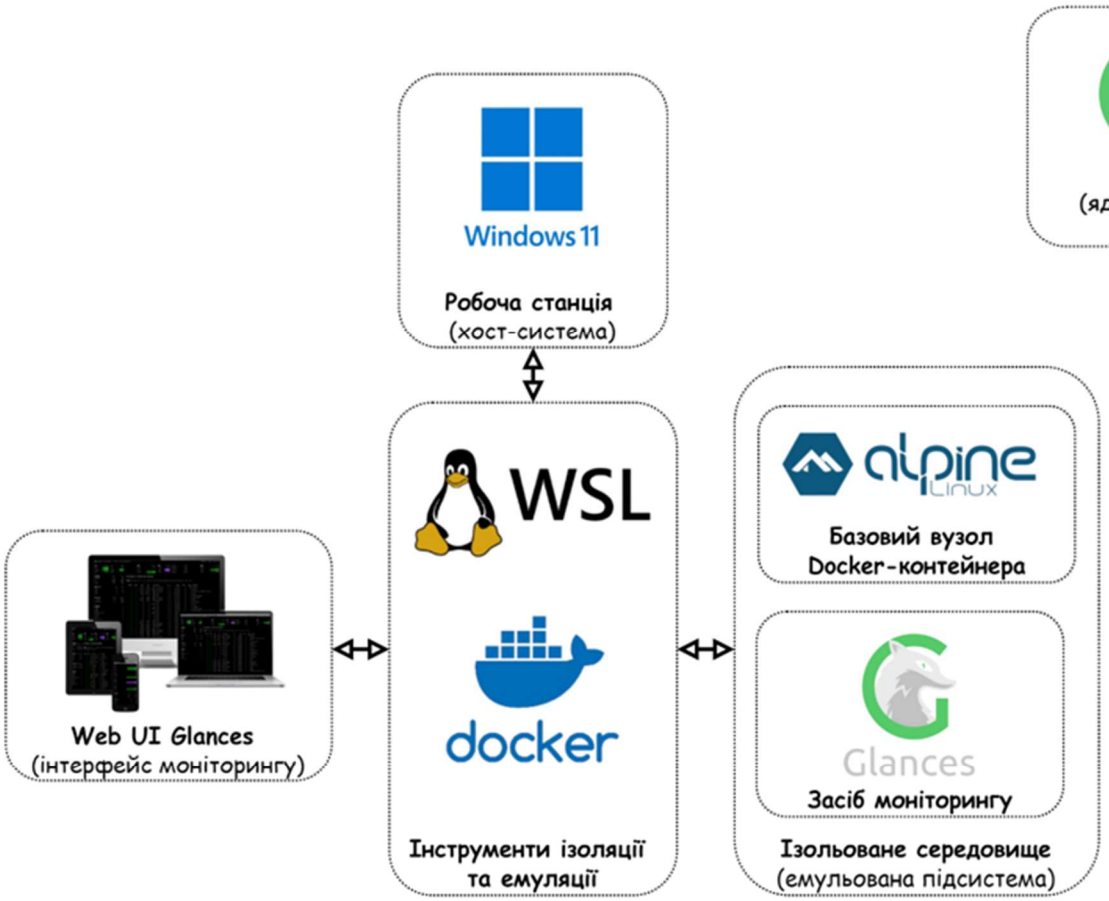


Рис. 2.2 – Архітектура системи моніторингу комп'ютерних систем на базі Glances у Docker

