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

КВАЛІФІКАЦІЙНА РОБОТА

Система діагностики та масштабування віртуалізованих обчислювальних ресурсів у комп'ютерних мережах.
Назва теми

Рівень вищої освіти *перший (бакалаврський)*

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

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

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

Шифр КвРКІ 2301123.23.01.17 ПЗ

Виконав здобувач ІІІ курсу, група КІ2с-23-1 _____ Андрій СТЕПАНЮК
Підпис Ініціали, прізвище

Керівник _____ Володимир ГРИГА
Науковий ступінь, учене звання Підпис Ініціали, прізвище

Нормоконтролер _____ Сергій ЛИСЕНКО
Науковий ступінь, учене звання Підпис Ініціали, прізвище

До захисту допускаю:
завідувач кафедри КІІС _____ Ольга ПАВЛОВА
«01» червня 2026 р. підпис Ініціали, прізвище
дата

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Рівень вищої освіти ПЕРШИЙ (БАКАЛАВРСЬКИЙ)

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС
_____ Ольга ПАВЛОВА

" 10 " 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Степанюку Андрію Ігоровичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Система діагностики та масштабування віртуалізованих обчислювальних ресурсів у комп'ютерних мережах.
Керівник проекту (роботи) Грига Володимир Михайлович, к.т.н., доцент.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7

2. Термін подання здобувачем роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

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

Аналіз предметної області та існуючих підходів до діагностики віртуалізованих обчислювальних ресурсів

Проєктування системи діагностики та формування рекомендацій щодо маштабування

Програмна реалізація та перевірка роботи системи діагностики

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

Блок-схема алгоритму діагностики стану вузла та формування рекомендацій

Блок-схема роботи агента збору телеметричних даних

Структурна схема компонентів системи діагностики та масштабування віртуалізованих ресурсів

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання « 10 » 01 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2026	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2026	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2026	виконано
4	Робота над розділом 2 – вибір програмних засобів та розроблення структури системи діагностики віртуалізованих обчислювальних ресурсів	01.04.2026	виконано
5	Робота над розділом 3 – реалізація та тестування системи діагностики віртуалізованих обчислювальних ресурсів	29.04.2026	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2026	виконано
7	Попередній захист ВКР	26.05.2026	виконано
8	Захист ВКР на засіданні ЕК	Червень 2026 року	

Здобувач _____ Андрій СТЕПАНЮК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Володимир ГРИГА
Підпис Ім'я, ПРІЗВИЩЕ

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 2301123.23.01.17 ПЗ	Пояснювальна записка	74		
			Графічні матеріали			
2		КвРКІ 2301123.23.01.17 Е8	Блок-схема алгоритму діагностики стану вузла та формування рекомендацій	1		
3		КвРКІ 2301123.23.01.17 Е8	Блок-схема роботи агента збору телеметричних даних	1		
4		КвРКІ 2301123.23.01.17 Е8	Структурна схема компонентів системи діагностики та масштабування віртуалізованих ресурсів	1		

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 2301123.23.01.17 ВП		
Розробив		Степанюк		1.06	Система діагностики та масштабування віртуалізованих обчислювальних ресурсів у комп'ютерних мережах Відомість проекту	Літера	Аркуш 1 / Аркушів 1
Перевір.		Грига		1.06		У	
Н. контр.		Лисенко		1.06		ХНУ, КІ2с-23-1	
Затв.		Павлова		9.06			

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Система діагностики та масштабування віртуалізованих обчислювальних ресурсів у комп'ютерних мережах ».

Автор роботи: Андрій СТЕПАНЮК.

Керівник роботи: Володимир ГРИГА.

Пояснювальна записка: 74 с., 19 рис., 4 табл., 5 дод., 46 джерел.

Графічна частина: 3 креслення.

АЛГОРИТМИ, ДІАГНОСТИКА, МАСШТАБУВАННЯ, РЕСУРСИ, РЕКОМЕНДАЦІЇ, СИСТЕМА, ВЕБ-ЗАСТОСУНОК, АГЕНТ, АНАЛІЗ СТАНУ, АРХІТЕКТУРА, БАЗА ДАНИХ.

Кваліфікаційна робота бакалавра присвячена розробці та дослідженню системи діагностики віртуалізованих обчислювальних ресурсів у комп'ютерних мережах. Актуальність теми зумовлена зростанням складності серверної інфраструктури, поширенням віртуалізації та хмарних технологій, а також підвищенням вимог до безперервності функціонування сервісів. Своєчасне виявлення ознак деградації, визначення першопричини проблеми та формування обґрунтованих рекомендацій щодо масштабування ресурсів дає змогу попереджати аварійні ситуації та підвищувати ефективність використання обчислювальних потужностей.

Метою роботи є проєктування, реалізація та тестування програмно-технічного комплексу для централізованого збору показників, автоматизованого аналізу стану вузлів, виявлення інцидентів і формування рекомендацій щодо масштабування. Для досягнення поставленої мети виконано аналіз предметної області, спроєктовано архітектуру системи, реалізовано агентську підсистему збору даних, серверну частину для обробки, механізми діагностики та формування рекомендацій, а також панель користувача.

Підпис здобувача

30.05.2026

Дата

ЗМІСТ

ВСТУП

Сучасна серверна інфраструктура все частіше використовує віртуалізовані обчислювальні ресурси. Це дозволяє ефективніше використовувати апаратні ресурси, спрощує адміністрування системи та дає можливість швидко масштабувати сервіси залежно від навантаження. Використання віртуальних машин, контейнерів, хмарних платформ і гібридних середовищ дозволяє розміщувати велику кількість сервісів на одній фізичній інфраструктурі, однак при цьому значно ускладнюється контроль за станом ресурсів і стабільністю роботи системи.

У традиційній серверній інфраструктурі користувач зазвичай працює з відносно невеликою кількістю фізичних вузлів, що дає змогу вручну оцінювати завантаження процесора, використання оперативної пам’яті, стан дискової підсистеми та мережевий трафік. На одному фізичному сервері може одночасно працювати велика кількість віртуальних машин або контейнерів, а окремі сервіси можуть бути розподілені між кількома вузлами, тому у віртуалізованих середовищах такий підхід уже не є ефективним. У таких умовах навантаження стає динамічним, а причини зниження продуктивності системи складніше визначити та проаналізувати.

Актуальність теми полягає в тому, що для забезпечення стабільної роботи комп’ютерних мереж і серверних сервісів недостатньо лише збору телеметричних даних. Також необхідно виконувати діагностику стану системи, визначати можливі першопричини інцидентів, оцінювати рівень ризику для інфраструктури та формувати практичні рекомендації щодо масштабування або оптимізації ресурсів. Система поєднує агентський збір метрик, обробку телеметричних даних на серверній частині, модуль діагностики, модуль формування рекомендацій, журнал інцидентів, планування ресурсів, інформаційну панель для візуалізації стану вузлів та засоби формування звітів. У кінцевому варіанті система розгортається як вебзастосунок: серверна частина

працює на платформі Render, клієнтська частина розміщується на Vercel, а база даних реалізована на основі віддаленого PostgreSQL-сервісу Supabase. Завдяки такому підходу інформаційна панель доступна користувачу через звичайне вебпосилання, а не лише в межах локального середовища розробника.

Об’єктом дослідження є процес моніторингу, діагностики та оцінювання стану віртуалізованих обчислювальних ресурсів у комп’ютерних мережах. Предметом дослідження є методи, алгоритми та програмно-технічні засоби збору телеметричних даних, виявлення ознак деградації, визначення першопричин виникнення проблем та формування рекомендацій щодо масштабування ресурсів.

Метою кваліфікаційної роботи є розроблення системи діагностики віртуалізованих обчислювальних ресурсів, яка забезпечує централізований збір метрик, автоматизований аналіз стану вузлів, виявлення інцидентів та формування рекомендацій щодо вертикального або горизонтального масштабування. У межах кваліфікаційної роботи було виконано аналіз віртуалізованих обчислювальних ресурсів, принципів моніторингу та діагностики серверної інфраструктури, а також існуючих систем спостережуваності. На основі цього визначено обмеження наявних підходів до автоматизованої діагностики та масштабування ресурсів. Спроєктовано архітектуру системи діагностики, визначено структуру обміну телеметричними даними та їх збереження. Реалізовано агентську підсистему збору даних, серверну частину, модуль діагностики та модуль рекомендацій, а також вебінтерфейс для відображення стану системи.

Практичне значення роботи полягає у створенні програмного-технічного комплексу, який може використовуватися як навчальний і демонстраційний інструмент для вивчення принципів моніторингу, діагностики та масштабування віртуалізованих ресурсів. Система не виконує небезпечних віддалених команд на вузлах і не змінює ресурси автоматично. Вона формує обґрунтовані рекомендації, залишаючи остаточне рішення за користувачем.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ ПІДХОДІВ ДО ДІАГНОСТИКИ ВІРТУАЛІЗОВАНИХ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ

1.1 Особливості функціонування віртуалізованих обчислювальних ресурсів та їх архітектура

Віртуалізація є одним із основних підходів до побудови сучасної IT-інфраструктури [1, 2]. Вона полягає у створенні логічного рівня між фізичним обладнанням і програмними системами, які використовують обчислювальні ресурси. Завдяки цьому процесорний час, оперативна пам’ять, дисковий простір і мережеві інтерфейси можуть розподілятися між кількома незалежними середовищами виконання.

Відповідно до сучасних підходів до проєктування програмних систем [3] та стандартів оцінки якості [4], віртуалізована інфраструктура повинна забезпечувати не лише продуктивність, але й передбачуваність роботи сервісів.

Найбільш поширеними формами віртуалізації є серверна, контейнерна, мережева віртуалізація та віртуалізація сховищ. Серверна віртуалізація базується на використанні гіпервізора, який керує віртуальними машинами та розподілом ресурсів між ними. До таких технологій належать KVM, VMware ESXi [2], Microsoft Hyper-V та Xen. Контейнерна віртуалізація, на відміну від серверної, використовує спільне ядро операційної системи, але забезпечує ізоляцію процесів, файлової системи та мережевого середовища. До основних рішень належать Docker [5, 6] і containerd, а також Kubernetes [7, 8] як система для керування та автоматичного розгортання контейнерів [9, 10].

Віртуалізований обчислювальний ресурс можна розглядати як логічний вузол із власними параметрами: кількість виділених процесорних ядер, обсяг оперативної пам’яті, дисковими обмеженнями, мережевими характеристиками, роль в системі, операційну систему та набором сервісів. У проєкті такий ресурс представлено сутністю Node. Система підтримує два основні типи вузлів: Agent Node для реального ПК або сервера, який надсилає метрики через агент, і Virtual

Node для змодельованого ресурсу, що використовується у демонстраційних сценаріях.

Особливістю віртуалізованого середовища є можливість перевиділення ресурсів. Це означає, що сумарний обсяг ресурсів, виділений віртуальним машинам, може перевищувати фізично доступні ресурси хост-системи. Такий підхід дозволяє підвищити щільність розміщення навантаження, однак створює ризик конкуренції за ресурси у випадку, коли декілька вузлів одночасно починають активно використовувати процесор або оперативну пам’ять. Наприклад, якщо фізичний сервер має 16 процесорних ядер, користувач може виділити різним віртуальним машинам сумарно 24 віртуальні ядра, розраховуючи на те, що пікове навантаження не виникатиме одночасно. Якщо така умова не виконується, це призводить до зниження продуктивності системи.

У віртуалізованих системах важливо розрізняти виділення ресурсів та їх фактичне використання. Виділення ресурсів показує, який обсяг процесорного часу, оперативної пам’яті або інших ресурсів було призначено вузлу, тоді як фактичне використання відображає, яку частину цих ресурсів вузол реально використовує під час роботи. Наприклад, вузол із 8 процесорними ядрами та середнім навантаженням процесора на рівні 5% може використовувати ресурси неефективно, тоді як вузол із 2 процесорними ядрами та навантаженням 95% уже потребує масштабування. Саме тому система моніторингу повинна враховувати не лише поточні значення метрик, а й загальний контекст роботи вузла: його призначення, встановлені обмеження, історію навантаження та динаміку зміни показників системи.

Архітектура віртуалізованої інфраструктури зазвичай складається з кількох рівнів. На базовому рівні знаходиться фізичне обладнання: процесори, оперативна пам’ять, дискова підсистема та мережеві інтерфейси. Над ним працює рівень віртуалізації або контейнеризації, який забезпечує створення та ізоляцію віртуальних середовищ. Вище розташовані віртуальні вузли, операційні системи, сервіси, бази даних [11, 12], програмні інтерфейси [13, 14],

вебзастосунки [15, 16], а також з використанням сучасних веб-стандартів, описаних у MDN Web Docs [17] та фонові процеси обробки даних. Над прикладним рівнем зазвичай працюють системи балансування навантаження, сервіси виявлення вузлів, шлюзи взаємодії між сервісами та системи моніторингу інфраструктури [18, 19, 20].

У таких умовах користувач працює з великою кількістю взаємопов'язаних компонентів інфраструктури. Підвищення затримки роботи вебсервісу може бути спричинене перевантаженням процесора, нестачею оперативної пам'яті, перевантаженням дискової підсистеми, проблемами мережевої взаємодії, помилками в коді застосунку, деградацією роботи бази даних або збоями зовнішнього програмного інтерфейсу. Саме тому діагностика повинна не лише реагувати на проблему, а й аналізувати причини її виникнення. Система має не просто відображати графік навантаження процесора, а визначати, чи є цей показник першопричиною проблеми, чи лише наслідком іншого інциденту.

У межах кваліфікаційної роботи віртуалізоване середовище розглядається як сукупність логічних вузлів, які генерують телеметричні дані. Кожен вузол має набір ресурсних показників: рівень навантаження на процесор, використання оперативної пам'яті, рівень використання дискової підсистеми, швидкість читання та запису даних на диск, швидкість надсилання та отримання мережевих даних, середній рівень навантаження системи, кількість активних процесів, час безперервної роботи системи [21], температуру, затримку роботи сервісів та рівень помилок сервісів. Такий набір показників дозволяє оцінювати як стан інфраструктури, так і частково стан сервісів, що працюють на вузлі.

Для масштабування віртуалізованих ресурсів використовуються два основні підходи: вертикальне та горизонтальне масштабування. Вертикальне масштабування передбачає збільшення ресурсів окремого вузла, наприклад додавання процесорних ядер або оперативної пам'яті. Горизонтальне масштабування передбачає додавання нових вузлів або нових екземплярів сервісу. Обидва підходи мають свої обмеження. Вертикальне масштабування

виконується швидше, проте обмежується фізичними ресурсами серверної системи. Горизонтальне масштабування ефективно працює для сервісів без збереження стану, однак потребує балансування навантаження, синхронізації даних та правильно побудованої архітектури застосунку [22].

Для вузлів баз даних ситуація є складнішою, оскільки збільшення кількості процесорних ядер або обсягу оперативної пам’яті не завжди вирішує проблему. Причиною низької продуктивності можуть бути затримки роботи дискової підсистеми, блокування транзакцій, неоптимізовані SQL-запити [14], необхідність виконання міграцій сховища даних [23] або недостатній обсяг кешу. Для сервісних вузлів масштабування є ефективнішим у випадках, коли застосунок підтримує паралельну роботу декількох екземплярів одночасно. Саме тому система формування рекомендацій повинна враховувати роль вузла та особливості його навантаження.

Таким чином, віртуалізовані обчислювальні ресурси характеризуються динамічним розподілом процесорних ресурсів, оперативної пам’яті, дискового простору та мережевої пропускної здатності. Для таких середовищ характерна можливість перевиділення ресурсів і виникнення конкуренції між вузлами за ресурси фізичного сервера. Продуктивність віртуальних вузлів залежить не лише від їхньої конфігурації, а й від стану системи та навантаження сусідніх застосунків. У таких умовах виникає потреба у постійному зборі телеметричних даних та аналізі історії метрик, а не лише поточних значень. Також важливо враховувати різницю між виділеними та фактично використаними ресурсами, оскільки це дозволяє точніше оцінювати ефективність використання інфраструктури. Додатково необхідно враховувати, що підходи до масштабування можуть відрізнятися залежно від типу вузла та причини погіршення продуктивності системи.

Ці особливості визначають основні вимоги до системи діагностики. Вона повинна мати модульну структуру, побудовану на основі фундаментальних патернів проєктування [3], підтримувати різні типи вузлів, зберігати історію

телеметричних даних, визначати першопричину проблеми, формувати підтвердження виявлених інцидентів та надавати практичні рекомендації щодо подальших дій користувача [24].

Окремо слід зазначити, що у віртуалізованих середовищах часто виникає непряма залежність між ресурсами. Наприклад, нестача оперативної пам’яті може призводити до активного використання пам’яті підкачки, а це своєю чергою збільшує навантаження на дискову підсистему та затримки роботи сервісів. Зовні така ситуація може виглядати як проблема дискової системи, хоча фактично першопричиною є нестача оперативної пам’яті. Аналогічно, високе навантаження на процесор може бути наслідком великої кількості мережевих запитів, неефективної обробки даних, криптографічних операцій або механізму повторних запитів на рівні застосунку. Саме тому система діагностики не повинна аналізувати кожну метрику окремо, а має враховувати взаємозв’язок між різними показниками системи [25].

Ще однією характерною рисою віртуалізованої інфраструктури є багаторівнева відповідальність. Частина проблем може виникати на рівні фізичного сервера, частина на рівні системи віртуалізації, частина всередині гостьової операційної системи [26], а частина у прикладному сервісі. Для користувача важливо швидко визначити, на якому саме рівні ймовірно знаходиться вузьке місце системи.

У межах кваліфікаційної роботи основну увагу приділено рівню вузла, тобто аналізу стану гостьової системи або сервера. Це означає, що система аналізує доступні вузлу метрики та формує висновки щодо навантаження процесора, використання оперативної пам’яті, використання пам’яті підкачки на диску, стану дискової підсистеми, мережевої активності та сервісних показників. Такий рівень аналізу є достатнім для практичної підтримки прийняття рішень, хоча у великих промислових середовищах його можна додатково розширити метриками системи віртуалізації та програмними інтерфейсами хмарної інфраструктури [27, 28].

У контексті комп’ютерних мереж віртуалізовані ресурси також залежать від мережевої топології. Один сервіс може звертатися до бази даних, кешу, брокера повідомлень [29], сховища об’єктів або зовнішніх програмних інтерфейсів. Зростання мережевих затримок або обсягу передавання даних може спричинити деградацію роботи навіть тоді, коли навантаження на процесор і використання оперативної пам’яті залишаються в межах норми. Саме тому в моделі телеметрії передбачено показники швидкості надсилання та отримання мережевих даних. Ці метрики не дають повної картини стану мережі [30, 31], але дозволяють визначити вузол, для якого мережевий трафік став основним фактором навантаження.

У віртуалізованих системах було реалізовано контроль за життєвим циклом вузлів. Кожен вузол проходить кілька чітких етапів: створення, очікування підключення агента, перехід у робочий режим після отримання сигналу активності, а також зміна стану на попереджувальний або критичний після перевірки, аж до повного видалення чи заміни. Для керування цими процесами в моделі вузла було застосовано такі параметри: статус, час останнього сигналу активності, тип вузла, роль, назва та версія операційної системи, а також версія агента. Це дозволило не просто збирати дані про ресурси, а чітко бачити й оцінювати поточний робочий стан кожного вузла.

З погляду масштабування важливо розрізняти короткочасні сплески навантаження та тривалі перевантаження. Короткочасний сплеск використання процесора може бути звичним для фонового завдання, процесу резервного копіювання або компіляції. Стабільне перевищення завантаження процесора протягом кількох вимірювань свідчить про нестачу процесорних ресурсів або неефективне навантаження. Саме тому в розробленій системі використано не лише останнє виміряне значення, а й вікно з кількох попередніх значень. Для окремих станів, наприклад для неповного використання ресурсів, аналізується тривала поведінка системи, а не одноразове вимірювання.

У реальних хмарних середовищах вартість ресурсів є важливим чинником. Надмірно виділені ядра процесора або оперативна пам'ять призводять до фінансових витрат, навіть якщо фактично ці ресурси не використовуються. З іншого боку, недостатність ресурсів спричиняє погіршення рівня обслуговування та збитки через простій або повільну роботу сервісу. Тому система діагностики має підтримувати баланс між продуктивністю та економічною ефективністю. Розроблена система AutoInfraDiag враховує це через сценарій виявлення неповного використання ресурсів та формування рекомендацій щодо зменшення виділених потужностей [19].

1.2 Принципи моніторингу та діагностики серверної інфраструктури

Моніторинг серверної інфраструктури це процес регулярного збирання, збереження, аналізу та візуалізації даних про стан апаратних, віртуальних і програмних компонентів системи [18, 19, 20]. Його метою є своєчасне виявлення відхилень, запобігання інцидентам, підтримання продуктивності та забезпечення передбачуваної роботи сервісів. Згідно з принципами програмної інженерії [32, 33], така система повинна проходити повний життєвий цикл: від збору вимог до супроводу.

У класичному розумінні моніторинг відповідає на питання «що відбувається». Наприклад, система показує, що завантаження процесора досягло 92 %, використання оперативної пам'яті становить 88 %, а швидкість запису на диск суттєво зросла. Діагностика відповідає на глибше питання «чому це відбувається і що з цим робити». Тому діагностика використовує дані моніторингу як вхідну інформацію, але доповнює їх правилами, пороговими значеннями, статистичним аналізом, розподілом станів за типами та механізмами формування рекомендацій [24, 25].

У класичному розумінні моніторинг показує поточний стан системи. Наприклад, система інформує, що завантаження процесора досягло 92 %,

використання оперативної пам'яті становить 88 %, а швидкість запису на диск суттєво зросла. Діагностика ж дає змогу зрозуміти причини такого стану та визначити необхідні дії для його виправлення. Тому діагностика використовує дані моніторингу як вхідну інформацію, але доповнює їх правилами, пороговими значеннями, статистичним аналізом, розподілом станів за типами та механізмами формування рекомендацій [24, 25].

Основними типами даних спостереження у серверній інфраструктурі є метрики, журнали подій та трасування запитів [24, 29]. Метрики є числовими показниками, які можна збирати в групи та аналізувати в часі. Журнали подій є текстовими записами, що описують роботу програм і системних компонентів. Трасування запитів є даними про проходження запиту між сервісами в розподіленій системі. У кваліфікаційній роботі основний акцент зроблено на метриках, оскільки саме вони безпосередньо характеризують використання обчислювальних ресурсів і придатні для автоматизованого аналізу.

Для діагностики ресурсного стану вузла важливими є кілька груп метрик [21, 25]. До них належать метрики процесора, а саме завантаження процесора, кількість ядер та середнє значення навантаження. Також враховуються метрики оперативної пам'яті: загальний обсяг, використана та доступна пам'ять, а також відсоток її використання. Метрики файлу підкачки включають загальний і використаний обсяг та відсоток його використання. Важливими є метрики дискової підсистеми: загальний обсяг диска, використаний простір, кількість записаних і прочитаних байтів, а також швидкість читання та запису. Мережеві метрики охоплюють обсяг відправлених і отриманих даних та відповідні швидкості передачі. Серед системних метрик враховуються кількість процесів, час безперервної роботи вузла та температура. Окрему групу становлять метрики сервісів, зокрема затримка відповіді, частота помилок та додаткові поля довільного формату.

Збір метрик може здійснюватися кількома способами [18, 20]. Підхід з використанням агента передбачає встановлення невеликої програми-агента на

контрольований вузол. Агент має доступ до локальних системних метрик і періодично надсилає їх на центральний сервер. Підхід без використання агента використовує зовнішні протоколи або програмні інтерфейси, наприклад SNMP, WMI, SSH або програмний інтерфейс хмарного провайдера. У кваліфікаційній роботі використано підхід з агентом для реальних вузлів, оскільки він забезпечує достатню деталізацію, простоту реалізації та незалежність від конкретного хмарного провайдера. Для забезпечення сумісності з різними системами збору телеметрії доцільно використовувати стандартизовані підходи, такі як OpenTelemetry [34].

Важливим принципом моніторингу є регулярність збору даних. Якщо інтервал між вимірюваннями занадто великий, система може пропустити короткочасні сплески навантаження. Якщо інтервал занадто малий, зростає навантаження на мережу, серверну частину та базу даних. У розробленій системі інтервало роботи агента становить 5 секунд. Це дає змогу отримати достатню деталізацію для демонстраційних сценаріїв і водночас не створює надмірних додаткових витрат ресурсів.

Ще одним принципом є централізація обробки. Метрики від багатьох вузлів мають надходити до єдиного серверного рівня, де вони нормалізуються, зберігаються та аналізуються. Це дає змогу порівнювати вузли між собою, будувати спільну панель моніторингу, створювати журнал інцидентів і формувати рекомендації з урахуванням контексту середовища.

Діагностика серверної інфраструктури зазвичай базується на поєднанні кількох методів [19, 24, 25]. До них належить аналіз на основі правил, тобто перевірка метрик за попередньо визначеними пороговими значеннями. Також використовується аналіз тенденцій, що передбачає оцінку зміни метрик у часі. Застосовується виявлення аномалій, тобто пошук нетипових комбінацій показників. Важливим методом є аналіз першопричин, який дає змогу визначити ймовірне джерело погіршення роботи системи. Також виконується кореляційний аналіз для оцінки взаємозв'язку між різними метриками. Крім того,

застосовується планування потужностей, тобто прогнозування моменту, коли показники досягнуть порогових значень у майбутньому.

У розробленій системі реалізовано діагностичний механізм із додатковим виявленням аномалій [25]. Для виявлення аномалій використано метод, який знаходить рідкісні та нехарактерні комбінації показників, за умови наявності достатньої кількості метрик. Якщо історія спостережень ще коротка, система переходить на аналіз за правилами. Такий підхід є прагматичним для кваліфікаційної роботи, оскільки логіка на основі правил є прозорою та зрозумілою, а механізм машинного навчання дає додатковий сигнал про нетипову поведінку.

Для практичного використання системи діагностики важливою є зрозумілість її рішень. Користувач повинен бачити не лише кінцевий статус, наприклад «критичний» або «попередження», а й докази: які саме метрики перевищили порогові значення, які показники було зафіксовано, який ризик оцінено та наскільки впевнено зроблено висновок. У роботі це реалізовано через поле з доказами у записі діагностики, де зберігаються метрики, порогові значення, пояснення та результат виявлення аномалій.

У процесі діагностики важливо уникати хибних рекомендацій. Наприклад, якщо сервіс працює повільно через обмеження продуктивності дискової системи, рекомендація додати процесорні потужності може не дати бажаного ефекту. Якщо проблема пов'язана з перевантаженням мережі, збільшення обсягу оперативної пам'яті також не вирішить першопричину. Тому механізм діагностики повинен спочатку визначити першопричину, а механізм формування рекомендацій має пропонувати дії відповідно до цієї причини [19, 22].

Також важливим є життєвий цикл інциденту. Якщо система виявила погіршення роботи, вона створює інцидент. Якщо наступна діагностика показує, що стан повернувся до норми, відкриті інциденти мають бути закриті або позначені як вирішені. Це дає змогу відрізняти поточні проблеми від подій, що

сталися в минулому, а також використовувати журнал інцидентів для аналізу стабільності інфраструктури.

Отже, ефективна система моніторингу та діагностики має виконувати багато функцій. Вона повинна забезпечувати збір системних і сервісних метрик, а також перевірку автентичності джерела даних спостереження. Система має зберігати історію показників, будувати панель моніторингу та графіки, виявляти відхилення й аномалії. Важливими функціями є визначення першопричини, формування доказів, створення інцидентів та формування рекомендацій щодо масштабування або оптимізації. Крім того, система повинна підтримувати експорт даних і створення звітів [18, 19, 20].

Важливим аспектом моніторингу є вибір показників, які справді мають діагностичну цінність. Надмірна кількість метрик ускладнює аналіз і може створювати завади. Недостатня кількість метрик призводить до неправильних висновків. У роботі обрано набір, який охоплює базові області ресурсів: процесор, оперативну пам'ять, додаткову пам'ять, сховище даних, мережу та стан сервісів. Такий підхід є достатнім для визначення більшості типових вузьких місць у невеликому або середньому серверному середовищі [21].

Показник використання процесора сам по собі не завжди є достатнім. Наприклад, завантаження процесора на рівні 90 % на вузлі з одним ядром і таке саме завантаження на вузлі з 16 ядрами мають різний операційний контекст. Тому разом із показником використання процесора зберігається кількість ядер процесора та середнє значення навантаження. Середнє значення навантаження допомагає оцінити чергу виконання завдань, особливо в системах, подібних до Unix. Якщо середнє значення навантаження суттєво перевищує кількість ядер, це може свідчити про утворення черги та конкуренцію за процесор [26].

Показники пам'яті також потребують комплексного аналізу. Високе використання оперативної пам'яті не завжди є проблемою, оскільки операційні системи можуть використовувати пам'ять для тимчасового зберігання даних. Тому важливо враховувати доступну оперативну пам'ять і відсоток

використання додаткової пам'яті. Якщо використання оперативної пам'яті високе, але доступна пам'ять достатня і додаткова пам'ять не використовується, ситуація може бути нормальною. Якщо ж використання оперативної пам'яті високе, доступна пам'ять низька, а використання додаткової пам'яті зростає, це є вагомим свідченням нестачі пам'яті [26].

Показники диска поділяються на показники заповнення та показники інтенсивності обміну даними. Відсоток заповнення диска показує, наскільки заповнений диск. Швидкість читання та запису диска показує інтенсивність операцій введення та виведення. Переповнення диска може призвести до аварійного завершення служб, неможливості запису журналів подій або зупинки бази даних. Висока інтенсивність обміну даними може створювати затримки навіть тоді, коли вільного місця достатньо. Тому в системі зберігаються обидва типи показників.

Показники мережі дають змогу виявляти вузли, які обробляють великий обсяг переданих даних. Для веб служб або програмних служб це може бути звичним у години пікового навантаження, але якщо швидкість передачі даних супроводжується зростанням затримок або частоти помилок, потрібно перевіряти розподіл навантаження, залежності між складовими частинами системи або обмеження пропускної здатності [30, 31]. Програма визначає нестачу мережевих ресурсів тоді, коли швидкість передачі даних висока, а використання процесора та оперативної пам'яті не є критичним. Це допомагає не плутати вузьке місце мережі з нестачею обчислювальних ресурсів або пам'яті.

Показники служб, такі як затримка відповіді та частота помилок, виконують роль сполучної ланки між рівнем інфраструктури та рівнем застосунку. Показники ресурсів можуть виглядати нормальними, але служба все одно може погіршувати роботу через помилки самого застосунку, зовнішні залежності або логічні проблеми. У такому разі система діагностики може визначити стан як погіршення роботи служби та не рекомендувати збільшення ресурсів без додаткових доказів.

Окрему роль відіграє часовий стан. Для керування інфраструктурою важливо знати не лише те, які показники зараз, а й те, як вони змінювалися. Якщо використання оперативної пам'яті повільно зростає протягом тривалого часу, це може свідчити про витік пам'яті. Якщо використання процесора різко зросло після оновлення застосунку, причина може бути у новій версії застосунку. Якщо заповнення диска стабільно зростає, потрібно планувати розширення сховища. У програмі частково втілено цей підхід через механізм планування місткості [24].

У системах спостереження часто виникає втома від попереджень. Це стан у якому користувач отримує надто багато сповіщень і перестає реагувати на них правильно. Щоб уникнути цього, розроблена система не просто створює сповіщення на кожне перевищення порогового значення, а формує структурований запис діагностики та інцидент. Це робить подію більш змістовною: вона має першопричину, ступінь важливості, показник ризику та підтвердження [19].

1.3 Аналіз наявних систем спостереження для стеження за станом

Існує значна кількість систем спостереження для стеження за станом. Вони відрізняються способом збирання даних, побудовою, способом зберігання, можливостями показу даних, створенням сповіщень, поєднанням з іншими системами та рівнем самостійності діагностики. Найбільш відомими є Prometheus, Grafana, Zabbix, Nagios, Datadog, New Relic, Elastic Observability.

Prometheus є однією з найпоширеніших вільних систем спостереження для хмарних середовищ [19]. Вона збирає показники у вигляді часових рядів, використовує спосіб отримання даних за запитом і підтримує мову запитів PromQL. Prometheus добре підходить для середовищ Kubernetes [7, 8], різних служб і систем, які передають показники через веб-точку доступу. Її перевагами є простота поєднання з іншими системами, велика кількість додаткових засобів збирання показників, підтримка правил сповіщення та популярність у практиках

розробки та супроводу. Обмеженням Prometheus є те, що вона переважно зосереджена на збиранні, зберіганні та отриманні показників, а не на повністю самостійному визначенні першопричин (рис 1.1).

Рисунок 1.1 – Інтерфейс додатка Prometheus та Gafana [35]

Grafana зазвичай використовується як система показу даних спостереження [36]. Вона підтримує панелі спостереження, різні види відображення, змінні, сповіщення, різні джерела даних і способи створення звітів. Grafana може працювати з Prometheus, Loki, Elasticsearch, базами даних SQL, хмарними службами. Її сильною стороною є гнучке відображення даних та зручний користувацький вигляд. Водночас Grafana здебільшого показує дані та створює сповіщення, але не завжди самостійно визначає першопричину проблеми або пропонує чіткий план розширення потужностей (рис. 1.1). Для побудови ефективних графіків та діаграм можуть використовуватися бібліотеки візуалізації, як-от Apache ECharts [37], що дозволяють створювати інтерактивні панелі.

Zabbix є системою спостереження, яка підтримує сервери, мережеве обладнання, віртуальні машини, хмарні служби, зразки налаштувань, сповіщення та розподілене спостереження [20]. Zabbix має сильні можливості

для великих корпоративних інфраструктур, зокрема спостереження з агентом і без агента. Її перевагою є широкий набір можливостей, що працюють одразу після встановлення. Недоліком є відносна складність налаштування, а також те, що глибока діагностика та поради часто залежать від правильно налаштованих вирішувачів, зразків налаштувань і досвіду керівника (рис. 1.2).

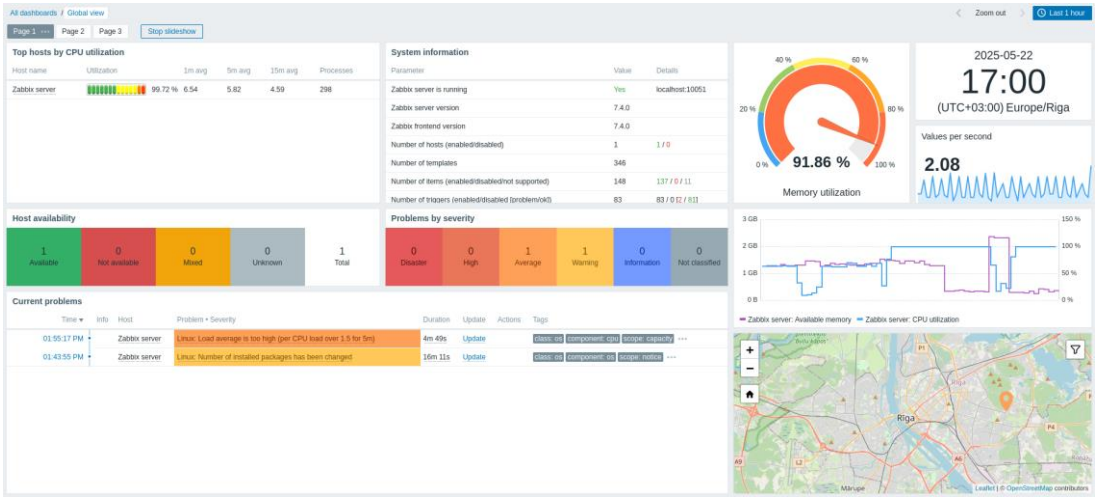


Рисунок 1.2 – Інтерфейс додатка Zabbix [38]

Nagios є класичним рішенням для спостереження за доступністю служб і вузлів [39]. Воно добре підходить для перевірки стану вузлів, портів, служб і основних показників. Однак сучасні розподілені системи потребують більш гнучкого підходу до даних спостереження, графіків, зв'язків між подіями та самостійного аналізу. Через це Nagios часто використовується разом з іншими знаряддями або поступається новітнім платформам стеження за станом (рис. 1.3). Nagios не має вбудованих механізмів для автоматичного визначення першопричин проблем або формування рекомендацій щодо масштабування. Його архітектура передбачає лише базове сповіщення про перевищення порогових значень, що змушує адміністратора самостійно аналізувати причини збоїв та приймати рішення щодо збільшення ресурсів. Тому для сучасних віртуалізованих середовищ Nagios є недостатньо ефективним без додаткових надбудов.

Рисунок 1.3 – Інтерфейс додатка Nagios [40]

Datadog і New Relic є платними платформами стеження за станом, які поєднують показники, журнали подій, трасування запитів, виявлення помилок у застосунках, панелі спостереження, виявлення аномалій та поєднання з хмарними службами [18]. Їхньою перевагою є високий рівень готовності до роботи, здатність до розширення та зручність для промислових середовищ. Однак такі системи можуть бути дорогими для навчальних, малих або дослідницьких робіт. Крім того, їхній внутрішній спосіб формування порад може бути прихованим, що не завжди підходить для кваліфікаційної роботи, де важливо показати власне втілення способу дій (рис. 1.4).

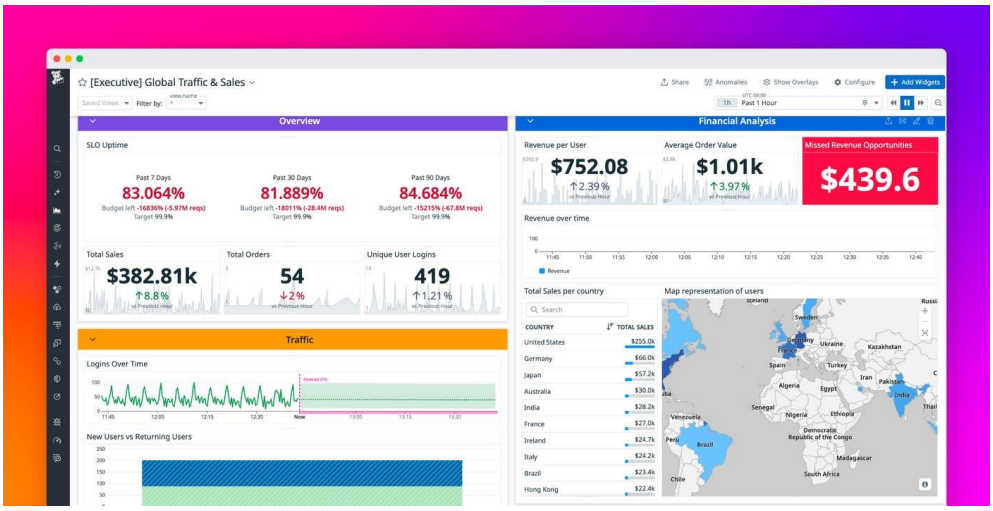


Рисунок 1.4 – Інтерфейс додатка Datadog [41]

Elastic Observability будується на основі сукупності Elasticsearch, Kibana, Beats і APM [42]. Це гарне рішення для роботи з журналами подій, показниками та трасуваннями запитів, але воно потребує складнішого розгортання і налаштування. Для створення невеликої системи діагностики з власним механізмом діагностики така платформа є завеликою.

Порівняльну характеристику розглянутих систем наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння існуючих систем моніторингу

Система	Основне призначення	Переваги	Обмеження
Prometheus	Спостереження за часовими рядами даних	Мова запитів PromQL, додаткові засоби збирання показників, сукупність знарядь для хмарного середовища	Потребує окремого додаткового шару для визначення першопричин та формування порад
Grafana	Показ даних у вигляді панелей	Зручні панелі спостереження, багато джерел даних	Основним спрямуванням є показ даних, а не діагностика
Zabbix	Корпоративне спостереження	Агенти, зразки налаштувань, сповіщення	Налаштування є складнішим, поради залежать від заданих правил
Nagios	Спостереження за доступністю	Простота перевірок служб	Обмежена модель стеження за станом для сучасних розподілених систем

Кінець таблиці 1.1

Datadog	Платна платформа стеження за станом	Показники, журнали подій, трасування запитів, виявлення помилок у застосунках	Висока вартість, закрита логіка частини функцій
Elastic Observability	Журнали подій, показники, трасування запитів	Гарний пошук і дослідження даних	Висока складність для рішення в межах кваліфікаційної роботи

Аналіз показує, що наявні системи добре вирішують завдання збирання й показу даних спостереження, але не завжди надають простий, зрозумілий і пристосований до навчального проєкту спосіб формування рекомендацій щодо масштабування. Часто користувач бачить сповіщення, але самостійно має визначити, чи потрібно збільшити обчислювальні ядра, оперативну пам'ять, покращити обмін даними з диском, перевірити шлях у мережі або виконати зменшення потужностей.

Саме тому доцільно створити власну систему, яка не намагається замінити Prometheus або Grafana, а зосереджується на зрозумілій діагностиці та порадах щодо розширення потужностей для віртуалізованих ресурсів. Розроблена система має власний шлях передавання даних спостереження, визначення першопричин, журнал інцидентів, планувальник місткості та панель спостереження українською мовою. Реалізація системи виконана з використанням мови Python [43, 44], зокрема з урахуванням сучасних практик.

Для коректної роботи програми важливо не лише порівняти можливості наявних систем, а й визначити, які побудовані ідеї можна використати у власній програмі. Від Prometheus доцільно взяти ідею показників у вигляді часових рядів і добору даних за розкладом. Від Grafana спрямування на панель спостереження

як головний вигляд для дослідження. Від Zabbix збирання даних за допомогою агента та зосереджену модель керування вузлами. Водночас розроблена система має власний головний напрям: робочий процес діагностики та формування порад.

Існуючі системи також відрізняються за моделлю збору даних. Prometheus використовує модель отримання даних за запитом: сервер опитує джерела даних. Системи на основі агентів, зокрема Zabbix або агент розробленої системи, можуть використовувати модель надсилання, коли агент самостійно передає дані. Модель надсилання є зручною, оскільки агенту достатньо знати адресу серверної частини та ключ доступу. Серверна частина не повинна мати прямого мережевого доступу до кожного вузла, що спрощує роботу в мережах з перетворенням адрес або локальних мережах.

Ще одна відмінність полягає у моделі зберігання даних. Prometheus має власне сховище часових рядів, Elastic використовує Elasticsearch, Zabbix використовує SQL-базу даних, а комерційні платформи використовують власні розподілені системи зберігання. Розроблена система використовує SQL-модель, оскільки вона є прозорою, простою у роботі та дає змогу зберігати пов'язані сутності: показники, діагностику, інциденти та рекомендації. Для великого промислового навантаження сховище часових рядів могло б бути ефективнішим, але для цілей роботи SQL-модель є виправданою.

У багатьох платформах стеження за станом виявлення аномалій і поради на основі штучного інтелекту є окремими платними або складними функціями. Їхній внутрішній механізм може бути незрозумілим. Тому механізм діагностики, що розробляється, будується на явних правилах і порогових значеннях, а складова машинного навчання використовується як додатковий сигнал.

Отже, аналіз наявних платформ показав, що створення власної системи є виправданим не через відсутність знарядь спостереження, а через потребу втілити невелике, зрозуміле й пристосоване рішення, яке показує саме процес діагностики та формування порад щодо масштабування.

1.4 Проблеми та обмеження існуючих підходів до діагностики та масштабування ресурсів

Попри розвиток систем стеження за станом, у практиці керування віртуалізованими ресурсами залишається низка проблем. Одна з них це розрив між спостереженням і підтримкою ухвалення рішень. Система може показати, що використання процесора високе, але не завжди пояснює, чи є процесор першопричиною, чи це лише наслідок іншої проблеми. Наприклад, застосунок може активно використовувати його через надмірну кількість повторних звернень до повільної бази даних. У такому разі просте збільшення обчислювальної потужності по вертикалі може лише тимчасово зменшити прояв, але не усунути першопричину [24].