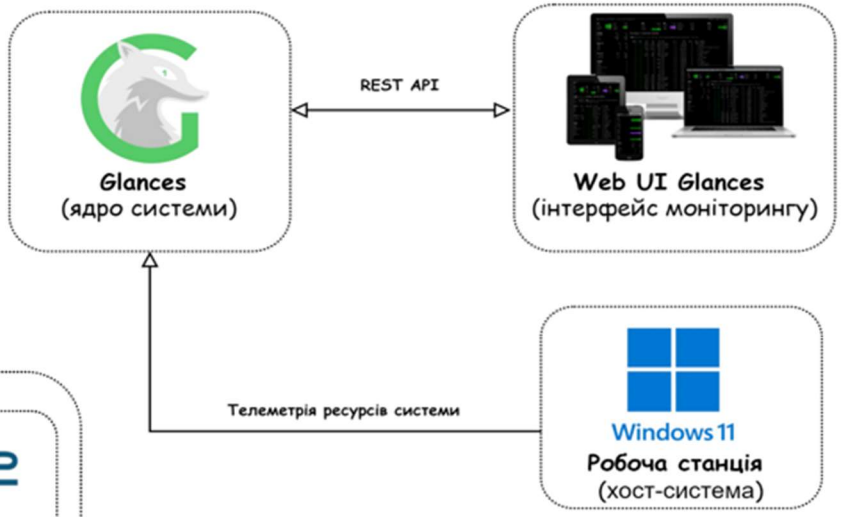


Рис. 2.3 – Взаємодія між модулями системи

Розгортання віртуального ізольованого середовища

```
docker run -d \
  --name glances \
  --restart always \
  --pid host \
  --network host \
  -v /var/run/docker.sock:/var/run/docker.sock:ro \
  nicolargo/glances:latest \
  -w
```

Рис. 2.4 – Ручна команда запуску Glances у Docker

```
version: '3'
services:
  glances:
    image: nicolargo/glances:latest
    container_name: glances
    restart: always
    pid: "host"
    network_mode: "host"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
    environment:
      - GLANCES_OPT=-w
```