Друга проблема це надмірна кількість даних спостереження. У розподілених системах навіть невелика кількість вузлів може створювати тисячі точок показників за короткий проміжок часу [19]. Якщо ці дані лише зберігаються та показуються на графіках, керівник змушений самостійно переглядати панелі спостереження, порівнювати часові ряди й робити висновки. Це збільшує середній час визначення причини несправності та підвищує ризик людської помилки.

Третя проблема це складність визначення першопричини. Показники можуть змінюватися одночасно. Наприклад, при нестачі оперативної пам'яті може зростати використання файлу підкачування, швидкість запису на диск і затримка відповіді служби. Якщо система не враховує зв'язок між цими показниками, вона може помилково визначити проблему як вузьке місце обміну даними з диском, хоча першопричиною є нестача оперативної пам'яті та активне використання файлу підкачування [26].

Четверта проблема це те що порогові значення не завжди є правильними. Для різних типів вузлів звичайні показники можуть відрізнятися. Вузол бази даних може мати вищу активність обміну даними з диском, робочий вузол може

періодично мати високе використання процесора, а вузол-перехідник високу пропускну здатність мережі. Тому порогові значення мають використовуватися не окремо, а в межах ролі вузла та разом з іншими показниками.

П'ята проблема це невідповідність рекомендацій реальній причині деградації. У багатьох установах першою дією на проблему продуктивності є збільшення ресурсів. Проте розширення потужностей має сенс лише тоді, коли проблема справді пов'язана з нестачею відповідного ресурсу. Якщо проблема пов'язана з надмірним виділенням ресурсів, затримками роботи диска або зниженням швидкодії через перегрів, то збільшення обчислювального ядра чи оперативної пам'яті може не дати очікуваного результату [22].

Шоста проблема це недостатня увага до зменшення потужностей. Спостереження зазвичай спрямоване на виявлення перевантаження, але віртуалізоване середовище також потребує виявлення неповного використання ресурсів. Якщо віртуальна машина тривалий час використовує 5–10 % процесора та менше 30 % оперативної пам'яті, ресурси можуть бути виділені надмірно. Це збільшує витрати або зменшує доступний запас для інших робочих навантажень. Тому система має формувати не лише поради щодо збільшення ресурсів, а й поради щодо ущільнення або зменшення потужностей [18].

Сьома проблема складність промислових рішень для навчальних і малих середовищ. Prometheus, Grafana, Zabbix або Elastic Observability є гарними системами, але їхнє розгортання, налаштування, підтримання правил зберігання даних, зразків налаштувань, додаткових засобів збирання показників і сповіщень є занадто складним для звичайного дослідного зразку [19, 20, 42]. Для демонстрації принципів діагностики доцільно мати компактну систему з контрольованою логікою.

Восьма проблема це безпека підходу з використанням агента. Якщо агент має змогу виконувати команди на вузлі або змінювати ресурси, це створює додаткові загрози. У кваліфікаційній роботі прийнято обмеження: агент лише збирає показники та надсилає сигнали справності або показники на серверну

частину. Він не зупиняє процеси, не виконує системних команд і не змінює налаштування вузла. Це зменшує ризик небажаного впливу на контрольований сервер.

Дев'ята проблема це необхідність зрозумілої основи для висновків. У промислових середовищах поради без доказової бази можуть бути небезпечними. Користувач має розуміти, чому система пропонує збільшити обчислювальне ядро або оперативну пам'ять. Тому кожна діагностика має містити докази: показник, поточне значення, порогове значення та пояснення [25].

Десятою проблемою є відсутність єдиного циклу від показника до діагностики, далі до інциденту, потім до поради і завершального звіту. У багатьох знаряддях ці складові втілені окремо або потребують поєднання між собою [18, 19, 20]. У кваліфікаційній роботі вони об'єднані в одному програмному комплексі, що спрощує показ і перевірку отриманих результатів.

Важливою методологічною проблемою є також визначення меж відповідальності системи. Якщо система спостереження самостійно радить дію, користувач може сприймати її як наказ до виконання. У справжньому середовищі це небезпечно, оскільки розширення потужностей може мати побічні наслідки: збільшення витрат, зміну затримок, перерозподіл навантаження, конфлікт із ліцензійними обмеженнями або вичерпання спроможності основного вузла. Тому розроблена система формує поради як підтримку ухвалення рішень. Порада містить очікуваний вплив і причину, але не виконує зміну самостійно [22].

Проблемою звичайних підходів на основі порогових значень є незмінність порогів. Наприклад, завантаження процесора більше 85 % може бути критичним для сервера програмного доступу, але нормальним для короткого фонового завдання. У програмі ця проблема частково вирішується через поєднану перевірку: враховує використання ядра та середнє значення черги завдань, вузьке місце диска або мережі перевіряється лише якщо процесор не є головною

причиною, нестача пам'яті враховує доступну пам'ять. У майбутньому систему можна доповнити пороговими значеннями, що залежать від ролі вузла.

Іншою проблемою є недостатня якість даних. Агент може тимчасово втратити зв'язок із серверною частиною, деякі показники можуть бути недоступні на певній операційній системі, температурні сенсори можуть не підтримуватися, середня завантаженість черги завдань може бути відсутня у Windows. Тому схема даних спостереження допускає відсутність значень для багатьох полів. Механізм діагностики має працювати з неповними даними та використовувати перехідну логіку для таких випадків.

Також існує проблема останнього виміряного значення. Якщо рішення ухвалюється лише на основі останньої точки, можливі помилкові висновки через випадкові скачки показників. Якщо рішення ухвалюється лише на основі тривалого середнього, система може повільно діяти у відповідь на справжні події. Розроблена система використовує змішаний підхід: частина правил перевіряє останній показник, частина середнє значення або тривалий проміжок часу, а виявлення аномалій досліджує набір останніх точок [25].

Складність масштабування полягає ще й у тому, що різні вузькі місця мають різну природу усунення. Навантаження процесора часто вирішується збільшенням процесорних потужностей або розширенням по горизонталі [22]. Концепції розподілених обчислень, закладені в ранніх роботах з Grid-систем [45], знайшли своє продовження в сучасних підходах до горизонтального масштабування. Нестача пам'яті вирішується збільшенням оперативної пам'яті або зменшенням обсягу пам'яті, яку займає застосунок. Вузьке місце обміну даними з диском вирішується зміною рівня зберігання, використанням покажчиків у базі даних, тимчасовим зберіганням даних у пам'яті або зменшенням кількості записів. Нестача мережевих ресурсів вирішується збільшенням пропускної здатності, розподілом навантаження, стисненням даних, тимчасовим зберіганням або зміною способу взаємодії між службами [30,

31]. Загроза перегріву взагалі не вирішується зміною програмного забезпечення. Через це загальна порада «додати ресурси» є неправильною.

Аналіз розглянутих систем показав, що переважна більшість з них орієнтована на професійних адміністраторів, які мають глибокі знання у сфері моніторингу. Для навчальних цілей або малих проєктів такі рішення є складними у налаштуванні та супроводі. Крім того, багато систем не надають автоматизованих рекомендацій щодо масштабування, залишаючи прийняття рішень повністю на людину. Це створює додаткове навантаження на користувача, особливо в умовах обмежених ресурсів. Відсутність чітких доказів та пояснень у готових рішеннях ускладнює довіру до їхніх висновків. Жодна з розглянутих систем не має вбудованого механізму для тестування сценаріїв деградації без реального навантаження на обладнання. Це ускладнює перевірку роботи алгоритмів діагностики в умовах, наближених до реальних.

1.5 Постановка задачі

На основі проведеного дослідження та наявних підходів сформульовано завдання кваліфікаційної роботи: розробити систему діагностики віртуалізованих обчислювальних ресурсів у комп'ютерних мережах, яка забезпечує збирання даних спостереження, дослідження стану вузлів, визначення першопричини погіршення роботи та формування порад щодо розширення потужностей або поліпшення використання ресурсів.

Система повинна складатися з багатьох складових. До них належить засіб спостереження для збирання показників на справжніх вузлах, серверний програмний вхід для приймання сигналів справності та показників, а також база даних для зберігання відомостей про середовища, вузли, показники, діагностику, інциденти, поради, передбачення місткості та звіти. Також система має містити механізм діагностики для визначення першопричин, механізм формування порад для створення практичних дій, механізм відтворення станів для змоделювання

типових сценаріїв погіршення роботи та планувальник місткості для оцінки напрямів зміни показників використання ресурсів. Крім того, до складу системи входять панель спостереження для показу стану системи та підсистема виведення даних і створення звітів.

До функціональних вимог до системи належать такі. Система має забезпечувати створення та перегляд середовищ, створення вузлів із справжніми засобами спостереження та віртуальних вузлів, реєстрацію засобу спостереження з наданням ключа доступу, а також приймання сигналів справності від засобу спостереження. Важливою вимогою є приймання даних спостереження, збереження минулих показників, запуск діагностики для вузла, а також самостійне створення інцидентів у разі виявлення погіршення роботи та самостійне створення порад. Система повинна надавати можливість перегляду діагностики, інцидентів та порад, запуску сценаріїв відтворення станів для віртуальних вузлів, запуску планування місткості, формування звітів про вузли, виведення показників у формати CSV та JSON, а також показ ситуації а панелі спостереження.

Нефункціональні вимоги до системи включають такі положення. Архітектура системи має бути побудована за модульним принципом. Система повинна забезпечувати простий місцевий запуск для розроблення та випробування, а також можливість запуску за допомогою Docker Compose у місцевому середовищі. Для місцевого режиму роботи має підтримуватися база даних SQLite, тоді як як основна віддалена база даних у кінцевому розгортанні використовується Supabase PostgreSQL [12]. Серверна частина системи має розгортатися на платформі Render, а користувацька частина на платформі Vercel [28]. Для валідації даних та управління конфігурацією на сервері використовується бібліотека Pydantic [46]. Для приймання даних від засобу спостереження використовується перевірка справжності за допомогою ключів доступу. Має бути налаштовано узгодження джерел для взаємодії між користувацькою та серверною частинами. Інтерфейс системи повинен бути

зрозумілим. Також важливою вимогою є відсутність небезпечного віддаленого виконання команд агентом.

У роботі також визначено такі обмеження. Система не виконує справжнє самостійне розширення потужностей. Вона не є повноцінною платформою стеження за станом рівня Datadog або Elastic. Система зосереджується на показниках використання ресурсів і службових показниках, а не на стеженні за проходженням запитів у розподіленому середовищі. Поради системи є підтримкою для ухвалення рішень користувачем, а не самостійним механізмом виконання дій.

У підсумку сформульована задача визначає напрям подальшого проєктування та реалізації системи, де основна увага приділяється загальній архітектурі системи діагностики, обґрунтуванню вибору технологій та середовища реалізації, розробленню структури даних спостереження та формату обміну повідомленнями, створенню алгоритмів діагностики та формування рекомендацій, моделюванню роботи системи з оптимізацією обробки даних спостереження, забезпеченню надійності та стабільності її функціонування, а також практичній програмній реалізації агентської підсистеми збору показників, серверної частини для приймання та обробки даних, механізмів діагностики, формування порад та панелі спостереження, включно з тестуванням роботи системи, аналізом результатів діагностики та оцінкою ефективності, стабільності й надійності запропонованого рішення.

2 ПРОЄКТУВАННЯ СИСТЕМИ ДІАГНОСТИКИ ТА ФОРМУВАННЯ РЕКОМЕНДАЦІЙ ЩОДО МАСШТАБУВАННЯ

2.1 Загальна архітектура системи діагностики

Система спроєктована як модульний застосунок. Її будова поділена на рівень засобу спостереження, серверний рівень, рівень зберігання даних, аналітичні служби та користувацьку панель спостереження. Такий поділ дає змогу незалежно розвивати збирання показників, логіку діагностики, програмний вхід та користувацький вигляд.

Основними складовими системи є Remote Agent, який є програмою на Python, що встановлюється на справжній вузол і збирає показники за допомогою бібліотеки psutil [21]. Також до складу входить Backend API на основі FastAPI [13], який приймає дані спостереження, керує вузлами, запускає діагностику та передає дані користувацькій частині. База даних представлена Supabase PostgreSQL [12] у кінцевому розгортанні та SQLite у локальному режимі розробки. Механізм діагностики відповідає за визначення першопричини, ступеня важливості, впевненості у висновку та показника ризику [24, 25]. Модуль формування порад створює рекомендації українською мовою. Модуль відтворення станів забезпечує створення минулих показників для віртуальних вузлів. Планувальник місткості оцінює напрям зміни показників використання процесора, пам'яті та диска [24]. Це дає змогу передбачити момент досягнення критичних порогів навантаження. Завдяки цьому адміністратор може вчасно запланувати розширення ресурсів до того, як продуктивність системи почне погіршуватися. Такий підхід дозволяє перейти від реактивної до проактивної діагностики. Механізм створення звітів формує підсумкові документи. Користувацька панель спостереження побудована на SvelteKit [15] із графіками ECharts [37] та виглядом українською мовою (рис. 2.1).

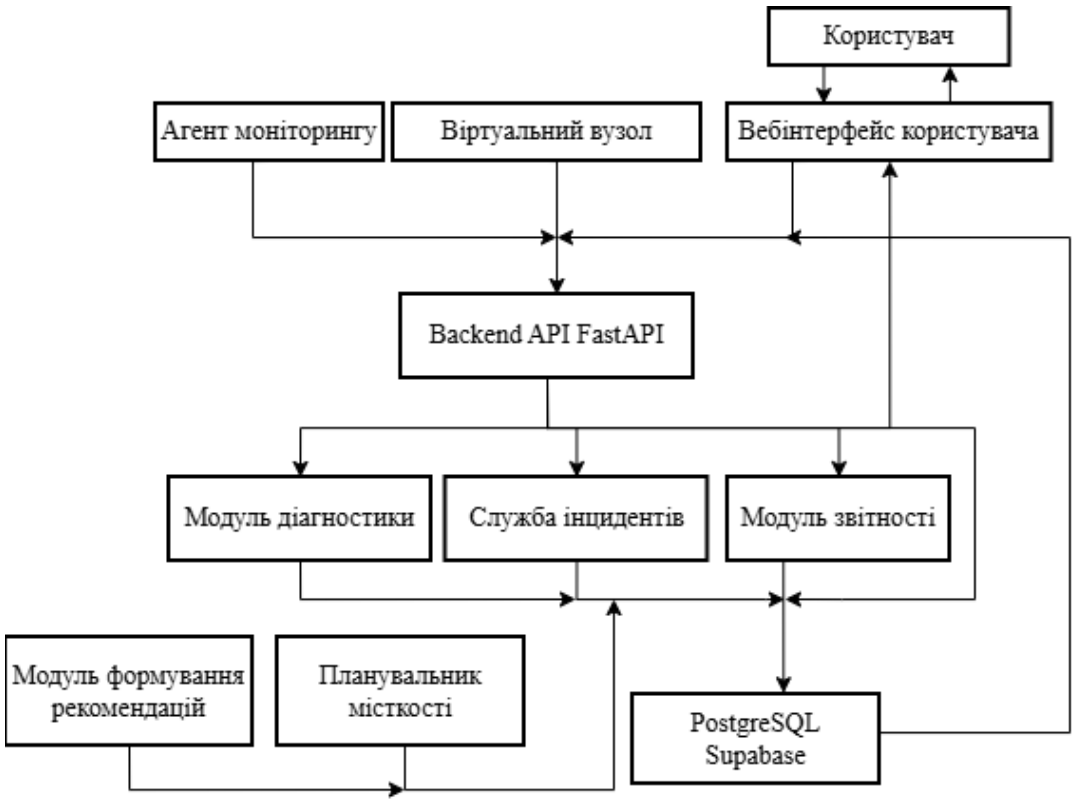


Рисунок 2.1 – Структурна схема компонентів системи діагностики та масштабування віртуалізованих ресурсів

Користувач створює вузол у панелі спостереження або реєструє вузол з агентом. Для реального вузла система генерує ключ доступу та команду запуску агента. Агент періодично надсилає сигнали та показники в серверну частину. Серверна частина перевіряє ключ доступу, зберігає дані спостереження в таблиці та оновлює статус вузла. Користувач або система запускає діагностику для вузла. Механізм діагностики аналізує останнє вікно показників, визначає першопричину, формує показник ризику [24, 25]. Якщо стан не є задовільним, створюється інцидент та рекомендація. Панель спостереження відображає вузли, графіки, інциденти, рекомендації та ризики.

У кінцевому варіанті архітектура має хмарне розгортання. Користувацький застосунок розміщується на Vercel [28] і є основною точкою входу для користувача. Серверна частина розгортається як Render Web Service [27]. Supabase PostgreSQL [12] виконує роль централізованої бази даних, у якій зберігаються середовища, вузли, показники спостереження, діагностика,

інциденти, рекомендації, прогнозування місткості та метадані звітів. Такий підхід відокремлює інтерфейс, серверну логіку та зберігання даних, а також забезпечує доступ до панелі спостереження через звичайне HTTPS-посилання.

Архітектура є вузлоцентричною. Це означає, що центральною сутністю є вузол, а всі інші дані пов'язані з ним: показники, діагностика, інциденти, рекомендації, звіти. Така модель спрощує перегляд стану конкретного ресурсу та дає змогу будувати сторінку вузла як основний операційний вигляд.

У системі передбачено сутність Environment. Воно групує вузли у логічне середовище. Також дає змогу аналізувати не лише окремий вузол, а й контекст групи. Це важливо для виявлення надмірного виділення ресурсів, коли сумарні виділені ресурси вузлів можуть перевищувати задані обмеження [1, 2].

Серверна частина виконує роль центрального координатора. Вона не лише приймає API-запити, а й викликає служби діагностики, рекомендацій, інцидентів, планування місткості та формування звітів [13]. Такий підхід дає змогу зберегти користувацьку частину тонкою: інтерфейс не містить складної бізнес-логіки, а лише відображає результати.

Важливою частиною архітектури є розмежування вузла з агентом та віртуального вузла. Вузол з агентом представляє реальний сервер або комп'ютер, де працює агент. Віртуальний вузол використовується для моделювання сценаріїв перевантаження процесора, нестачі пам'яті, вузького місця диска, нестачі мережевих ресурсів, неповного використання ресурсів та інших станів. Це дає змогу перевірити механізм діагностики без необхідності створювати реальні аварійні навантаження на обладнанні.

Для обміну даними між компонентами використовується REST API. Основні групи точок доступу: вузли, агенти, показники, діагностика, інциденти, рекомендації, відтворення станів, місткість, звіти та панель спостереження. У режимі розгортання користувацька частина на Vercel [28] звертається до серверної частини на Render [27] через змінну VITE_API_BASE_URL, а серверна частина отримує адресу Supabase PostgreSQL [12] через DATABASE_URL.

Такий підхід з API є простим для локального запуску, тестування через curl або Postman та інтеграції з користувацькою частиною.

Проєктування архітектури також враховує принцип розділення відповідальності. Агент відповідає лише за збір та передачу даних [21]. Серверна частина відповідає за приймання запитів, перевірку, авторизацію агентів та виклик служб. Служби відповідають за бізнес-логіку. База даних відповідає за зберігання даних [12]. Користувацька частина відповідає за показ інформації [15]. Такий розподіл дає змогу змінювати один компонент без повного переписування системи [3].

Архітектура підтримує два потоки даних: робочі дані спостереження та керування через інтерфейс. Робочі дані спостереження це регулярні сигнали справності та показники від агентів або механізму відтворення станів. Керування це дії користувача в панелі спостереження: створення вузла, запуск сценарію, запуск діагностики, перегляд рекомендацій, виведення даних. Розділення цих потоків є важливим, оскільки отримання даних спостереження має бути стабільним і простим, а для користувача може мати ширший функціонал.

У системі використано централізовану модель зберігання. Усі показники потрапляють у єдину базу даних, що спрощує побудову звітів і панелі спостереження. Альтернативою могла б бути розподілена модель, коли кожен агент зберігає локальну історію, а серверна частина лише періодично її забирає. Однак для кваліфікаційної роботи централізована модель є більш доречною, оскільки вона є прозорою та дає змогу швидко виконувати запити до історії вузла.

Окремо спроєктовано життєвий цикл вузла з агентом. Спочатку користувач створює вузол або реєструє агента. Серверна частина створює ключ доступу для агента та повертає команду встановлення. Після запуску агент надсилає сигнали справності та показники. Якщо сигнал справності успішний, вузол отримує статус «працює». Якщо діагностика виявляє високий ризик, статус може перейти у «попередження» або «критичний». Таким чином, статус вузла

формується не тільки фактом підключення агента, а й результатами аналізу даних спостереження.

Життєвий цикл віртуального вузла відрізняється. Такий вузол може не мати реального агента, а його показники генеруються механізмом відтворення станів. Це дає змогу створити контрольоване середовище для тестування механізму діагностики. Віртуальний вузол є особливо важливим для показу сценаріїв, які небажано або складно відтворювати на реальному обладнанні, наприклад загроза перегріву або надмірне виділення ресурсів [1, 2].

У межах архітектури передбачено також механізм звітів. Він отримує дані з бази і формує структурований звіт.

2.2 Обґрунтування вибору технологій та середовища реалізації

Вибір засобів розробки здійснено з урахуванням того, що система працює як мережевий програмний комплекс [30, 31]. Агент, серверна частина, база даних та вебінтерфейс можуть працювати на різних машинах або хмарних платформах. Ці компоненти об'єднані в єдину систему за допомогою комп'ютерної мережі, яка може бути як локальною, так і глобальною. Такий підхід накладає додаткові вимоги до протоколів обміну даними, безпеки передачі інформації, контролю цілісності пакетів та стійкості до розривів з'єднання (табл. 2.1).

Тому ключовими вимогами були підтримка HTTP API, простий формат обміну JSON, можливість хмарного розгортання, стабільна робота через мережу, підтримка віддаленої бази даних. У процесі функціонування системи агент може бути віддалений від серверної частини на значну відстань, працювати через NAT або в середовищі з нестабільним каналом зв'язку [36]. Тому важливо враховувати можливі втрати пакетів, затримки передачі даних, обмеження пропускної здатності.

Для серверної частини обрано Python і FastAPI [13, 43, 44]. Python має велику екосистему для системного спостереження, аналізу даних і компонентів

машинного навчання. FastAPI забезпечує швидку розробку REST API, автоматичну OpenAPI-документацію, перевірку через Pydantic [46] та зручну інтеграцію з асинхронними засобами та серверним оточенням. Це дає змогу швидко реалізувати програмний інтерфейс та зосередитися на логіці діагностики.

Для роботи з базою даних використано SQLAlchemy 2 [14]. Це дає можливість описати доменну модель через ORM-класи, підтримувати зв'язки між сутностями та працювати як із SQLite, так і з PostgreSQL [11]. Для міграцій використовується Alembic [23]. У локальному режимі система може працювати з SQLite, що спрощує запуск. У кінцевому варіанті використовується Supabase PostgreSQL [12], підключений до серверної частини на Render [27] через змінну DATABASE_URL.

Для збору системних показників агент використовує бібліотеку psutil [21]. Вона надає кросплатформений доступ до показників процесора, оперативної пам'яті, додаткової пам'яті, диска, мережі, процесів і часу безперервної роботи. Це дає змогу запускати агента на Linux, macOS або Windows без прив'язки до конкретного гіпервізора.

Для виявлення аномалій використано бібліотеку scikit-learn та алгоритм, що знаходить рідкісні та нехарактерні комбінації показників [25]. Він застосовується тоді, коли накопичено достатню кількість показників. Якщо історії недостатньо або модуль машинного навчання недоступний, система переходить до перевірки за правилами. Такий підхід забезпечує стабільність і не робить механізм діагностики залежним виключно від моделі машинного навчання.

Для локального тестування використано Docker Compose [5, 6]. У проєкті передбачено серверну службу та користувацьку службу. Серверна частина працює на порту 8000, користувацька частина на порту 3000 при запуску через Docker. У локальному середовищі всі компоненти системи можуть працювати в межах однієї віртуальної мережі Docker, що забезпечує їхню ізоляцію та спрощує

налаштування мережевих взаємодій між контейнерами. Такий підхід імітує роботу системи в умовах реальної комп'ютерної мережі, де кожен компонент має власну мережеву адресу та порти для обміну даними [30, 31].

Для кінцевого доступу через веб-посилання використано іншу модель розгортання: серверна частина розміщується на Render [27], користувацька частина на Vercel [28], база даних у Supabase [12].

Таблиця 2.1 – Обґрунтування вибору технологій

Компонент	Технологія	Причина вибору
Серверна частина	FastAPI	Швидка розробка REST API, Pydantic, OpenAPI
Мова серверної частини	Python 3.11	Система для спостереження, роботи з даними та машинного навчання, простота розробки
ORM	SQLAlchemy 2	Гнучка робота з SQLite та PostgreSQL
Міграції	Alembic	Контроль схеми бази даних
Локальна база даних	SQLite	Простий локальний запуск для розробки
Промислова база даних	Supabase PostgreSQL	Віддалене зберігання даних для публічного вебзастосунку
Показники агента	psutil	Кросплатформений збір системних показників
Виявлення аномалій	scikit-learn	Виявлення нетипових комбінацій показників
Користувацька частина	SvelteKit	Панель спостереження, компонентний підхід
Типізація	TypeScript	Зменшення кількості помилок при роботі

Кінець таблиці 2.1

Графіки	ECharts	Інтерактивне відображення даних спостереження
Звіти	ReportLab / HTML reports	Формування звітів
Розгортання	Render, Vercel, Supabase	Публічна серверна частина, користувацька частина за веб-посиланням і віддалена база даних
Локальне тестування	Docker Compose	Перевірка системи на локальній машині
Стилі	Tailwind CSS	Швидка побудова користувацького інтерфейсу

Середовище реалізації орієнтоване на локальну розробку та публічне розгортання. У кінцевому розгортанні користувацька частина працює на Vercel [28], серверна частина на Render [27], а агент надсилає дані спостереження вже не на localhost, а на публічну адресу серверної частини на Render.

Вибір FastAPI [13] також обґрунтований тим, що система активно використовує Pydantic-моделі [46]. Pydantic дає змогу описати схеми запитів і відповідей у коді, автоматично перевіряти типи, створювати JSON-схеми та зменшити кількість ручної перевірки. Для вхідних даних спостереження це особливо важливо, оскільки агент може надсилати неповні або помилкові дані. Якщо структура запиту не відповідає схемі, серверна частина може відхилити його ще до запису в базу даних.

SQLAlchemy [14] обрано як спосіб чітко описати доменну модель. Наприклад, вузол має зв'язки з показниками, діагностикою, інцидентами, рекомендаціями та прогнозами місткості. Такі зв'язки легко представити в ORM і використовувати у службах. Крім того, SQLAlchemy дозволяє працювати з різними серверними частинами SQL без суттєвої зміни бізнес-логіки.

SQLite використовується як локальна база даних через простоту. Для кваліфікаційної роботи це важливо, оскільки перевірити систему можна без встановлення окремого сервера бази даних. Водночас основним варіантом для кінцевої версії є Supabase PostgreSQL [12]. Supabase надає віддалену базу даних PostgreSQL, до якої підключається серверна частина на Render [27]. Це дає змогу зберігати дані спостереження, діагностику та рекомендації незалежно від локального комп'ютера розробника та забезпечує роботу вебінтерфейсу для користувача за посиланням.

Бібліотека psutil [21] є доречним вибором для агента, тому що бібліотека надає засіб доступу мовою Python для різних операційних систем. Без psutil довелося б використовувати різні системні команди або програмні входи, залежно від конкретної платформи. Це ускладнило б агента та зробило б його менш переносимим.

Вибір SvelteKit [15] для користувацької частини пояснюється тим, що панель спостереження є інтерактивною, але не потребує надмірно складного рішення для користувацької частини. Svelte має компактну модель складових частин, а SvelteKit забезпечує маршрутизацію та збирання проєкту. TypeScript допомагає узгодити типи користувацької частини з серверною частиною. Tailwind CSS [16] дає змогу швидко створювати таблиці, картки, позначки та зручне розташування елементів.

ECharts [37] використовується для графіків даних спостереження. Графічне подання є дуже важливим для панелі спостереження інфраструктури, оскільки часто оцінюється не лише числове значення, а й вигляд графіка: сплеск, плато, поступове зростання, коливання. ECharts дає можливість будувати лінійні графіки для показників процесора, оперативної пам'яті, диска, мережі та огляду ризиків без розроблення власної логіки побудови графіків.

2.3 Розроблення структури телеметричних даних та формату обміну повідомленнями

Дані спостереження є основою системи, що розробляється. Вони описують поточний стан вузла і використовуються для побудови графіків, діагностики, створення інцидентів, формування рекомендацій та планування місткості [24]. Формат вхідних даних спостереження повинен бути достатньо повним для аналізу, але водночас простим для надсилання через HTTP.

У системі використовуються два основні типи повідомлень від агента: повідомлення з сигналом справності та повідомлення з показниками [21]. Такий підхід дозволяє розділити контроль стану агента та передачу безпосередньо даних спостереження. Сигнал справності надсилається періодично для підтвердження, що агент працює та має зв'язок із серверною частиною. Повідомлення з показниками містить деталізовану інформацію про стан ресурсів вузла. Завдяки такому розділенню серверна частина може окремо відстежувати доступність агента і якість даних, що надходять. Це спрощує діагностику проблем у разі втрати зв'язку або пошкодження даних спостереження.

Повідомлення з сигналом справності повідомляє серверну частину, що агент є активним, а також передає назву вузла, назву операційної системи, її версію та версію агента. Повідомлення з показниками містить числові дані спостереження [21]. Обидва повідомлення містять ідентифікатор вузла та ключ доступу, які використовуються для розпізнавання вузла та підтвердження справжності агента (рис. 2.2).Такий підхід забезпечує базовий рівень безпеки при передачі даних. Крім того, використання ключа доступу дозволяє серверній частині однозначно ідентифікувати джерело даних без додаткової автентифікації.

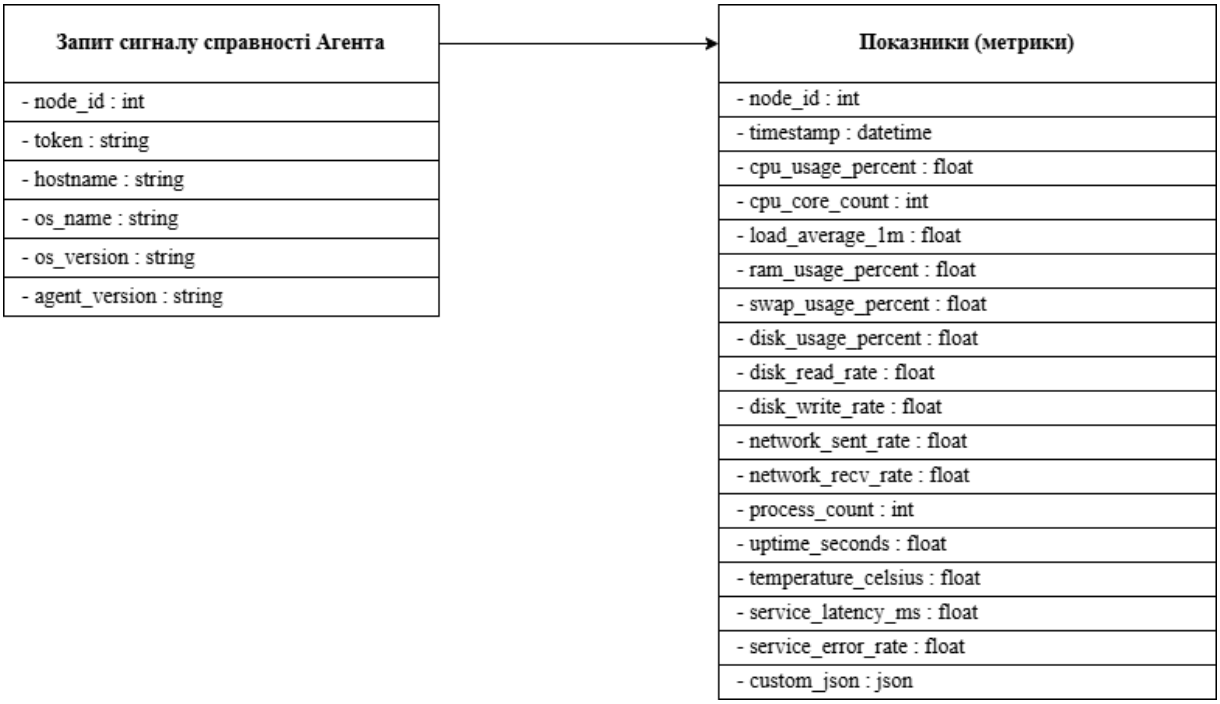


Рисунок 2.2 – Схема структури вхідних даних спостереження

Поле з custom_json дає змогу додавати специфічні дані без негайної зміни схеми. Наприклад, агент додає версію агента та назву вузла. Механізм відтворення станів додає позначку про те, що дані згенеровано штучно, та назву сценарію. Це робить формат таким, що легко розширюється.

Важливо, що вхідні дані спостереження передаються у форматі JSON [17]. Це спрощує поєднання з REST API, користувацькою частиною, програмою curl, Postman та іншими. Серверна частина використовує Pydantic-схеми [46] для перевірки вхідних даних. Якщо обов'язкові поля відсутні або ключ доступу некоректний, запит відхиляється.

У загальному вигляді повідомлення з показниками містить поточні значення ресурсів і може надсилатися кожні 5 секунд. Серверна частина зберігає кожну точку як окремий запис у таблиці показників ресурсів. Це дає змогу будувати графіки зміни показників у часі та аналізувати останні кілька значень [37].

Для механізму діагностики використовується вікно останніх 30 показників. Це є компромісом між своєчасністю та стійкістю. Останнє значення

показує поточний стан, а вікно дає змогу оцінити середнє використання процесора та пам'яті, тривале неповне використання ресурсів та виявлення аномалій [25]. Для планувальника місткості використовується до 80 останніх точок, що дає змогу оцінити напрям зміни показників [24].

При розробленні формату даних спостереження важливо було забезпечити стабільність формату. Агент може працювати довго, тому зміни в серверній частині не повинні часто порушувати сумісність. Для цього більшість полів зроблено необов'язковими. Якщо певна операційна система не підтримує давачі температури або середнє значення черги завдань [26], агент може передати відсутнє значення або не передавати значення зовсім, а серверна частина все одно збереже інші показники.

Ще одним рішенням є використання однакових одиниць вимірювання. Оперативна пам'ять передається в мегабайтах, обсяг диска у гігабайтах, швидкість передачі у байтах за секунду, використання процесора, пам'яті, додаткової пам'яті та диска у відсотках. Це важливо для правильної інтерпретації даних і для панелі спостереження. Невизначені або змішані одиниці могли б призвести до помилкових рекомендацій.

Дані спостереження також містять сумарні лічильники: кількість прочитаних і записаних байтів на диску, обсяг відправлених і отриманих байтів у мережі [21]. Вони корисні для історії, але механізм діагностики переважно використовує поля швидкості передачі, оскільки вони краще показують поточну інтенсивність. Агент обчислює швидкість на основі різниці між поточним і попереднім значеннями лічильників та часу, що минув між вимірюваннями.

У схемі передбачено затримку відповіді служби в мілісекундах та частоту помилок служби. Ці поля можуть не збиратися базовим агентом на psutil [21], але вони важливі для майбутнього розширення. Наприклад, застосунок може передавати власні службові показники через додаткове поєднання. Якщо показники ресурсів є нормальними, але затримка або частота помилок є високими, механізм діагностики може визначити стан як погіршення роботи

служби та не рекомендувати збільшення процесора або пам'яті без додаткових підтверджень [22].

Структура даних спостереження також підтримує перевірку подій. Кожна точка має позначку часу, а діагностика та рекомендації мають час створення. Це дає змогу відновити послідовність подій: коли показники почали погіршуватися, коли була запущена діагностика, коли створено інцидент і коли сформовано рекомендацію [24]. Для зберігання доказів обрано поле у форматі JSON [17]. Альтернативою було б створення окремої таблиці для доказів діагностики, але JSON є простішим і достатньо гнучким. Докази можуть містити різні набори полів залежно від типу діагностики, тому жорстка реляційна схема була б занадто громіздкою (таб. 2.2).

Таблиця 2.2 – Логічний склад даних спостереження з погляду механізму діагностики

Група	Поля	Діагностичне призначення
Процесор	cpu_usage_percent, cpu_core_count, load_average	Виявлення навантаження процесора та утворення черги завдань
Оперативна пам'ять	ram_usage_percent, ram_available_mb, ram_total_mb	Виявлення нестачі пам'яті
Додаткова пам'ять	swap_usage_percent, swap_used_mb	Виявлення тиску на файл підкачування
Диск	disk_usage_percent, disk_read_rate, disk_write_rate	Виявлення заповнення диска та вузького місця обміну даними

Кінець таблиці 2.2

Мережа	network_sent_rate, network_recv_rate	Виявлення нестачі мережевих ресурсів
Система	process_count, uptime, temperature	Виявлення загрози перегріву та системний контекст
Служба	latency, error_rate	Виявлення погіршення роботи служби
Додаткові дані	custom_json	Розширення без зміни схеми

2.4 Розроблення алгоритму діагностики та формування рекомендацій

Алгоритм діагностики в системі, що розробляється, реалізовано в модулі механізму діагностики. Він отримує вузол та останні показники спостереження, після чого послідовно перевіряє набір умов для визначення першопричини [24, 25]. Результатом є запис діагностики з полями, що містять тип діагностики, першопричину, ступінь важливості, впевненість у висновку, показник ризику, пояснення, докази та стан (рис. 2.3).

Алгоритм побудовано як пріоритетну послідовність перевірок. Спочатку перевіряється надмірне виділення ресурсів [1, 2], оскільки він пов'язаний не лише з поточними показниками, а й з моделлю виділення та обмежень вузла або середовища. Далі перевіряється загроза перегріву, стани пам'яті та додаткової пам'яті [26], навантаження процесора [26], вузьке місце обміну даними з диском, нестача мережевих ресурсів [30, 31], неповне використання ресурсів, погіршення роботи служби та стан аномалій [25].

Для кожного типу діагностики визначено власну логіку збору доказів. Наприклад, для навантаження процесора докази містять відсоток використання процесора, поріг вище 85 % та середнє значення черги завдань за останню

хвилину як додатковий доказ [26]. Для нестачі пам'яті перевіряється не лише відсоток використання оперативної пам'яті вище 90 %, а й те, що доступний обсяг оперативної пам'яті становить менше 12 % від загального обсягу або менше 512 мегабайтів. Для тиску на файл підкачування потрібне одночасне високе використання оперативної пам'яті та додаткової пам'яті. Така логіка зменшує кількість помилкових висновків у роботі програми.

Показник ризику є числовим значенням від 0 до 100. Він використовується для визначення ступеня важливості та статусу вузла. Наприклад, високі значення ризику відповідають критичному стану, середні відповідають попередженню а низькі здоровому або інформаційному стану. Впевненість також задається у відсотках і показує, наскільки впевнено система визначає стан [24]. Ризик обчислюється на основі відхилення поточних показників від порогових значень. Чим більше відхилення, тим вищий ризик для працездатності вузла. Впевненість формується з урахуванням кількості доступних доказів та історії спостережень. Якщо доказів недостатньо, впевненість знижується, навіть якщо ризик є високим. Це дозволяє уникнути хибних спрацьовувань на обмежених даних. Користувач бачить обидва показники разом, що допомагає оцінити, наскільки критичною є ситуація і чи варто довіряти висновку системи. Такий підхід підвищує прозорість прийняття рішень та зменшує ймовірність помилкових дій з боку адміністратора. Завдяки цьому користувач може свідомо обрати, чи варто виконувати рекомендовану дію. Система не приймає рішення за людину, а лише надає обґрунтовану пропозицію. Це особливо важливо у випадках, коли зміна ресурсів може мати фінансові наслідки або вплинути на роботу інших сервісів. Такий підхід відповідає сучасним практикам побудови систем підтримки прийняття рішень, де остаточний вибір залишається за оператором.

Крім того, наявність доказів та пояснень дозволяє користувачу навчатися на основі рекомендацій системи. З часом це підвищує його власну компетентність у виявленні та усуненні подібних проблем без покладання виключно на автоматизовані інструменти.

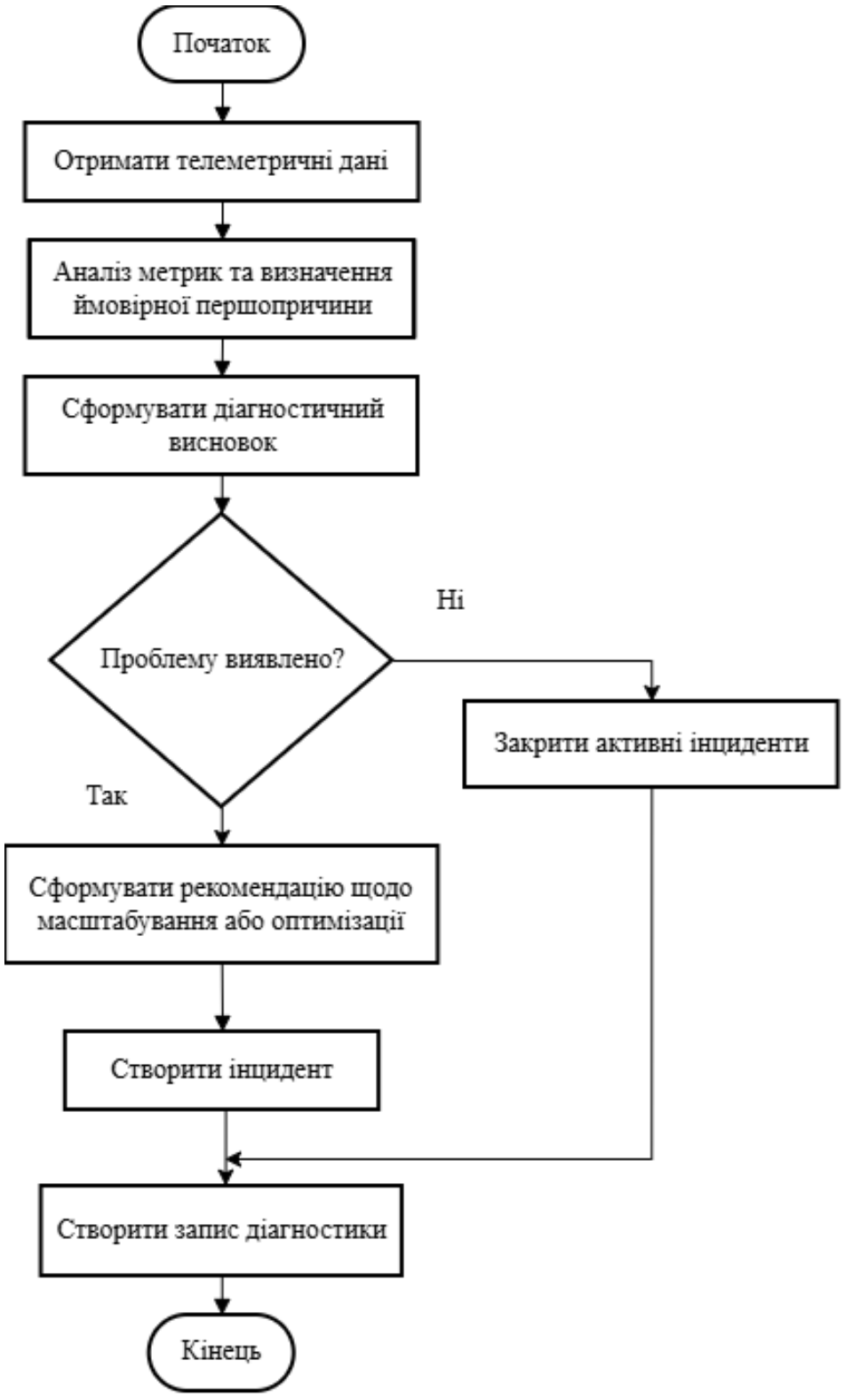


Рисунок 2.3 – Блок схема алгоритму діагностики стану вузла та формування рекомендацій

Механізм формування рекомендацій отримує запис діагностики та формує рекомендацію залежно від типу діагностики [22]. Для навантаження процесора

система рекомендує збільшення процесорних потужностей по вертикалі або перевірку можливості розширення по горизонталі для окремих примірників застосунку [18]. Для нестачі пам'яті та тиску на файл підкачування рекомендується збільшення оперативної пам'яті та перевірка витоків пам'яті [26]. Для вузького місця обміну даними з диском система не рекомендує збільшення процесора, а пропонує оптимізувати черги диска, SQL-запити [14], рівень зберігання даних або кешування. Для нестачі мережевих ресурсів пропонується перевірити пропускну здатність, розподілювач навантаження та обсяг переданих даних між службами [30, 31]. Для неповного використання ресурсів формується рекомендація щодо зменшення потужностей або ущільнення (табл. 2.3).