Рис. 2.5 – Файл конфігурації для автоматичного розгортання Glances у Docker

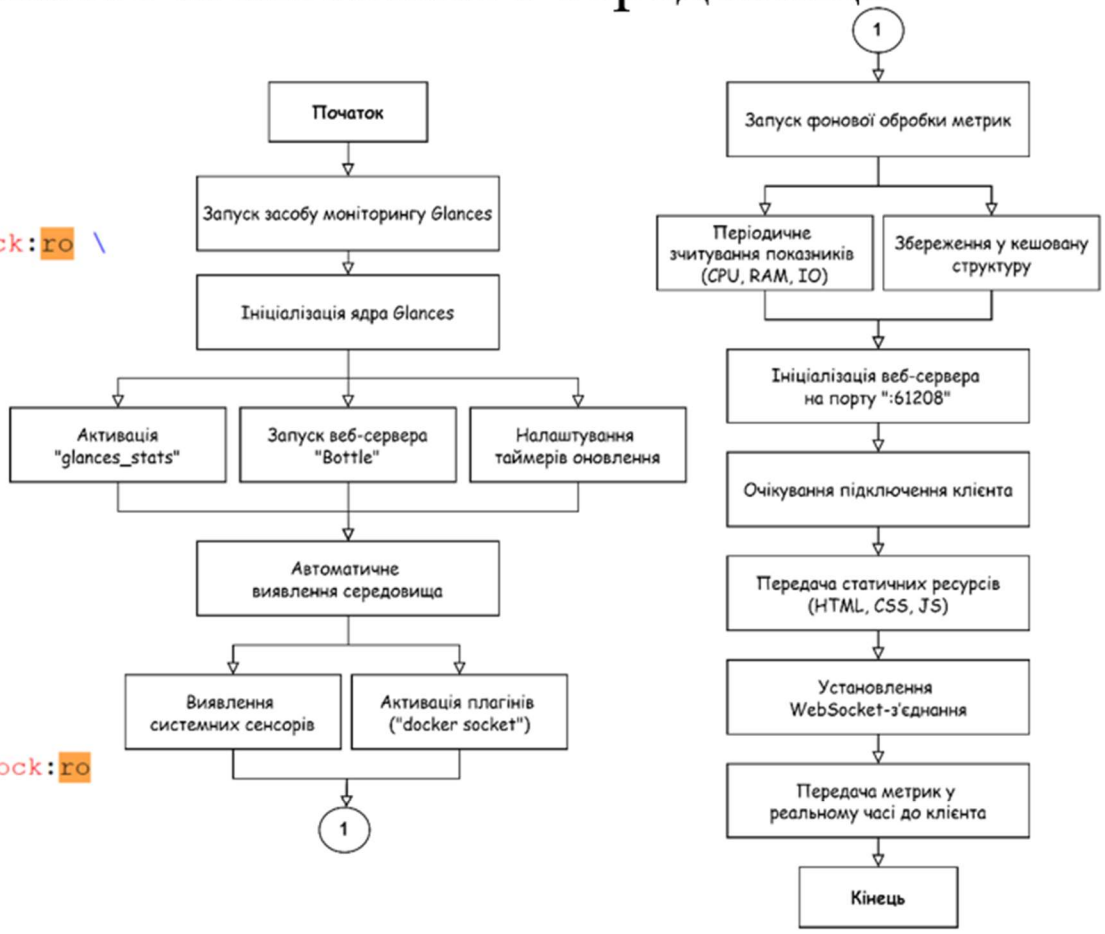


Рис. 2.7 – Алгоритм запуску та ініціалізації системи

Алгоритми функціонування у ізольованій системі

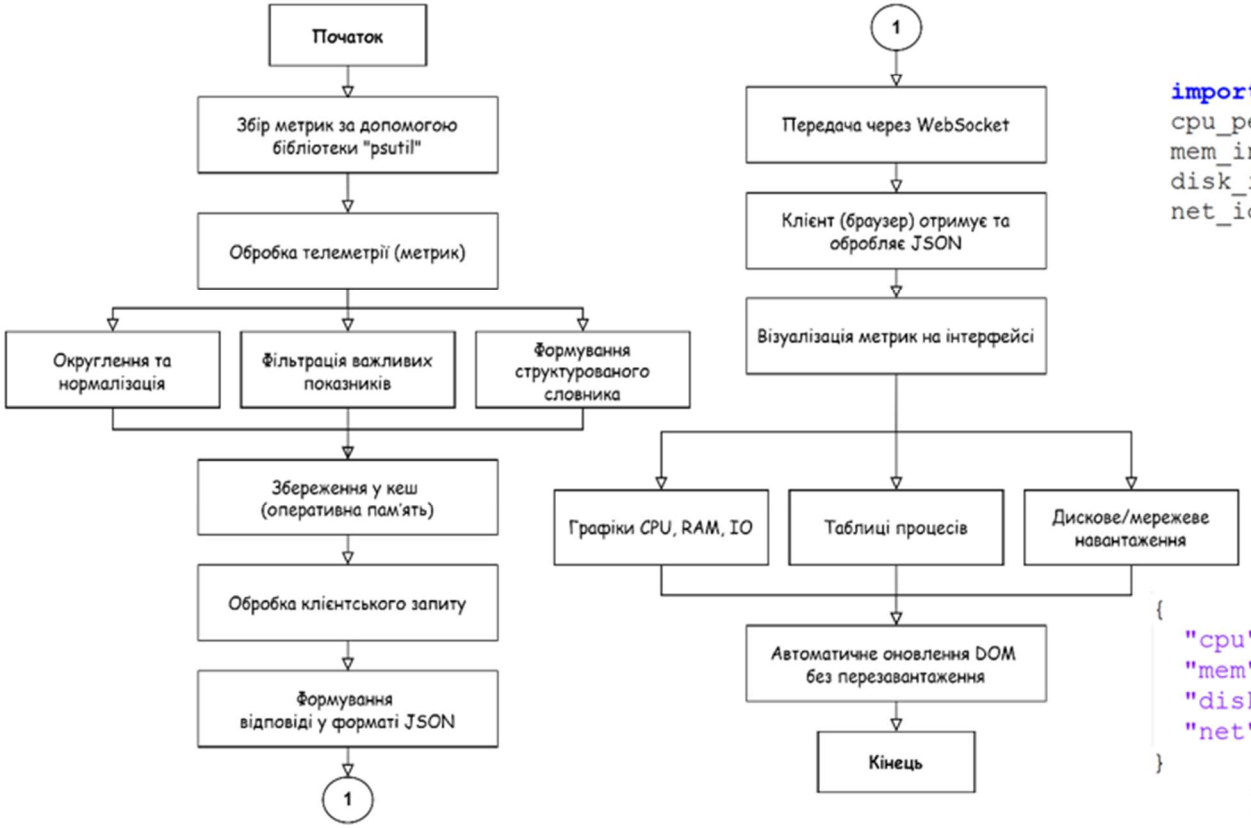


Рис. 2.10 – Алгоритм обробки даних у системі

```python
import psutil
cpu_percent = psutil.cpu_percent(interval=None)
mem_info = psutil.virtual_memory()
disk_io = psutil.disk_io_counters()
net_io = psutil.net_io_counters()
```