Таблиця 2.3 – Відповідність діагнозів і рекомендацій

Тип діагностики	Тип рекомендації	Основна дія
Навантаження процесора	Вертикальне збільшення процесора	Збільшити кількість ядер процесора або розширити кількість примірників
Нестача пам'яті	Вертикальне збільшення пам'яті	Збільшити обсяг оперативної пам'яті, перевірити витоки пам'яті
Тиск на файл підкачування	Вертикальне збільшення пам'яті	Зменшити залежність від файлу підкачування, збільшити обсяг оперативної пам'яті
Вузьке місце диска	Оптимізація сховища	Оптимізувати сховище даних, затримки диска, кешування
Нестача мережевих ресурсів	Оптимізація мережі	Перевірити пропускну здатність і розподіл мережевого трафіку

Кінець таблиці 2.3

Неповне використання ресурсів	Зменшення потужностей	Зменшити виділення ресурсів або об'єднати робочі навантаження
Надмірне виділення ресурсів	Оптимізація виділення	Переглянути квоти, коефіцієнт надмірного виділення або ресурси основного вузла
Невідоме погіршення	Масштабування не рекомендується	Зібрати більше даних і перевірити журнали подій

Рекомендації містять назву, опис, пріоритет, поточну кількість ядер процесора, рекомендовану кількість ядер процесора, поточний обсяг оперативної пам'яті, рекомендований обсяг оперативної пам'яті, очікуваний вплив, причину, покрокові дії у форматі та стан. Це робить рекомендацію не просто текстовим повідомленням, а структурованим об'єктом, який можна показувати в панелі спостереження, фільтрувати, приймати, ігнорувати або закривати [22].

Порядок перевірок у механізмі діагностики має практичне значення [24]. Наприклад, якщо одночасно виявлено використання оперативної пам'яті більше 85 % і використання додаткової пам'яті більше 20 %, система визначає стан як тиск на файл підкачування, а не просто як нестачу пам'яті [26]. Це точніше описує реальний ризик, оскільки активне використання файлу підкачування зазвичай має більший вплив на затримки. Якщо активність диска є високою, але використання процесора також перевищує 85 %, система спочатку визначає перевантаження процесора, оскільки перевантаження процесора може впливати на інші підсистеми. Така пріоритезація є частиною експертної логіки.

Надмірне виділення ресурсів перевіряється до перевірки показників ресурсів [1, 2], оскільки це структурна проблема виділення. Вона може існувати навіть до того, як вузол почне показувати високе використання під час роботи.

Наприклад, віртуальна машина може мати виділену кількість ядер процесора більше за максимальну кількість ядер, або середовище може мати сумарне виділення ресурсів вище за сумарне обмеження. У такому разі рекомендація має стосуватися квот, перенесення віртуальної машини або додавання ресурсів основного вузла.

Загроза перегріву також має високий пріоритет. Якщо температура перевищує безпечний поріг, збільшення процесора або оперативної пам'яті не вирішить проблему. Навпаки, додаткове навантаження може погіршити ситуацію. Тому механізм формування рекомендацій для загрози перегріву пропонує перевірити охолодження, зменшити навантаження та переглянути можливості пристрою.

Для неповного використання ресурсів використовується тривалий проміжок спостереження. Це потрібно, щоб не рекомендувати зменшення потужностей через одну випадково низьку точку. Якщо останні значення використання процесора та пам'яті є стабільно низькими, система робить висновок про надлишкове виділення ресурсів [24]. Така рекомендація є важливою для оптимізації витрат і підвищення ефективності використання кластера.

У механізмі формування рекомендацій передбачено обмеження на максимальну кількість ядер процесора та максимальний обсяг оперативної пам'яті. Якщо вузол має заданий максимум, рекомендоване збільшення не повинно його перевищувати. Це робить рекомендації більш реалістичними. Наприклад, якщо вузол має 4 ядра процесора та максимальну кількість ядер 8, система може рекомендувати 8, але не 16 [1, 2].

Покрокові дії сформульовані як практичний план. Вони не обмежуються одним реченням «збільшити ресурс». Для навантаження процесора система також пропонує перевірити процеси, які найбільше використовують процесор, розглянути горизонтальне масштабування [18] та перевірити розподіл навантаження. Для нестачі пам'яті пропонується перевірити витоки пам'яті та

процеси з великим обсягом використання пам'яті [26]. Для вузького місця диска пропонується перевірити черги диска, SQL-запити [14], рівень зберігання даних та кешування. Це підвищує практичну цінність рекомендацій.

Якщо користувач бачить лише назву рекомендації, він може сумніватися у правильності висновку. Якщо ж разом із рекомендацією доступні докази, порогові значення та поточні показники, рішення стає більш прозорим [24]. У панелі спостереження це може бути відображено як перелік доказів для кожної діагностики.

Алгоритм також враховує ситуації, коли основний спосіб дії недоступний. Якщо показників немає, система не робить жорстких висновків, а повертає невідоме помилку з поясненням про недостатність даних. Якщо виявлення аномалій показало нетипову поведінку, але конкретне вузьке місце не підтверджено, система також визначає стан як невідоме помилка [25]. Це краще, ніж формувати необґрунтовану рекомендацію. У такому випадку система пропонує користувачу почекати щоб зібрати додаткові дані. Така поведінка знижує ризик прийняття неправильних рішень на основі неповної інформації. Це стимулює користувача до глибшого вивчення проблеми замість автоматичного збільшення ресурсів.

2.5 Модель роботи системи та оптимізація обробки телеметричних даних

Модель роботи системи побудована навколо кола отримання даних спостереження. Для реального вузла агент у нескінченному циклі виконує такі дії: формує повідомлення з сигналом справності, надсилає його на серверну частину, збирає системні показники через бібліотеку psutil [21], обчислює показники швидкості для диска та мережі на основі попереднього вимірювання, формує повідомлення з показниками, надсилає його на серверну частину, очікує задану кількість секунд і повторює цикл.

Серверна частина під час отримання сигналу справності перевіряє ключ доступу, оновлює статус вузла, назву вузла, дані про операційну систему, версію агента та час останнього сигналу. Під час отримання показників серверна частина також перевіряє ключ доступу та зберігає запис показників ресурсів. Важливо, що точки отримання даних відокремлені від точок доступу панелі спостереження. Це дає змогу зробити агента простим і стабільним, а користувацький інтерфейс незалежним [13].

Для віртуальних вузлів дані спостереження створюються механізмом відтворення станів. Він створює послідовність показників, які імітують певний сценарій. Наприклад, навантаження процесора поступово підвищує використання процесора до 90-98 %, збільшує затримку та частоту помилок [21]. Нестача пам'яті підвищує використання оперативної пам'яті та додаткової пам'яті [26]. Вузьке місце диска збільшує швидкість читання та запису диска при відносно нормальному використанні процесора. Нестача мережевих ресурсів підвищує швидкість відправлення та отримання даних [30, 31]. Неповне використання ресурсів створює низькі значення використання процесора та пам'яті.

Оптимізація обробки даних спостереження в роботі здійснюється такими рішеннями. Механізм діагностики аналізує лише останні 30 показників, а не всю історію [24]. Планувальник місткості аналізує до 80 останніх показників. Точки доступу панелі спостереження повертають обмежені переліки останніх записів. Запити до бази даних використовують ідентифікатор вузла як індексоване поле [12]. Старі прогнози місткості для вузла видаляються перед записом нових. Докази зберігаються у полі JSON [17], що дає змогу уникнути надмірного розділення даних на окремі таблиці. SQLite використовується для локального режиму, а Supabase PostgreSQL [12] для кінцевого розгортання.

У базі даних основними таблицями є середовища, вузли, ключі доступу агентів, показники ресурсів, діагностика, інциденти, рекомендації, сценарії відтворення станів, прогнози місткості та звіти. Зв'язки між таблицями

побудовано так, щоб видалення вузла могло каскадно видалити пов'язані показники, діагностику, інциденти, рекомендації та прогнози місткості. Це спрощує підтримання цілісності даних у локальному середовищі [14].

Планувальник місткості використовує просту модель напряму зміни показників. Для кожного показника з набору використання процесора, використання оперативної пам'яті та використання диска обчислюється нахил між першим та останнім значенням у вікні [24]. Якщо напрям зростає, система прогнозує можливий час досягнення порогового значення. Якщо поріг вже перевищено, рекомендація в прогнозі вказує на необхідність діагностики та плану розширення потужностей. Якщо напрям спадає або є стабільним, система рекомендує продовжувати збирання показників.

Такий підхід не претендує на складне прогнозування промислового рівня, але є зрозумілим, пояснюваним і достатнім для демонстрації в межах кваліфікаційної роботи. Він показує, як історія даних спостереження може використовуватися не лише для реактивної діагностики, а й для попереднього планування місткості [13, 24].

Для оптимізації запитів важливо, що основні вибірки виконуються за ідентифікатором вузла та позначкою часу. Показники природно групуються за вузлом, а панель спостереження зазвичай показує останні значення для конкретного вузла. Тому ідентифікатор вузла є ключовим фільтром [12].

Модель даних також підтримує зв'язок між діагностикою, інцидентом та рекомендацією. Діагностика є результатом аналізу [24]. Інцидент є операційною подією, яку потрібно відстежувати. Рекомендація є пропозицією дії [22]. Розділення цих сутностей є важливим: одна діагностика може мати інцидент і рекомендацію, але їхні життєві кола відрізняються. Інцидент може бути відкритим або закритим, рекомендація може бути новою, прийнятою, проігнорованою або виконаною.

Механізм відтворення станів оптимізує перевірку системи тим, що не виконує реального навантаження. Він створює показники без навантаження на

процесор або диск на основному комп'ютері. Це є безпечнішим для локального комп'ютера та дає змогу відтворювати однакові сценарії. У коді використовується певне початкове значення випадковості, що робить відтворення стабільним.

Під час формування демонстраційного середовища система одразу створює кілька вузлів із різними станами. Це допомагає перевірити панель спостереження без ручного наповнення бази даних. Веб-вузол демонструє нормальний стан, вузол навантаження процесора, вузол бази даних нестачу пам'яті, робочий вузол неповне використання ресурсів. Такий набір охоплює кілька типових робочих станів [24].

Виведення даних у формати CSV та JSON реалізує окрему потребу: можливість аналізувати дані поза панеллю спостереження. CSV можна відкрити в табличному редакторі або використати для побудови графіків. JSON [17] є зручним для поєднання з іншими системами або автоматизованого випробування.

З погляду продуктивності, головним джерелом зростання даних є таблиця показників ресурсів. Якщо агент надсилає показники кожні 5 секунд, один вузол створює 720 записів за годину та 17280 записів за добу [21]. Для невеликого дослідного зразка це є прийнятним. Для десятків або сотень вузлів потрібно впроваджувати правила зберігання, узагальнення або зменшення кількості точок. Це можна позначити як напрям для майбутнього розвитку програми.

2.6 Забезпечення надійності та стабільності роботи системи

Надійність системи забезпечується на кількох рівнях: агентському, рівні програмного входу, рівні зберігання даних і рівні бізнес-логіки.

На агентському рівні реалізовано логіку повторних спроб. Функція надсилання даних виконує до трьох спроб надсилання HTTP-запиту. У разі помилки застосовується експоненційна затримка з очікуванням до 20 секунд.

Якщо всі спроби є неуспішними, агент не завершує роботу, а переходить до наступного кола. Це важливо, оскільки короткочасна недоступність серверної частини не повинна зупиняти спостереження назавжди [21] (рис. 2.4).

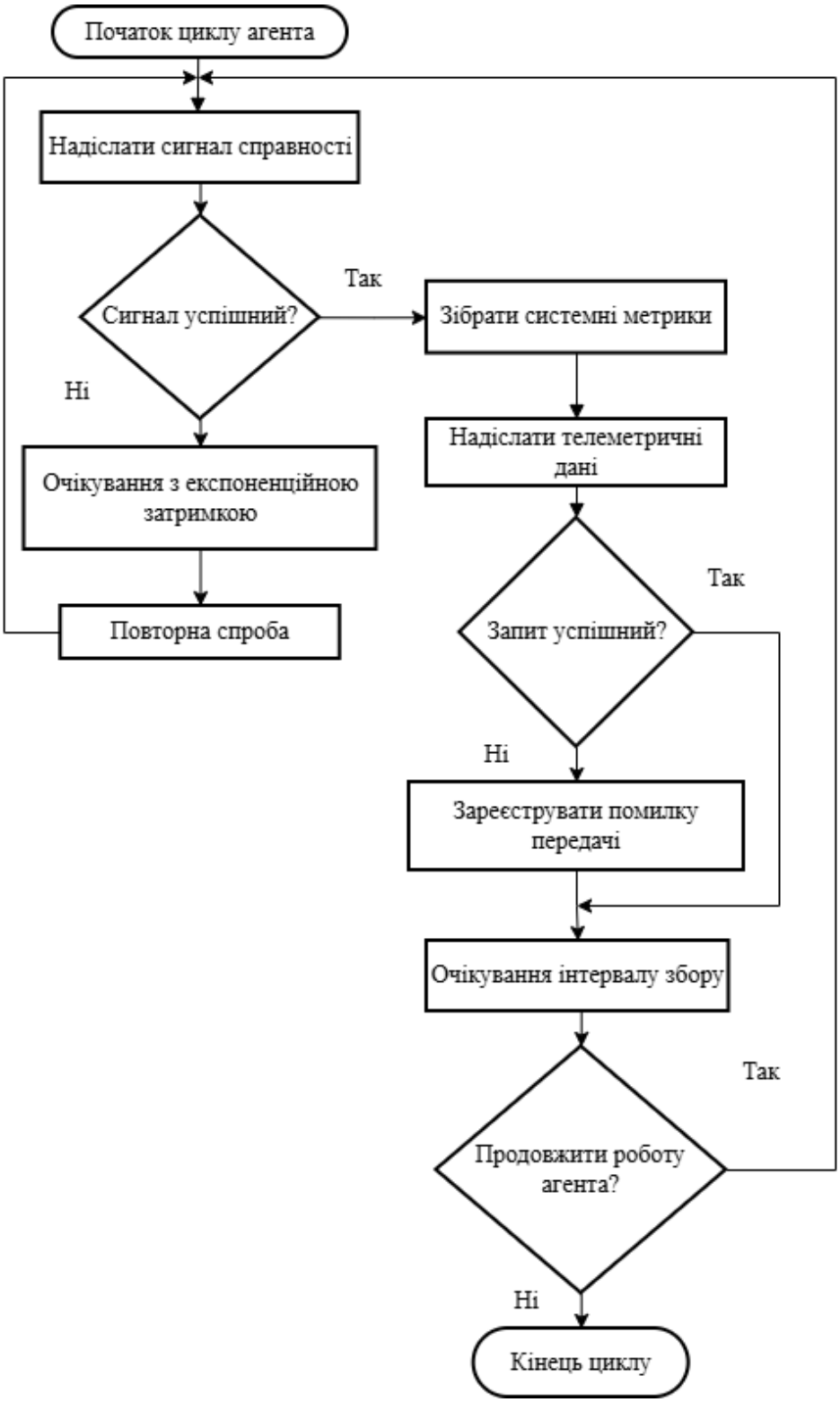


Рисунок 2.4 – Блок-схема роботи агента збору телеметричних даних

На рівні програмного входу використовується перевірка справжності за допомогою ключів доступу для точок входу агента. Ключ доступу не зберігається у відкритому вигляді в базі даних. Зберігається хеш ключа та його короткий зразок для попереднього перегляду. При кожному запиті сигналу справності або показників серверна частина обчислює хеш отриманого ключа та порівнює його з наявним записом [12]. Це зменшує ризик підробки агента через витік бази даних.

На рівні бізнес-логіки система не виконує небезпечних операцій на вузлах. Агент не має функцій запуску системних команд, зупинення процесів або зміни ресурсів. Механізм формування рекомендацій лише створює поради, а остаточне рішення ухвалює користувач. Це є принципово важливим для безпеки кваліфікаційного рішення [18, 22].

Стабільність серверної частини забезпечується розділенням модулів. Точки доступу лише приймають запити та викликають служби. Служби містять основну логіку. Моделі описують базу даних, схеми визначають перевірку вхідних даних [13, 14]. Такий поділ спрощує супровід і зменшує ризик помилок при зміні окремої частини системи [3].

Для локального запуску система використовує заповнення бази даних демонстраційними даними, якщо вона є порожньою. Якщо база даних порожня, серверна частина створює демонстраційне середовище з кількома віртуальними вузлами та початковою історією показників. Це підвищує зручність перевірки, оскільки користувач одразу бачить дані в панелі спостереження.

Для розгортання на Render, Vercel і Supabase передбачено змінні середовища: адреса підключення до бази даних, назва середовища, дозволені джерела для CORS, префікс API, назва застосунку, публічна адреса серверної частини та базова адреса API для користувацької частини [12, 27, 28]. Змінна з адресою бази даних у Render містить рядок підключення до Supabase PostgreSQL. Змінна з дозволеними джерелами містить адресу користувацької частини на

Vercel. Публічна адреса серверної частини містить відкриту адресу серверної частини на Render і використовується для формування команди запуску агента. Змінна з базовою адресою у Vercel вказує на серверну частину на Render із префіксом /api. Це дає змогу адаптувати систему до локального, Docker або хмарного середовища без зміни коду.

Стабільність системи також забезпечується тим, що механізм діагностики не блокує отримання даних. Приймання показників і запуск діагностики є окремими операціями. Це означає, що агент може продовжувати надсилати дані навіть тоді, коли користувач не запускає діагностику. У майбутньому можна додати фоновий планувальник, який періодично запускатиме діагностику для всіх вузлів, що працюють.

Перевірка за допомогою ключів доступу є мінімально необхідним механізмом захисту точок отримання даних. Без ключа будь-який клієнт міг би надсилати фальшиві показники для будь-якого ідентифікатора вузла. У поточному втіленні ключ прив'язаний до вузла і зберігається у вигляді хеша. Це не є повною промисловою моделлю безпеки, але є достатнім для кваліфікаційної системи та демонструє правильний напрям [12].

Налаштування CORS є важливим для роботи користувацької та серверної частин на різних доменах. У кінцевому розгортанні користувацька частина працює на домені Vercel [28], а серверна частина на домені Render [27], тому серверна частина на Render повинна мати у дозволених джерелах адресу користувацької частини на Vercel.

Відмовостійкість агента має обмеження. Якщо серверна частина є недоступною тривалий час, агент не накопичує всі показники локально, а лише продовжує спроби надсилання. Для промислового рівня можна було б додати локальну чергу або буфер попереднього запису. У кваліфікаційній роботі обрано простішу модель, оскільки вона зменшує складність агента та відповідає навчальним цілям [21].

Надійність даних також залежить від бази даних. SQLite добре підходить для локального запуску, але має обмеження щодо одночасного запису. Supabase PostgreSQL [12] краще підходить для розгортання, де одночасно працюють користувацька частина, агенти та серверні служби. Тому підтримка змінної адреси бази даних є важливою архітектурною перевагою: один і той самий код серверної частини може працювати локально з SQLite або в хмарному середовищі з Supabase.

Для забезпечення стабільного користувацького досвіду панель спостереження має правильно відображати стани відсутності даних. Якщо вузол ще не має показників, інтерфейс повинен показати, що дані очікуються, а не переставати працювати. Якщо діагностика відсутня, користувач має мати можливість запустити діагностику. Якщо рекомендації відсутні, це може означати здоровий стан або недостатню історію [24].

Важливим аспектом надійності є також стійкість до помилок у вхідних даних. Якщо агент надсилає показники з пропущеними полями або некоректними значеннями, серверна частина відхиляє лише проблемний запис, але продовжує приймати наступні. Це запобігає накопиченню помилок у базі даних та зберігає працездатність системи загалом. Крім того, агент має власний механізм обмеження частоти надсилання даних, що захищає серверну частину від надмірного навантаження у разі збоїв. Для запобігання втраті даних при раптовому завершенні роботи агента передбачено збереження останніх значень перед виходом. Серверна частина, у свою чергу, використовує з'єднаня з базою даних, що дозволяє ефективно обробляти одночасні запити без вичерпання ресурсів. Усі критичні операції діагностики виконуються в окремих транзакціях, що гарантує цілісність даних навіть при часткових збоях. Такий комплексний підхід до забезпечення надійності робить систему придатною для використання не лише в навчальних, але й у невеликих промислових середовищах.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ПЕРЕВІРКА РОБОТИ СИСТЕМИ ДІАГНОСТИКИ

3.1 Розроблення агентської підсистеми збору телеметрії

Агентська підсистема реалізована як окрема програма agent.py мовою Python. Вона призначена для запуску на реальному комп'ютері або сервері, який потрібно підключити до системи. Агент збирає показники за допомогою бібліотеки psutil і надсилає їх у серверну частину через HTTP POST-запити.

Агент складається з класу збирача показників та допоміжних функцій для надсилання даних у форматі JSON, завантаження налаштувань, розбору аргументів командного рядка та головної функції. Збирач показників зберігає попередні значення показників диска та мережі, щоб обчислювати швидкість читання та запису, а також швидкість мережевого трафіку. Це важливо, оскільки сумарні лічильники в байтах не завжди є зручними для діагностики, а показники швидкості краще відображають поточне навантаження (рис. 3.1).

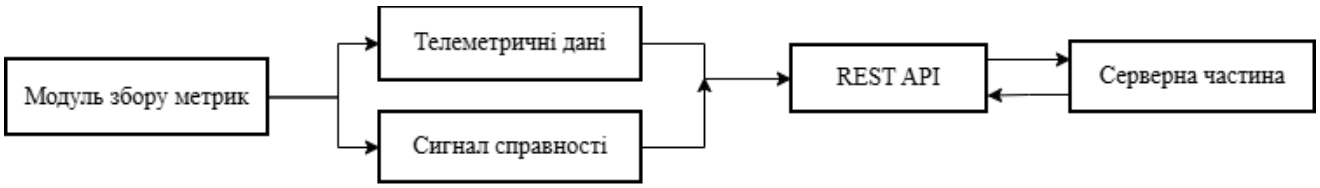


Рисунок. 3.1 – Структурна схема агентської підсистеми збору телеметрії

Для процесора агент використовує функцію отримання відсотка використання. Для оперативної пам'яті функцію отримання даних про віртуальну пам'ять. Для додаткової пам'яті функцію отримання даних про файл підкачування. Для диска функції отримання даних про використання диска та лічильники операцій введення та виведення. Для мережі функцію отримання лічильників мережевого обміну. Також збираються кількість процесів, час безперервної роботи в секундах, середнє значення черги завдань та температура, якщо така інформація є доступною на платформі.

Агент не виконує жодних змін на вузлі. Він не має функцій перезапуску, зупинення процесів, встановлення програмних пакетів або зміни ресурсів. Це робить його безпечним для демонстраційного використання. У разі помилок мережі агент виконує повторні спроби та продовжує роботу.

Особливістю втілення агента є те, що він не потребує складної інсталяції. Достатньо встановити залежності Python і запустити команду з ключем доступу. Агент може запускатися у звичайному середовищі терміналу, а в майбутньому його можна оформити як системну службу або службу Windows.

Функція розбору аргументів дозволяє передавати параметри через командний рядок або конфігураційний JSON-файл. Це зручно для різних випадків використання. Під час демонстрації простіше запускати агента з параметрами командного рядка. Для тривалого запуску зручнішим є конфігураційний файл, де можна зберігати адресу серверної частини, ідентифікатор вузла, інтервал і рівень ведення журналу подій.

Сигнал справності та показники надсилаються окремими запитами. Це має практичну перевагу: навіть якщо збір певних показників є частково неуспішним, сигнал справності може оновити статус роботи вузла. Крім того, серверна частина може в майбутньому використовувати сигнал справності для виявлення вузлів, які не працюють, незалежно від повноти вхідних даних спостереження.

При обчисленні показників швидкості агент використовує попередні значення лічильників операцій диска та лічильників мережевого обміну. На першій ітерації швидкість дорівнює нулю, оскільки немає попередньої точки для порівняння. На наступних ітераціях швидкість обчислюється як різниця байтів, поділена на час, що минув між вимірюваннями. Такий підхід є типовим для лічильників, які постійно зростають від моменту запуску системи.

Для ведення журналу подій використовується стандартний засіб ведення журналу. Агент повідомляє про запуск, успішне надсилання показників або помилки доставки. Це дає змогу під час випробування швидко зрозуміти, чи є

проблема з мережею, адресою серверної частини, ключем доступу або ідентифікатором вузла.

3.2 Розроблення серверної частини для приймання та обробки телеметричних даних

Серверна частина реалізована на FastAPI. Основний файл застосунку ініціалізує базу даних, виконує заповнення демонстраційного середовища, налаштовує CORS і підключає маршрутизатори API. Серверна частина має префікс /api, тому точка перевірки справності доступна як /api/health.

Архітектура серверної частини побудована за модульним принципом. Головний файл відповідає за створення FastAPI-застосунку. Файл налаштувань містить параметри конфігурації. Файл підключення до бази даних забезпечує роботу з базою. Файли моделей містять ORM-моделі SQLAlchemy. Файли схем містять Pydantic-схеми. Окрема тека містить маршрути REST, інша тека служби з бізнес-логікою, ще одна допоміжні функції (рис. 3.2). Така організація коду відповідає принципу розділення відповідальності. Маршрути лише приймають запити та викликають відповідні служби, не містять бізнес-логіки. Служби реалізують основну логіку діагностики, формування рекомендацій та роботи з даними. Допоміжні функції забезпечують повторне використання коду, наприклад нормалізацію даних або допоміжні розрахунки. Завдяки такому підходу зміна в одному модулі не потребує переписування інших. Це спрощує тестування окремих компонентів системи. Додавання нових функцій у такій архітектурі потребує змін лише в одному модулі, що зменшує ризик помилок. За потреби можна легко замінити одну технологію на іншу, не переписуючи весь застосунок. Також ізольованість модулів підвищує безпеку системи. Навіть якщо один компонент вийде з ладу або буде скомпрометований, інші продовжать працювати в штатному режимі. Це досягається завдяки тому, що кожен модуль має чітко визначені входи та виходи.

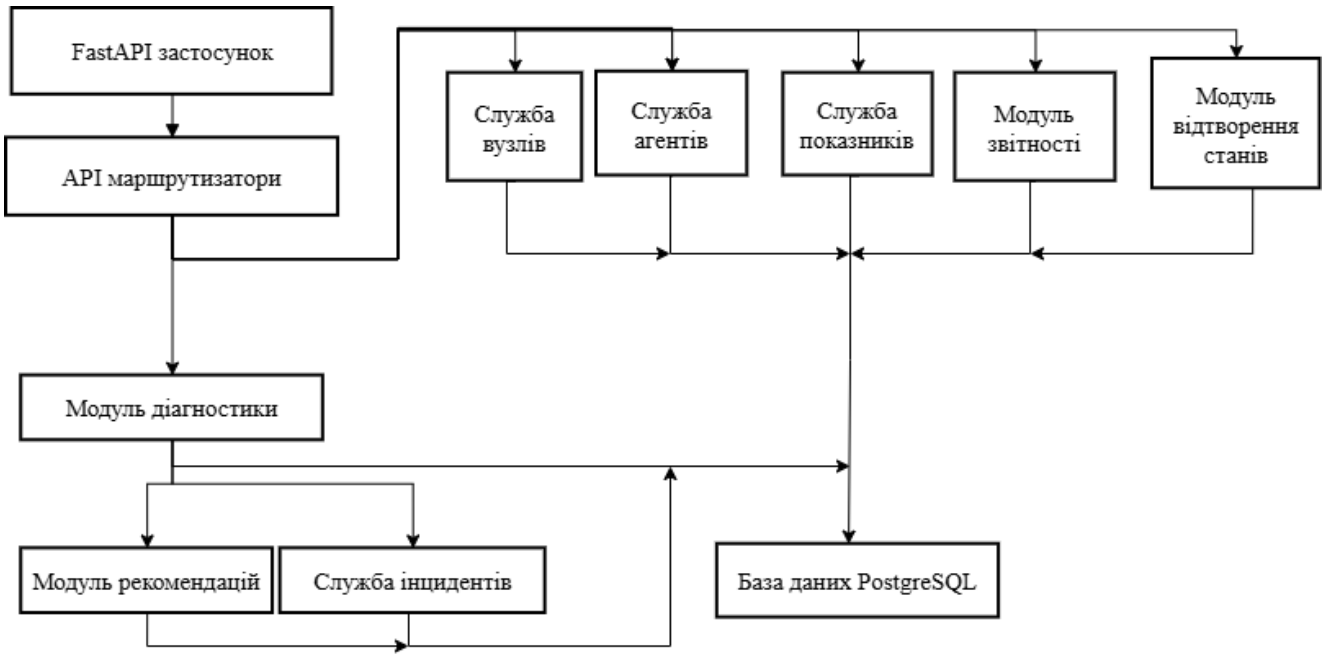


Рисунок. 3.2 – Структурна схема серверної частини системи діагностики та масштабування віртуалізованих обчислювальних ресурсів

До основних сутностей серверної частини належать середовище як логічне середовище або кластер, контрольний вузол, ключ доступу агента для підтвердження справжності, показник ресурсів як дані спостереження, діагностика як результат аналізу, інцидент, рекомендація, сценарій відтворення станів, прогноз планування місткості та сформований звіт.

Точки доступу /api/heartbeat та /api/metrics призначені для агента. Точки доступу /api/agents забезпечують реєстрацію вузла з агентом, створення ключа доступу та отримання команди встановлення. Точки доступу /api/nodes дозволяють працювати з вузлами та отримувати показники, діагностику, рекомендації, прогнози та виведення даних. Точки доступу /api/diagnostics, /api/incidents та /api/recommendations забезпечують роботу з результатами аналізу.

Серверна частина підтримує два режими даних: реальні показники від агентів та відтворені показники. Це дає змогу перевіряти систему навіть без окремої серверної інфраструктури.

Під час запуску FastAPI-застосунку використовується функція життєвого циклу. Вона ініціалізує базу даних і створює демонстраційне середовище, якщо база даних є порожньою. Це рішення покращує досвід першого запуску: користувач не бачить порожню панель спостереження, а одразу може переглянути вузли, показники, інциденти та рекомендації.

Точки доступу API згруповано за предметними областями. Наприклад, файл роботи з агентами відповідає за реєстрацію агента та керування ключами доступу, файл роботи з показниками за отримання даних, файл роботи з діагностикою за запуск і перегляд діагностики, файл роботи з відтворенням станів за сценарії, файл роботи з місткістю за прогнозування, файл роботи зі звітами за звіти. Така структура спрощує пошук у коді та відповідає принципам зручності супроводу.

Служба агентів виконує нормалізацію адреси сервера, створення ключа доступу, перевірку ключа та обробку сигналів справності. Ключ доступу створюється як таємне значення, у базі даних зберігаються його хеш і короткий зразок для попереднього перегляду. Короткий зразок потрібен для інтерфейсу, щоб користувач міг розпізнати ключ, не бачачи повного секрету.

Служба показників відповідає за створення запису показників ресурсів. Вона отримує вже перевірені вхідні дані та записує їх в базу даних. У подальшому цю службу можна розширити логікою попередньої обробки: нормалізацією одиниць вимірювання, відкиданням неправильних значень, узагальненням або зменшенням кількості точок.

Служба інцидентів створює інцидент на основі діагностики. Це дає змогу автоматично перетворювати результат аналізу на операційну подію. Якщо наступна діагностика показує здоровий стан, відкриті інциденти можуть бути закриті. Такий цикл наближує систему до практик керування інцидентами.

Створювач звітів формує результат для документування стану вузла або середовища.

Серверна частина також підтримує точки доступу для виведення даних у формати CSV та JSON. Це демонструє відкритість системи: дані не замкнені в панелі спостереження, їх можна отримати через API та використати для зовнішнього аналізу.

3.3 Реалізація модуля діагностики, модуля формування рекомендацій та інформаційної панелі системи

Механізм діагностики реалізує основну інтелектуальну функцію системи. Він завантажує останні 30 показників вузла, перевіряє надмірне виділення ресурсів, температуру, стан оперативної та додаткової пам'яті, процесор, активність диска, мережу, неповне використання ресурсів, погіршення роботи служби та показник аномальності. Результатом є запис діагностики (рис. 3.3).

Кожен запис діагностики містить тип діагностики, першопричину, ступінь важливості, впевненість у висновку, показник ризику, пояснення, докази у форматі JSON та стан. Якщо тип діагностики дорівнює здоровому стану, система закриває відкриті інциденти для вузла. Якщо тип діагностики не є задовільним, створюються інцидент та рекомендація. Це забезпечує повний цикл реагування.

Механізм формування рекомендацій створює поради українською мовою. Він враховує тип діагностики, поточну виділену кількість ядер процесора, поточний виділений обсяг оперативної пам'яті в мегабайтах, максимальну кількість ядер процесора, максимальний обсяг оперативної пам'яті та роль вузла. Для вузлів бази даних додається покрокова дія щодо перевірки затримок диска, кешування запитів та буфера або кешу. Для службових вузлів додається покрокова дія щодо горизонтального розширення кількості примірників служби.

Це дозволяє розподілити навантаження між декількома екземплярами застосунку. Такий підхід є особливо ефективним для сервісів без збереження стану, наприклад вебзастосунків або API-шлюзів.

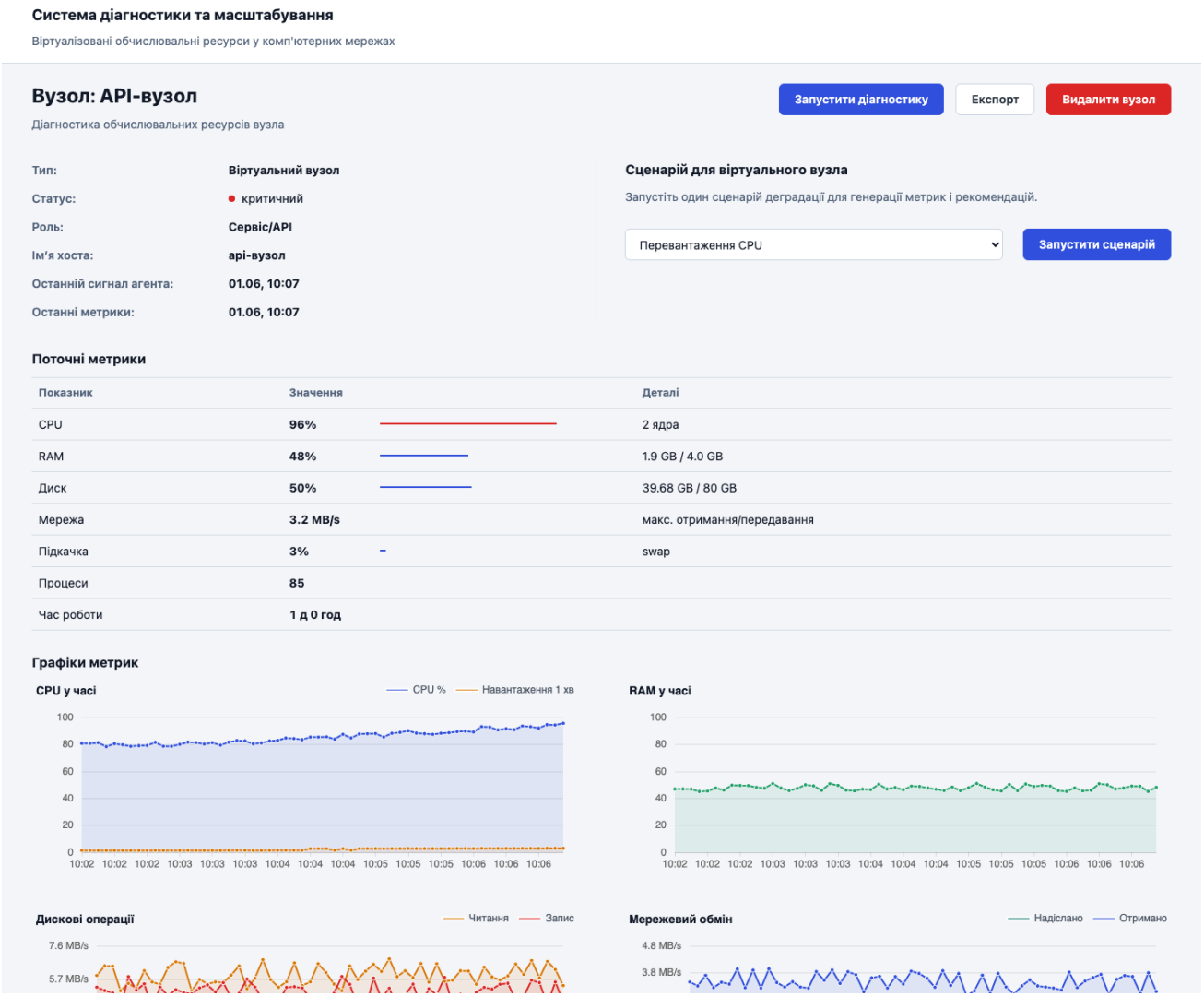


Рисунок. 3.3 – Представлення вузла який має критичний стан через високе використання процесора.

Для візуалізації використовуються компоненти графіків для процесора, оперативної пам'яті, додаткової пам'яті, диска, мережі, ризику та напряму зміни показників місткості. Панель спостереження звертається до серверної частини через REST API, базова адреса задається змінною середовища. У кінцевому варіанті ця змінна на платформі Vercel має значення, яке вказує на серверну частину на Render з префіксом /api, тому користувач відкриває посилання на Vercel, а всі API-запити автоматично спрямовуються до серверної частини на Render.

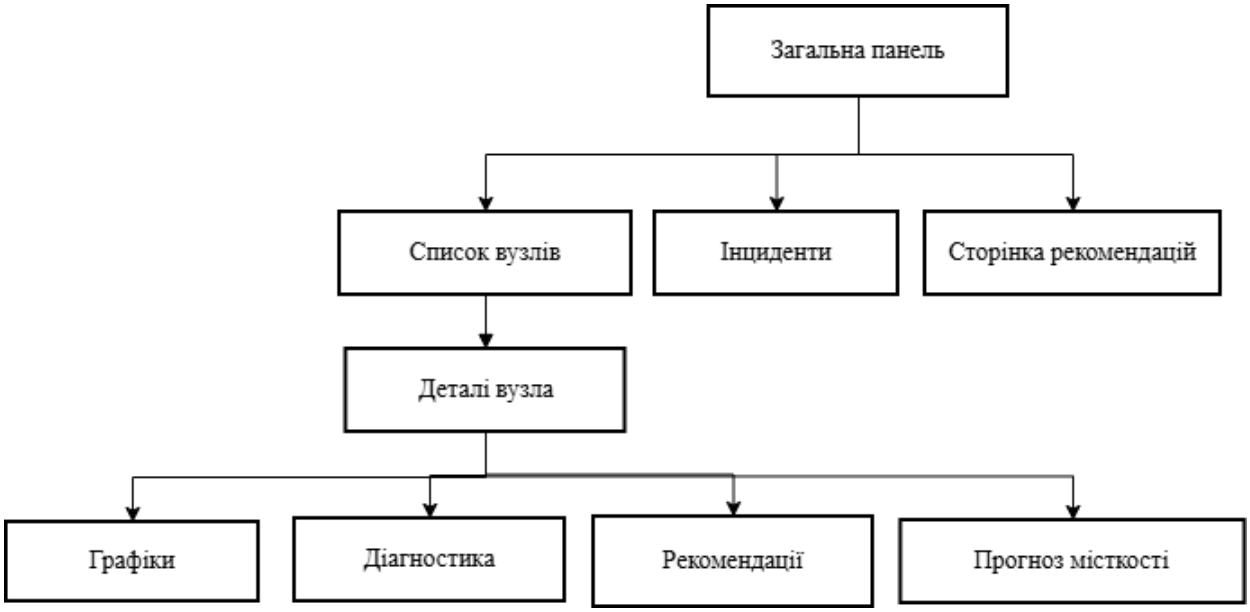


Рисунок. 3.4 – Структура користувацького вебінтерфейсу системи

Інтерфейс орієнтований на користувача, який має швидко оцінити стан інфраструктури. Для цього використовуються позначки стану, позначки ризику, картки показників, таблиці та графіки (рис. 3.4).

Механізм діагностики реалізовано як окрему службу. Це важливо для повторного використання. Його можна викликати з точки доступу запуску діагностики для вузла, з механізму відтворення станів після завершення сценарію або з майбутнього фонового планувальника. Такий підхід робить систему більш гнучкою.

Детектор аномалій працює у двох режимах. Якщо є щонайменше 12 показників, використовується метод, що знаходить рідкісні та нехарактерні комбінації показників, на основі набору ознак: використання процесора, оперативної пам'яті, додаткової пам'яті, заповнення диска, швидкості читання та запису диска, швидкості відправлення та отримання в мережі та температури. Якщо показників менше або модель машинного навчання недоступна, використовується перехідний варіант на основі правил. Це забезпечує стабільну роботу навіть у перші хвилини після підключення вузла.

Ступінь важливості визначається на основі показника ризику. Така модель є простою та зрозумілою. Вона дає змогу панелі спостереження однаково відображати різні типи проблем через позначку ризику.

Наприклад, навантаження процесора та тиск на файл підкачування можуть мати різні першопричини, але обидва можуть бути критичними, якщо показник ризику є високим.

Механізм формування рекомендацій створює не лише рекомендовані значення для кількісті ядер процесора та обсягу оперативної пам'яті, а й причину та очікуваний вплив. Це важливо, оскільки дія з розширення потужностей має бути обґрунтованою. Наприклад, для вузького місця диска рекомендована кількість ядер процесора та обсяг оперативної пам'яті можуть залишатися поточними, оскільки проблема не в процесорі чи пам'яті. У такому разі очікуваний вплив пов'язаний зі зменшенням затримок введення та виведення.

Панель спостереження реалізує типову операційну навігацію. Користувач починає з загальної панелі, переходить до проблемного вузла, аналізує графіки, переглядає діагностику та докази, а потім переходить до рекомендацій. Такий робочий процес відповідає реальній роботі користувача: спочатку потрібно побачити загальну картину, потім деталізувати проблему.

Компоненти графіків винесені окремо: графік процесора, графік оперативної пам'яті, графік диска, графік мережі, графік додаткової пам'яті, графік ризику та графік напряму зміни показників місткості. Це зменшує дублювання і дає змогу однаково відображати дані спостереження на різних сторінках.

Клієнти для користувацької частини розділені за областями: агенти, місткість, загальна панель, діагностика, середовища, інциденти, вузли, рекомендації, звіти та відтворення станів. Це відповідає маршрутизаторам серверної частини та робить код користувацької частини більш зрозумілим.