Рис. 2.8 – Фрагмент коду для читання системних метрик через psutil

```
{
  "cpu": {"total": 17.3},
  "mem": {"used": 2048, "total": 8192},
  "diskio": {"read_bytes": 12345678},
  "net": {"eth0": {"rx": 234567, "tx": 345678}}
}
```

Рис. 2.9 – Формування JSON-даних для Web-відображення

Демонстрація функціонування системи (веб-інтерфейс)

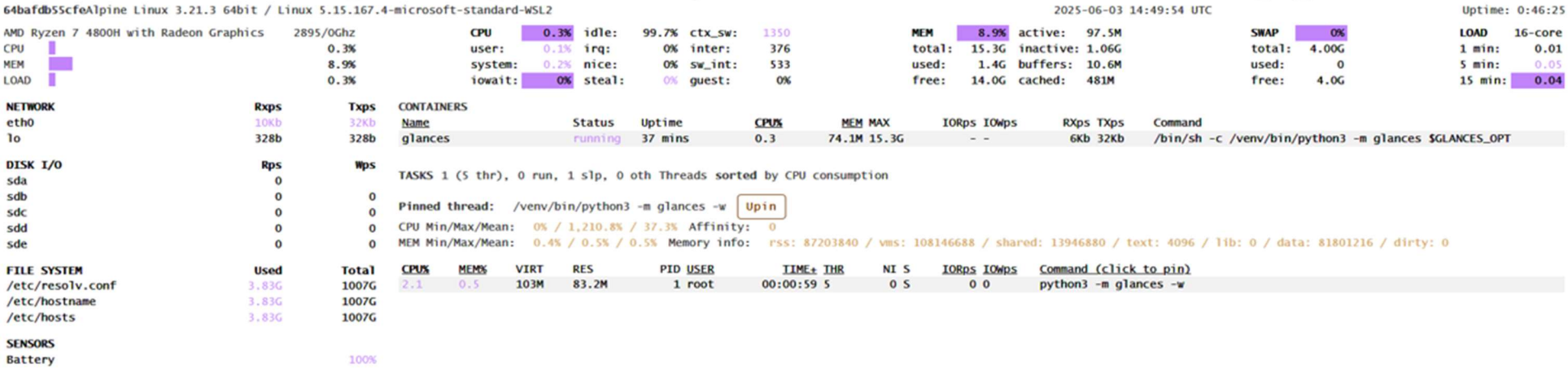


Рис. 2.12 – Веб-інтерфейс з основними метриками

```
Core 0:       +37.0°C  (high = +89.0°C, crit = +105.0°C)
Core 2:       +41.0°C  (high = +89.0°C, crit = +105.0°C)

it8720-isa-0290
Adapter: ISA adapter
in0:          +0.96 V  (min =  +0.00 V, max =  +4.08 V)
in1:          +1.54 V  (min =  +0.00 V, max =  +4.08 V)
in2:          +3.34 V  (min =  +0.00 V, max =  +4.08 V)
+5V:          +2.93 V  (min =  +0.00 V, max =  +4.08 V)
in4:          +0.00 V  (min =  +0.00 V, max =  +4.08 V)  ALARM
in5:          +3.10 V  (min =  +0.00 V, max =  +4.08 V)
in6:          +0.00 V  (min =  +0.00 V, max =  +4.08 V)  ALARM
5VSB:         +3.01 V  (min =  +0.00 V, max =  +4.08 V)
vbat:         +3.22 V
fan1:        1347 RPM  (min =    0 RPM)
fan2:           0 RPM  (min =    0 RPM)
temp1:        -55.0°C  (low  = +127.0°C, high = +127.0°C)  sensor
temp2:        +25.0°C  (low  = +127.0°C, high = +127.0°C)  sensor
temp3:        +30.0°C  (low  = +127.0°C, high = +127.0°C)  sensor
cpu0_vid:    +0.900 V
intrusion0:  ALARM
```