3.4 Тестування роботи системи та аналіз результатів діагностики

Для перевірки роботи системи використано демонстраційні та відтворювальні сценарії. При першому запуску серверна частина створює демонстраційний віртуальний кластер, якщо база даних є порожньою. У цьому середовищі створюються вузли веб-вузол, вузол API, вузол бази даних та робочий вузол. Для них створюється історія показників у станах нормального роботи, навантаження процесора, нестачі пам'яті та неповного використання ресурсів.

Першою відкривається головна сторінка (рис.3.5). На ній користувач бачить список уже створених вузлів, їхній тип, статус, поточні значення та кнопку «Додати вузол». Для тестування створюється новий віртуальний вузол, тому користувач натискає «Додати вузол» і переходить на сторінку створення.

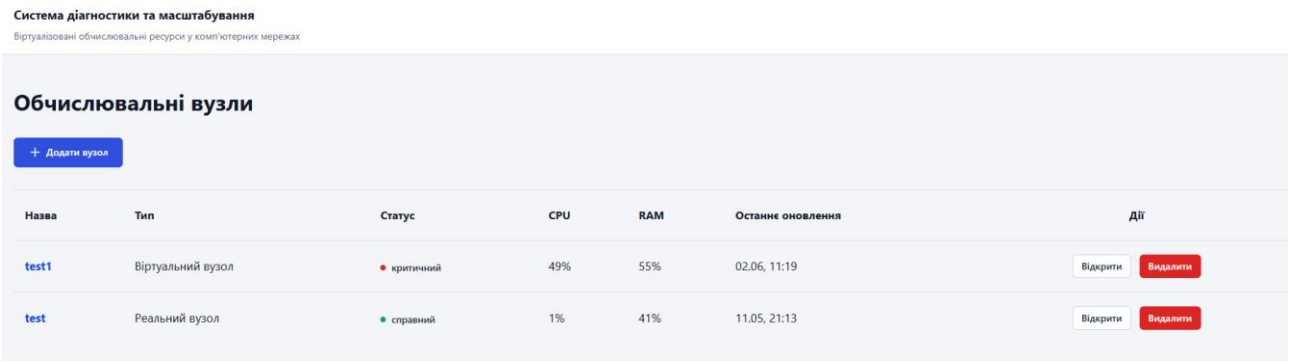


Рисунок. 3.5 – Інформаційна панель системи

На сторінці створення вибирається тип «Віртуальний вузол». Далі задаються назва, роль, кількість ядер процесора, обсяг оперативної пам'яті та розмір диска. Після натискання кнопки створення користувацька частина надсилає запит до серверної частини для створення вузла, серверна частина створює запис вузла в базі даних, а користувач автоматично переходить на сторінку цього вузла (рис. 3.6).

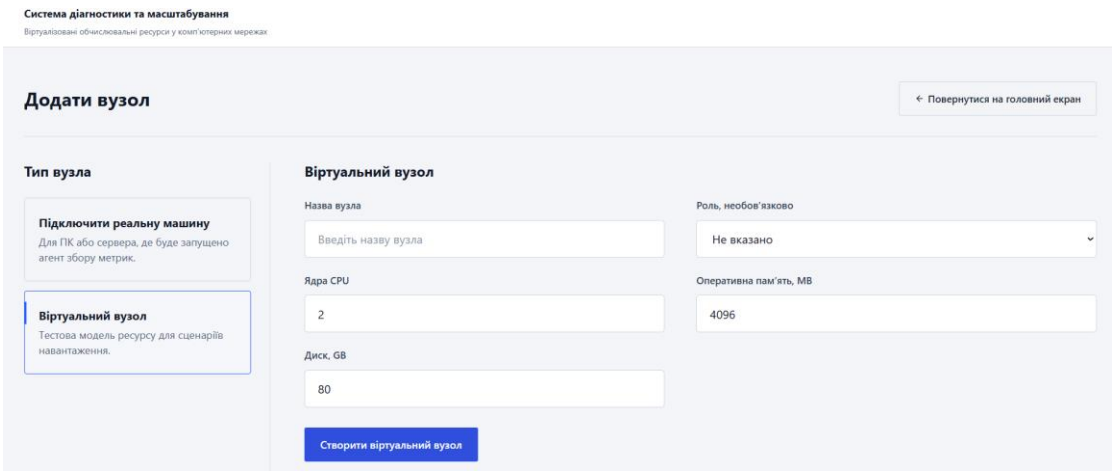


Рисунок. 3.6 – Сторінка створення вузла

Сторінка вузла у тестуванні використовується як сторінка результатів. До запуску сценарію на ній перевіряються базові дані вузла: тип, статус, роль, останні метрики та доступність кнопок «Запустити діагностику», «Експорт» і запуску сценарію для віртуального вузла. Після запуску сценарію з'являються оновлені графіки, діагноз, інциденти і рекомендації (рис. 3.7).

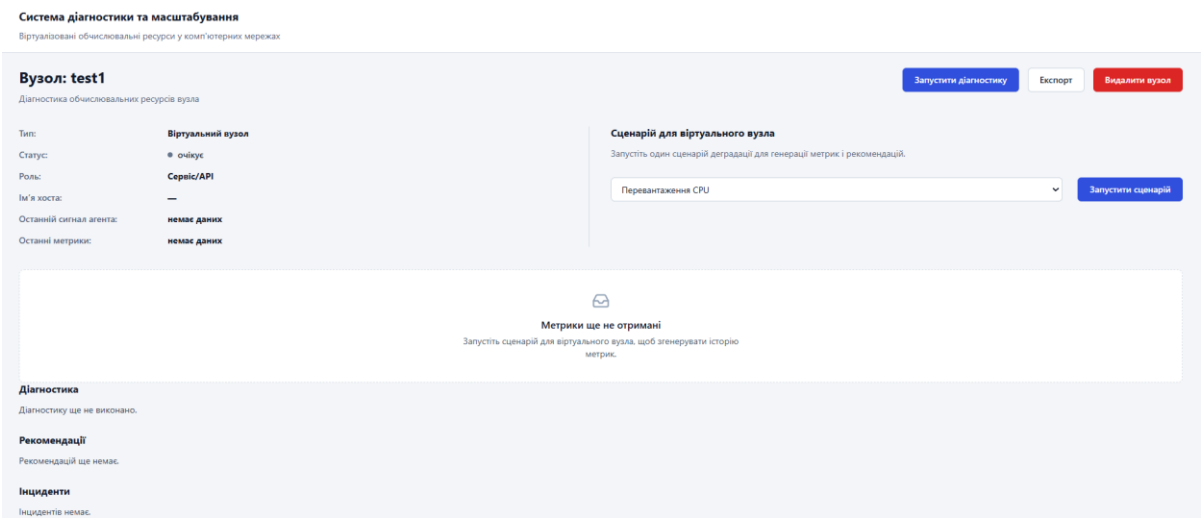


Рисунок. 3.7 – Сторінка аналізу стану вузла

Для перевірки було обрано сценарій перевантаження процесора. Користувач вибирає сценарій у блоці віртуального вузла і натискає «Запустити сценарій». Після завершення сценарію сторінка вузла оновлюється. На ній з’являються три групи результатів (рис. 3.8).

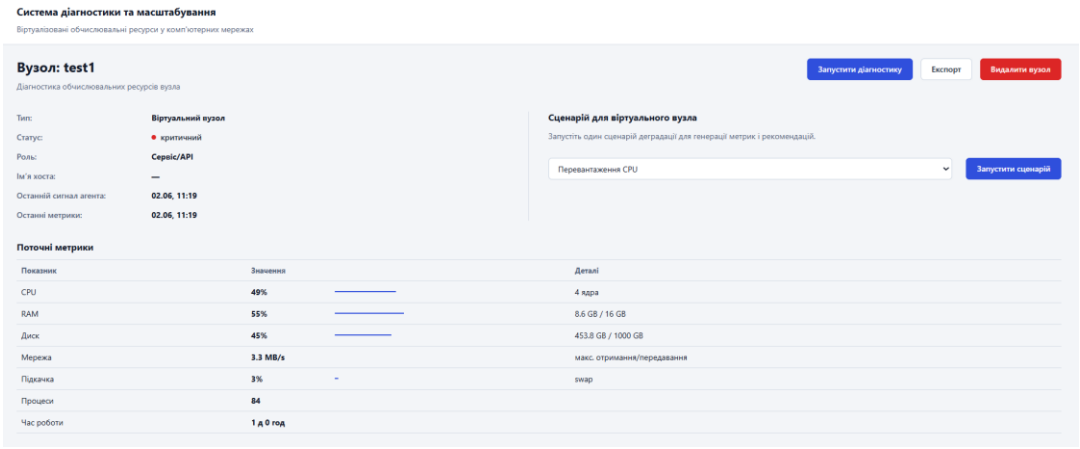


Рисунок. 3.8 – Сторінка аналізу стану вузла з отриманими даними+

Перша група це показники та графіки: значення використання процесора зростає до критичного рівня, а графік показує зміну навантаження в часі (рис. 3.9).

Рисунок. 3.9 – Показники та графіки

Друга група це результат діагностики: система визначила тип проблеми як насичення процесора, показала ступінь важливості, ризик та пояснення причини. Третя група це реакція системи: створені інциденти та рекомендації щодо збільшення потужностей процесора або розгляду горизонтального масштабування (рис. 3.10).

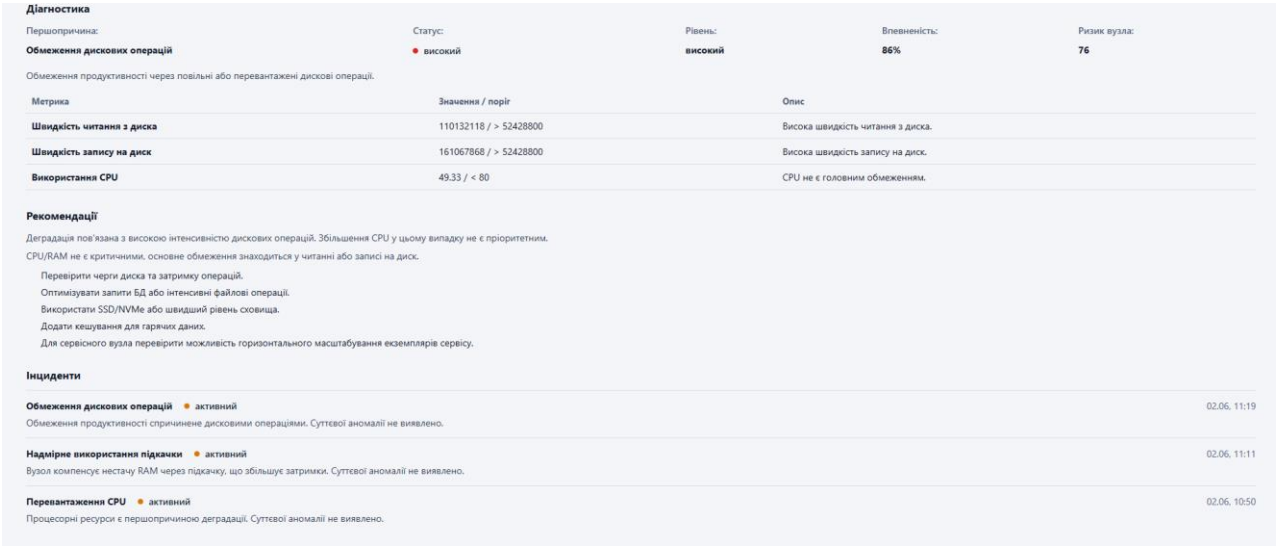


Рисунок. 3.10 – Результат діагностики системи

Фінальним частиною є експорт результатів. На сторінці вузла натискається кнопка «Експорт», після чого серверна створює звіт у форматі PDF. Дані сфомовані у результаті роботи системи зображені на рисунку Г.1 та на рисунку Г.2.

Отже, результати діагностики показали, що першопричиною критичного стану вузла є перевантаження процесора з упевненістю 92% та ризиком вузла на рівні 97. Система виявила, що використання процесора досягло 97,26% при порозі 85%, а середнє значення черги завдань за останню хвилину становило 6,22, що свідчить про стабільно високе навантаження та нестачу процесорних ресурсів. На основі цього сформовано рекомендацію щодо збільшення кількості ядер процесора з 4 до 8, перевірки процесів, які найбільше використовують процесор, розгляду горизонтального масштабування примірників застосунку та перевірки політик балансування навантаження.

Тепер перевірим інші сценарії деградації: нестача пам'яті, проблема з диском.

Дані сфомовані у результаті роботи системи з деградацією нестача пам'яті зображені на рисунку Г.3.

Діагностика показала, що першопричиною критичного стану вузла є надмірне використання файлу підкачування при впевненості 90% та ризику вузла

на рівні 94. Система виявила, що використання оперативної пам'яті досягло 93,88% при порозі 85%, а використання файлу підкачування склало 28,58% при порозі 20%, що свідчить про нестачу оперативної пам'яті та активне використання підкачування для компенсації. На основі цього сформовано рекомендацію щодо збільшення обсягу оперативної пам'яті з 15,6 ГБ до 31,2 ГБ, а також перевірки процесів, що активно використовують пам'ять, та витоків пам'яті. Для сервісного вузла додатково запропоновано розглянути горизонтальне масштабування примірників служби.

Дані сфомовані у результаті роботи системи з деградацією проблема з диском зображені на рисунку Г.4.

Результати діагностики показали, що першопричиною високого рівня критичності вузла є обмеження продуктивності через повільні або перевантажені дискові операції з упевненістю 86%. Система виявила, що швидкість читання з диска становить понад 110 МБ/с, швидкість запису понад 161 МБ/с при пороговому значенні 50 МБ/с, при цьому використання процесора становить 49,33 %, що не є критичним. На основі цього сформовано рекомендацію зосередитися на оптимізації дискових операцій, а не на збільшенні процесорних потужностей, зокрема перевірити черги диска та затримку операцій, оптимізувати запити до бази даних, використати швидший рівень сховища та додати кешування для гарячих даних.

У ході тестування системи було запущено серію сценаріїв. Фактичні результати роботи системи повністю відповідають очікуваній логіці діагностики та формування рекомендацій.

3.5. Оцінка ефективності, стабільності та надійності запропонованого рішення

Ефективність запропонованого рішення оцінюється за здатністю системи виконувати повний цикл діагностики: збирання даних спостереження,

збереження, аналіз, визначення першопричини, створення інциденту, формування рекомендації та відображення результату в панелі спостереження. Розроблена система виконує цей цикл для реальних вузлів з агентами та відтворених віртуальних вузлів.

Сильними сторонами рішення є модульна архітектура, простий запуск локально або через Docker Compose для розробки, кінцеве розгортання серверної частини на Render, кінцеве розгортання користувацької частини на Vercel, використання Supabase PostgreSQL як основної віддаленої бази даних, агентський збір реальних системних показників, безпечна модель агента без віддалених системних команд, пояснювана діагностика через поле з доказами у форматі JSON, структуровані рекомендації з покроковими діями, підтримка сценаріїв відтворення станів для перевірки алгоритму, панель спостереження українською мовою, а також виведення даних у формати CSV та JSON і створення звітів.

Стабільність системи забезпечується логікою повторних спроб агента, перевіркою справжності за допомогою ключів доступу, відокремленістю служб серверної частини та відсутністю небезпечних функцій безпосереднього виконання дій. Серверна частина може працювати локально з SQLite, а в кінцевому розгортанні використовує Supabase PostgreSQL. Публічна користувацька частина на Vercel забезпечує доступ до панелі спостереження через посилання, а серверна частина на Render приймає API-запити від панелі спостереження та дані спостереження від агентів (рис. 3.11).

Розроблена система демонструє практичну реалізацію робочого процесу діагностики та формування рекомендацій для віртуалізованих обчислювальних ресурсів і може бути розширена в майбутньому. Потенційними напрямами розвитку є поєднання з Prometheus як джерелом показників, підтримка показників Kubernetes, автоматичне закриття рекомендацій після повернення стану до норми, встановлення порогових значень залежно від ролі вузла, розширене виявлення аномалій, підтримка сповіщень через веб-гачки, а також

поєднання з програмними інтерфейсами автоматичного розширення в хмарі в режимі з підтвердженням від людини.

Рисунок. 3.11 – Блок-схема обробки телеметричних даних та формування рекомендацій

З погляду ефективності діагностики, основною перевагою системи є те, що вона скорочує шлях від даних спостереження до рішення користувача. Без такої системи користувач повинен вручну переглянути графіки використання процесора, оперативної пам'яті, диска та мережі, порівняти їх між собою, згадати порогові значення, зробити висновок і сформулювати план дій. Розроблена система автоматизує значну частину цього процесу, але залишає остаточне рішення за людиною.

З погляду стабільності, рішення є достатнім для локального дослідного зразка. Логіка повторних спроб агента зменшує вплив короткочасних мережевих помилок. Серверна частина має просту архітектуру без збереження стану між запитами, а стан зберігається в Supabase PostgreSQL. Користувацька частина отримує дані через REST API і не залежить від прямого доступу до бази даних. Користувач працює з системою через посилання на Vercel, що робить інтерфейс доступним без локального запуску користувацької частини.

З погляду надійності рекомендацій, важливо, що система не формує рекомендацію без наявності діагностики. Рекомендація завжди прив'язана до першопричини та доказів. Це зменшує ризик випадкових або необґрунтованих порад. Крім того, для невідомого погіршення система не рекомендує зміну потужностей, а пропонує зібрати більше показників і перевірити журнали подій.

Оцінка системи також показує, що її можна розширювати поступово. Наприклад, можна додати порогові значення залежно від ролі вузла без зміни агента. Можна додати новий тип сценарію без зміни архітектури панелі спостереження. Можна додати новий вид графіка або новий тип звіту без зміни механізму діагностики. Це свідчить про достатню модульність.

Для промислового рівня можна було б додатково реалізувати перевірку справжності для користувачів панелі спостереження, захищене з'єднання, правила зберігання даних, фонові завдання, сповіщення, канали сповіщень, обмеження частоти запитів і глибше спостереження за самою серверною частиною.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень було розглянуто задачу діагностики та формування рекомендацій щодо масштабування віртуалізованих обчислювальних ресурсів у комп'ютерних мережах.

У першому розділі проведено аналіз предметної області, особливостей функціонування віртуалізованих ресурсів, принципів спостереження серверної інфраструктури та наявних платформ стеження за станом. У результаті аналізу встановлено, що сучасні системи спостереження ефективно збирають і візуалізують дані спостереження, але часто залишають визначення першопричини погіршення роботи та вибір стратегії масштабування на адміністратора. Це створює потребу в системі, яка поєднує спостереження, діагностику, керування інцидентами, планування місткості та механізм формування рекомендацій.

У другому розділі проведено проєктування системи діагностики та формування рекомендацій щодо масштабування. У межах роботи спроєктовано систему, яка складається з засобу спостереження на Python, серверної частини на FastAPI, моделі бази даних SQLAlchemy, механізму діагностики, механізму формування рекомендацій, механізму відтворення станів, планувальника місткості, створювача звітів та панелі спостереження на SvelteKit. Система підтримує реальні вузли з агентами та відтворені віртуальні вузли.

У третьому розділі виконано програмну реалізацію та перевірку роботи системи діагностики. Реалізований механізм діагностики визначає такі стани, як навантаження процесора, нестача пам'яті, тиск на файл підкачування, вузьке місце диска, нестача мережевих ресурсів, загроза перегріву, надмірне виділення ресурсів, неповне використання ресурсів, погіршення роботи служби та здоровий стан. Для кожної діагностики формується показник ризику, ступінь важливості, впевненість у висновку та докази у форматі JSON. Механізм

формування рекомендацій створює практичні поради українською мовою з покроковими діями. Перевірка системи виконувалася за допомогою демонстраційного середовища та сценаріїв відтворення станів. Результати показали, що розроблена система здатна виявляти типові вузькі місця ресурсів, створювати інциденти, формувати рекомендації та відображати стан вузлів у панелі спостереження. Практична цінність роботи полягає в тому, що розроблена система може використовуватися як навчальний і демонстраційний інструмент для вивчення принципів спостереження, діагностики та масштабування віртуалізованих ресурсів. Розроблена система має низку переваг перед існуючими аналогами в контексті навчальних проєктів. Вона є повністю відкритою та має зрозумілу логіку діагностики, що дозволяє користувачеві бачити докази кожного висновку. Також не потребує складної інсталяції та може працювати як локально, так і в хмарному середовищі. Крім того, система забезпечує можливість відтворення сценаріїв деградації без реального навантаження на обладнання, що є безпечним для навчальних цілей. Розроблена архітектура є модульною, тому кожен компонент може вдосконалюватися незалежно. Використані технології є сучасними та широко розповсюдженими, що полегшує вивчення коду та подальшу модернізацію системи. Таким чином, результати роботи можуть бути впроваджені в навчальний процес спеціальностей, пов'язаних з комп'ютерними мережами та системним адмініструванням. Система також може бути основою для подальшої інтеграції з Prometheus, Kubernetes, програмними інтерфейсами хмарних провайдерів та сповіщеннями.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Red Hat Virtualization Documentation. URL: https://access.redhat.com/documentation/ (дата звернення: 09.05.2026).
2. VMware Virtualization Documentation. URL: https://docs.vmware.com/ (дата звернення: 09.05.2026).
3. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. 395 с.
4. ISO/IEC 25010:2011. Systems and software engineering Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models. Geneva : International Organization for Standardization, 2011.
5. Docker Compose Documentation. URL: https://docs.docker.com/compose/ (дата звернення: 09.05.2026).
6. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 09.05.2026).
7. Burns B., Grant B., Oppenheimer D., Brewer E., Wilkes J. Borg, Omega, and Kubernetes // Communications of the ACM. 2016. Vol. 59, № 5. P. 50–57.
8. Hightower K., Burns B., Beda J. Kubernetes: Up and Running. 3rd ed. O'Reilly Media, 2022. 386 с.
9. Merkel D. Docker: Lightweight Linux Containers for Consistent Development and Deployment // Linux Journal. 2014. № 239. С. 2–7.
10. Turnbull J. The Docker Book: Containerization is the New Virtualization. James Turnbull, 2014. 348 с.
11. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 09.05.2026).
12. Supabase Documentation. URL: https://supabase.com/docs (дата звернення: 09.05.2026).
13. FastAPI Documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 09.05.2026).