Рис. 2.14 – Приклад збору фізичних властивостей системи: датчик температури

Методи покращення існуючої системи

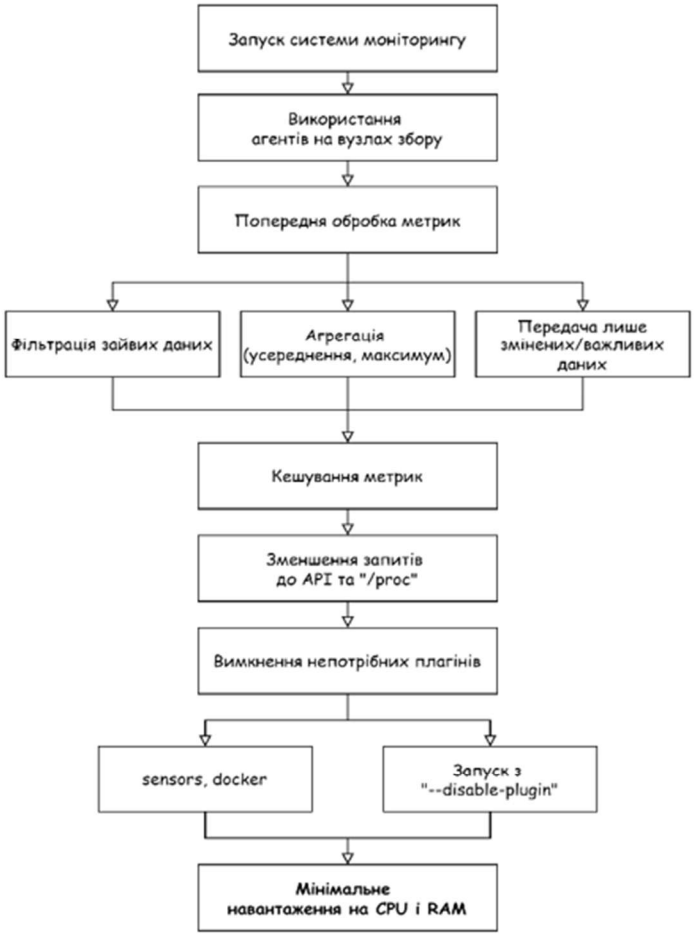


Рис. 3.2 – Метод зменшення навантаження на сервер

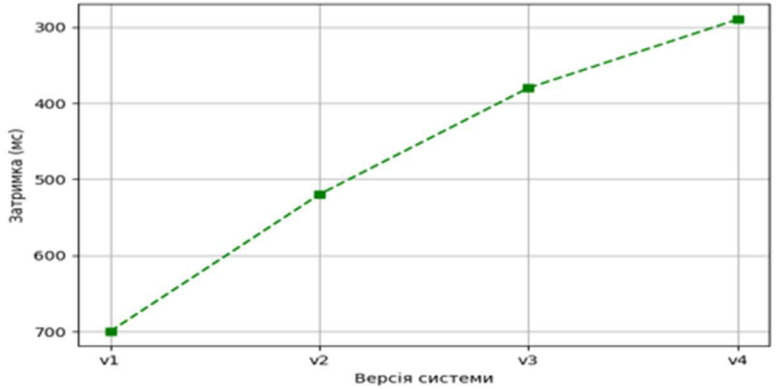


Рис. 3.1 – Графік покращення частоти оновлення телеметрії (метрик) після оптимізації

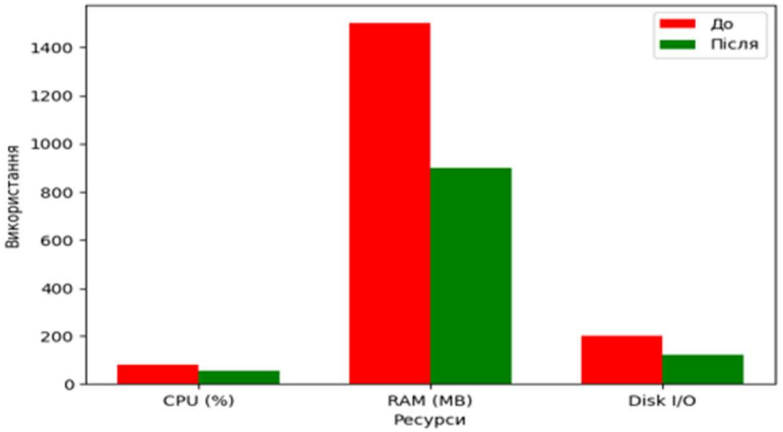


Рис. 3.6 – Порівняння використання ресурсів до та після оптимізації системи зі Glances

Методи покращення існуючої системи

```
sda9          0      0       0.
sdb1          0      0       0.
                             0.
SENSORS                      0.
acpitz 1              43C    0.
acpitz 2               0C    0.
acpitz 3              39C    0.
acpitz 4              38C    0.
acpitz 5              24C    0.
acpitz 6             127C    0.
pch_skylake 1         40C    0.
Package id 0          43C    0.
Core 0                40C    0.
Core 1                41C
```

Рис. 3.9 – Приклад додаткової телеметрії (метрик): GPU, чипсет

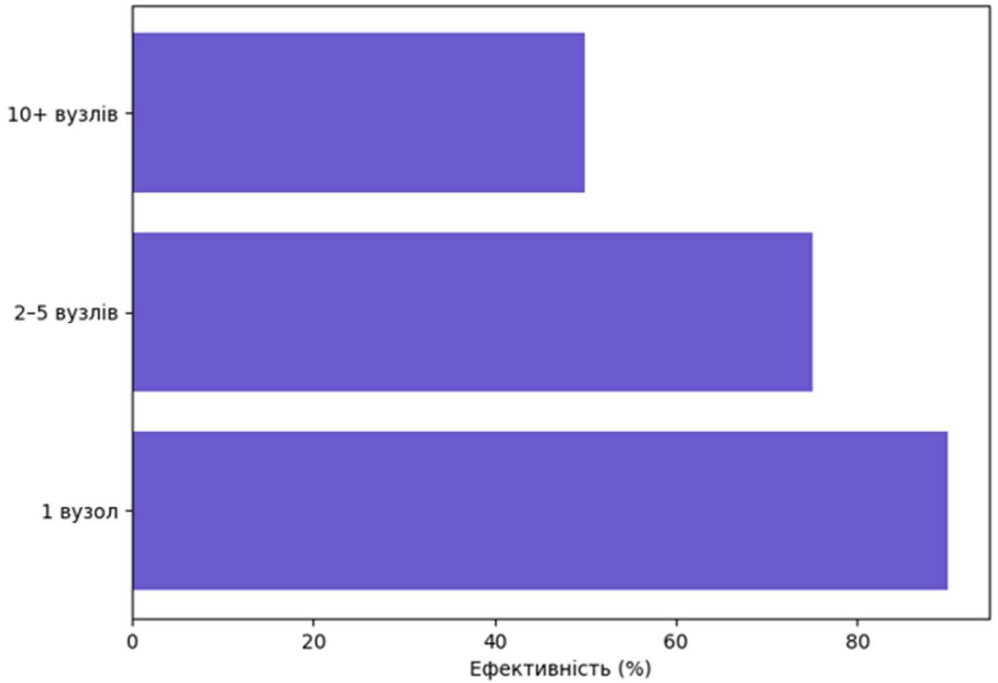


Рис. 3.10 – Процентне співвідношення різності у використанні одного агента на всіх вузлах

Порівняння запропонованої системи зі альтернативами

Табл. 3.1 – Порівняльна характеристика систем моніторингу Glances зі Prometheus + Grafana, Zabbix

Критерій	Glances	Prometheus + Grafana	Zabbix
Тип рішення	**Локальне / агентське**	Серверно-агентське з візуалізацією	Повноцінна серверно-агентська система
Принцип роботи	**Реального часу, pull-запити**	Pull-модель, збирання метрик через експортери	Переважно pull, з підтримкою trap-методу
Інтерфейс користувача	**CLI, Web (простий), API**	Web-дашборди Grafana, API, Prometheus UI	Web, конфігурований dashboard
Легкість встановлення	**Дуже проста, Docker/пакети**	Середня складність: окремі компоненти, експортери, конфігурація	Складна: потребує сервер, БД, агенти
Підтримка платформ	**Windows, Linux, macOS**	Linux, macOS, частково Windows (через WSL/WSA)	Windows, Linux, Unix
Моніторинг системних ресурсів	**Так (CPU, RAM, диск, мережа)**	Так (через Node Exporter та інші)	Так (CPU, RAM, диск, мережа)
Підтримка контейнерів/Docker	**Так (через Docker socket)**	Так (через cAdvisor, Docker Exporter)	Так (з налаштуванням)
Можливість кастомізації	**Обмежена**	Висока (PromQL, власні дашборди Grafana)	Висока (тригери, шаблони, сценарії)
Візуалізація історичних даних	**Ні (реальний час тільки)**	Так (довготривале зберігання метрик)	Так (зберігається в БД, повна історія)
Нотифікації/оповіщення	**Ні (можливе через розширення)**	Так (через Alertmanager, інтеграції)	Так (розвинена система тригерів та повідомлень)
Споживання ресурсів	**Низьке**	Середнє	Високе при великій кількості агентів
Масштабованість	**Ні (локальний моніторинг)**	Так (федерація, шардирування, автоматичне виявлення цілей)	Так (підтримує кластери, розподілену інфраструктуру)
Підтримка зовнішніх метрик (наприклад, SNMP)	**Ні**	Так (через SNMP Exporter, API, HTTP endpoints)	Так (повна підтримка SNMP, IPMI, API)
Інтеграції зі сторонніми сервісами	**Ні**	Так (Telegram, Slack, Email, Webhooks, API, Thanos, Loki)	Так (будь-які через API, webhooks)
Ліцензія	**Open Source (GNU GPL v3)**	Open Source (Apache 2.0)	Open Source (GPL v2), є комерційна версія

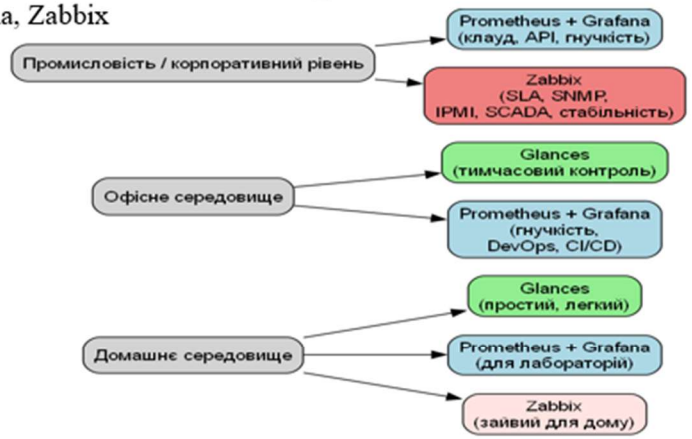


Рис. 3.7 – Сценарії застосування систем в умовах реального середовища



Рис. 3.5 – Діаграма складності розгортання різних систем

Висновки

У процесі розробки апаратно-програмного комплексу було використано Glances, що забезпечує зручний веб-інтерфейс для моніторингу основних показників системи в реальному часі. Завдяки контейнеризації за допомогою Docker було досягнуто простоти розгортання та кросплатформеності рішення. Реалізований мінімальний варіант системи (MVP) охоплює моніторинг CPU, оперативної пам’яті, дисків, мережі та інших критичних ресурсів, з можливістю подальшого розширення, наприклад, підключення температурних сенсорів.

Система показала стабільну роботу в умовах обмежених ресурсів, не створюючи значного навантаження. Простота інсталяції та доступність через браузер роблять її придатною для використання в навчальному процесі або офісному середовищі. Запропоновано шляхи покращення функціональності: реалізація сповіщень, логування метрик, налаштування порогів, а також можливість розширення системи до підтримки історичних даних.