14. SQLAlchemy Documentation. URL: https://docs.sqlalchemy.org/ (дата звернення: 09.05.2026).

15. SvelteKit Documentation. URL: https://kit.svelte.dev/docs (дата звернення: 09.05.2026).

16. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs (дата звернення: 09.05.2026).

17. MDN Web Docs. URL: https://developer.mozilla.org/ (дата звернення: 09.05.2026).

18. Datadog Infrastructure Monitoring Documentation. URL: https://docs.datadoghq.com/ (дата звернення: 09.05.2026).

19. Prometheus Documentation. Overview. URL: https://prometheus.io/docs/introduction/overview/ (дата звернення: 09.05.2026).

20. Zabbix Documentation. What is Zabbix. URL: https://www.zabbix.com/documentation/current/en/manual/introduction/about (дата звернення: 09.05.2026).

21. psutil Documentation. URL: https://psutil.readthedocs.io/ (дата звернення: 09.05.2026).

22. Newman S. Building Microservices. 2nd ed. O'Reilly Media, 2021. 617 с.

23. Alembic Documentation. URL: https://alembic.sqlalchemy.org/ (дата звернення: 09.05.2026).

24. Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media, 2017. 616 с.

25. scikit-learn Documentation. IsolationForest. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html (дата звернення: 09.05.2026).

26. Silberschatz A., Galvin P. B., Gagne G. Operating System Concepts. 10th ed. Wiley, 2018. 944 с.

27. Render Documentation. Web Services. URL: https://render.com/docs/web-services (дата звернення: 09.05.2026).

28. Vercel Documentation. Deployments. URL: https://vercel.com/docs/deployments (дата звернення: 09.05.2026).

29. Narkhede N., Shapira G., Palino T. Kafka: The Definitive Guide. O'Reilly Media, 2017. 297 с.

30. Stallings W. Data and Computer Communications. 10th ed. Pearson, 2013. 912 с.

31. Tanenbaum A. S., Wetherall D. J. Computer Networks. 5th ed. Pearson Education, 2011. 960 с.

32. Pressman R. Software Engineering: A Practitioner's Approach. 9th ed. McGraw-Hill Education, 2019. 880 с.

33. Sommerville I. Software Engineering. 10th ed. Pearson, 2015. 816 с.

34. OpenTelemetry Documentation. URL: https://opentelemetry.io/docs/ (дата звернення: 09.05.2026).

35. HTC DeepQ Research Engineering Team. Build A Monitoring Dashboard by Prometheus + Grafana. Medium. 2019. URL: https://medium.com/htc-research-engineering-blog/build-a-monitoring-dashboard-by-prometheus-grafana-741a7d949ec2 (дата звернення: 09.05.2026).

36. Grafana Documentation. Dashboards. URL: https://grafana.com/docs/grafana/latest/dashboards/ (дата звернення: 09.05.2026).

37. Apache ECharts Documentation. URL: https://echarts.apache.org/en/index.html (дата звернення: 09.05.2026).

38. Zabbix Documentation. Dashboards. URL: https://www.zabbix.com/documentation/devel/en/manual/web_interface/frontend_sections/dashboards (дата звернення: 09.05.2026).

39. Nagios Core Features. Nagios Open Source. URL: https://www.nagios.org/projects/nagios-core/features/ (дата звернення: 09.05.2026).

40. Nagios Core. Nagios Open Source. URL: https://www.nagios.org/projects/nagios-core/ (дата звернення: 09.05.2026).

41. Kelleher M., Truong K., Marcantonio E., Iglesias D. Design Effective Executive Dashboards with Datadog. Datadog Blog. URL: https://www.datadoghq.com/blog/datadog-executive-dashboards/ (дата звернення: 09.05.2026).

42. Elastic Observability solution overview. Elastic. URL: https://www.elastic.co/docs/solutions/observability (дата звернення: 09.05.2026).

43. Matthes E. Python Crash Course. 3rd ed. No Starch Press, 2023. 552 с.

44. Ramalho L. Fluent Python. 2nd ed. O'Reilly Media, 2022. 1014 с.

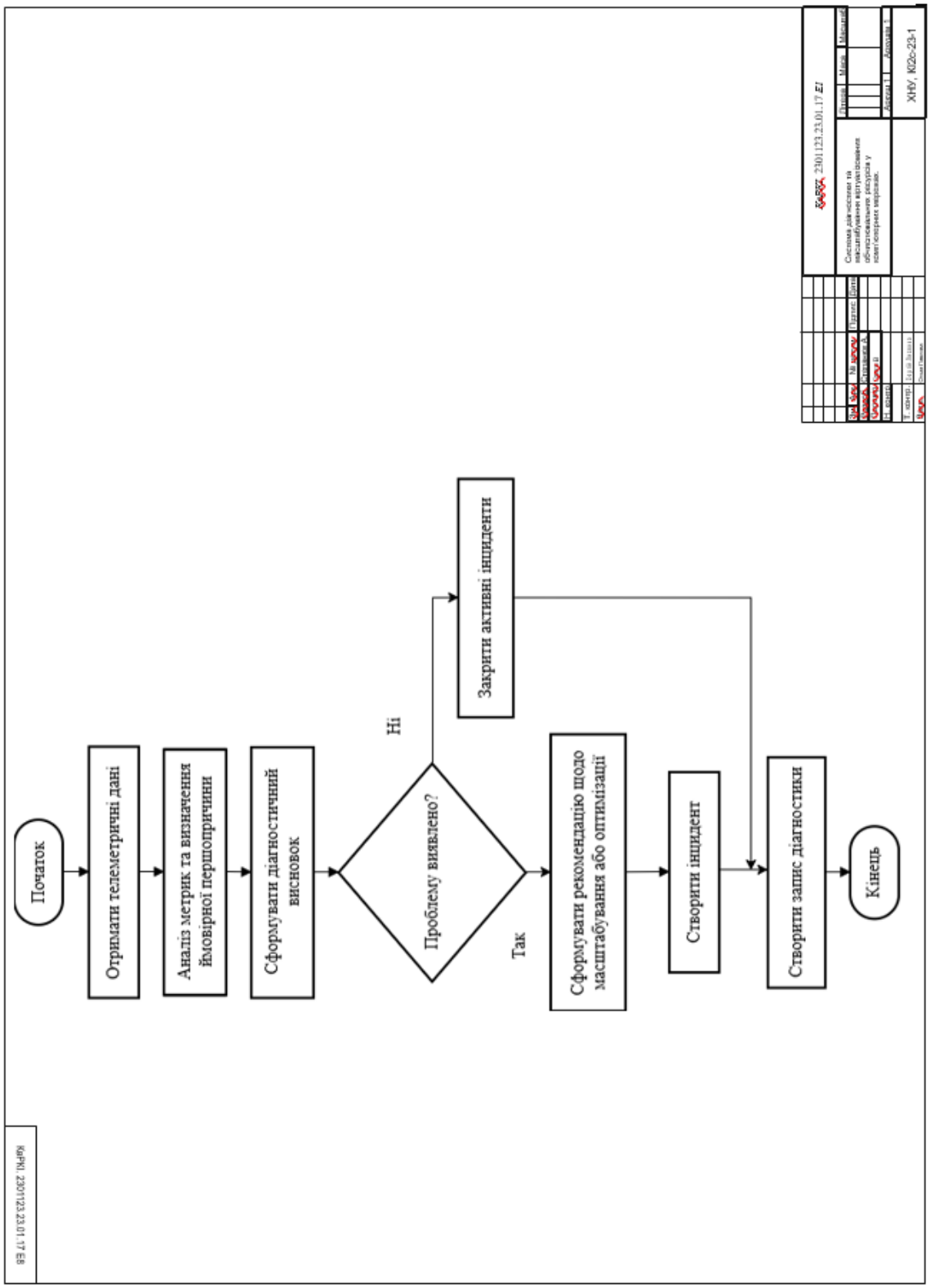
45. Foster I., Kesselman C. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 2004. 748 с.

46. Pydantic Documentation. URL: https://docs.pydantic.dev/ (дата звернення: 09.05.2026).

ДОДАТОК А

(обов’язковий)

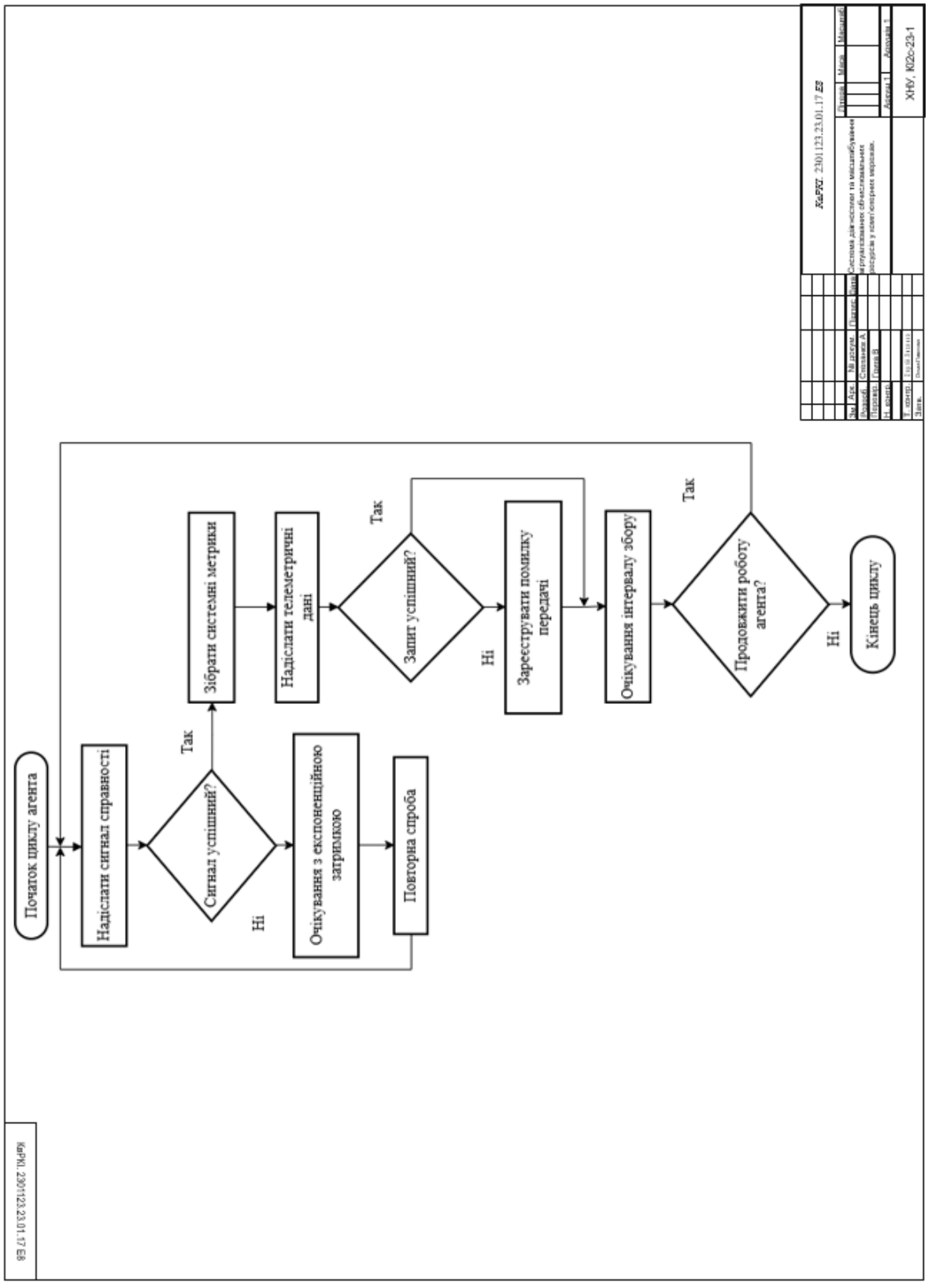
Копія креслення «Блок-схема алгоритму діагностики стану вузла та формування рекомендацій»



ДОДАТОК Б

(обов’язковий)

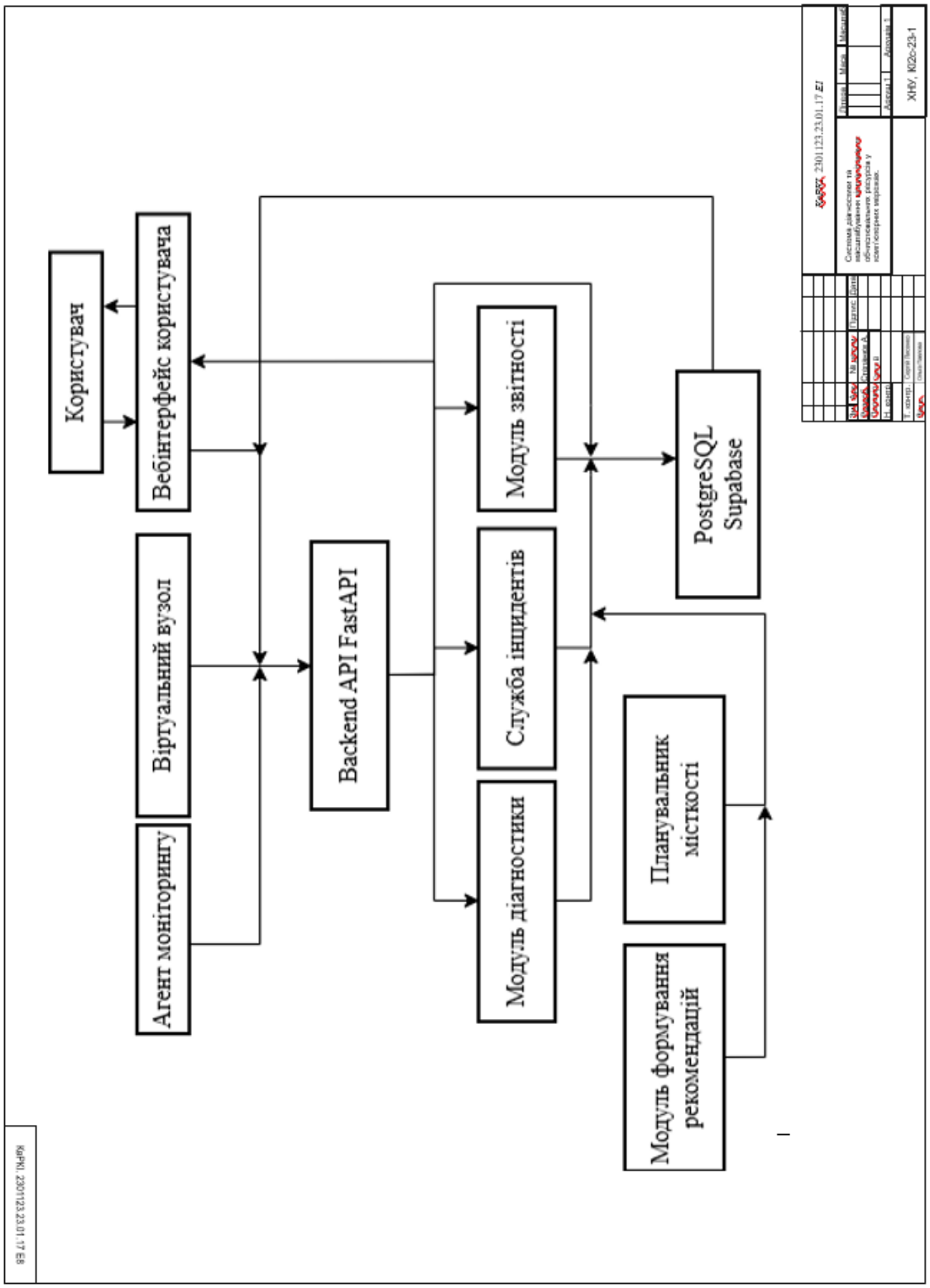
Копія креслення «Блок-схема роботи агента збору телеметричних даних»



ДОДАТОК В

(обов'язковий)

Копія креслення «Структурна схема компонентів системи діагностики та масштабування віртуалізованих ресурсів»

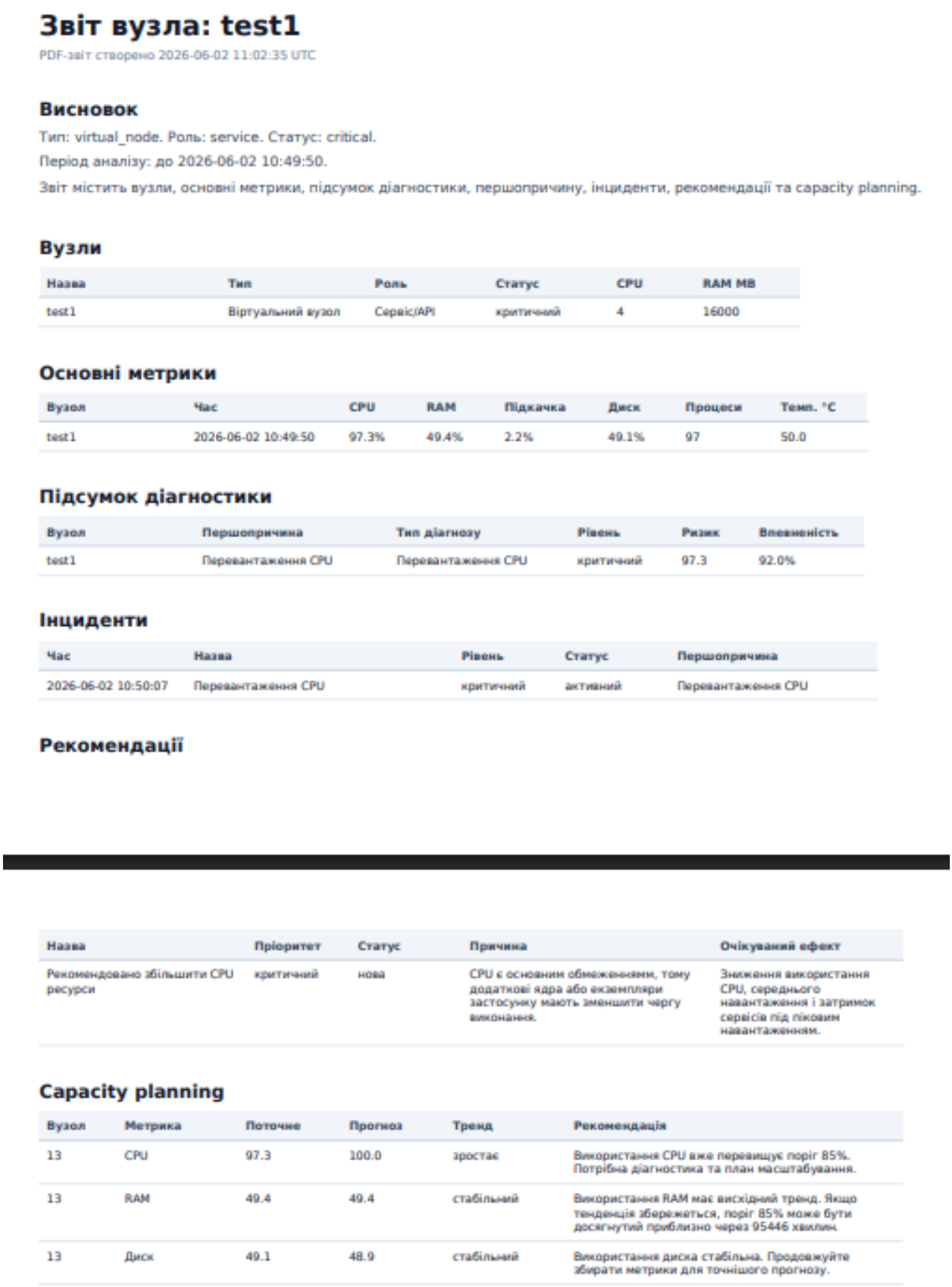


ДОДАТОК Г

(обов’язковий)

Графічний інтерфейс користувача веб-застосунку

Рисунок. Г.1 – Сторінка аналізу стану вузла з деградацією перевантаження процесора



Звіт вузла: test1

PDF-звіт створено 2026-06-02 11:02:35 UTC

Висновок

Тип: virtual_node. Роль: service. Статус: critical.

Період аналізу: до 2026-06-02 10:49:50.

Звіт містить вузли, основні метрики, підсумок діагностики, першопричину, інциденти, рекомендації та capacity planning.

Вузли

Назва	Тип	Роль	Статус	CPU	RAM MB
test1	Віртуальний вузол	Сервіс/API	критичний	4	16000

Основні метрики

Вузол	Час	CPU	RAM	Підкачка	Диск	Процеси	Темп. °C
test1	2026-06-02 10:49:50	97.3%	49.4%	2.2%	49.1%	97	50.0

Підсумок діагностики

Вузол	Першопричина	Тип діагнозу	Рівень	Ризик	Впевненість
test1	Перевантаження CPU	Перевантаження CPU	критичний	97.3	92.0%

Інциденти

Час	Назва	Рівень	Статус	Першопричина
2026-06-02 10:50:07	Перевантаження CPU	критичний	активний	Перевантаження CPU

Рекомендації

Назва	Пріоритет	Статус	Причина	Очікуваний ефект
Рекомендовано збільшити CPU ресурси	критичний	нова	CPU є основним обмеженням, тому додаткові ядра або екземпляри застосунку мають зменшити чергу виконання.	Зниження використання CPU, середнього навантаження і затримок сервісів під піковим навантаженням.

Capacity planning

Вузол	Метрика	Поточне	Прогноз	Тренд	Рекомендація
13	CPU	97.3	100.0	зростає	Використання CPU вже перевищує поріг 85%. Потрібна діагностика та план масштабування.
13	RAM	49.4	49.4	стабільний	Використання RAM має висхідний тренд. Якщо тенденція збережеться, поріг 85% може бути досягнутий приблизно через 95446 хвилин.
13	Диск	49.1	48.9	стабільний	Використання диска стабільна. Продовжуйте збирати метрики для точнішого прогнозу.

Рисунок. Г.2 – Сформований звіт за результатами діагностики

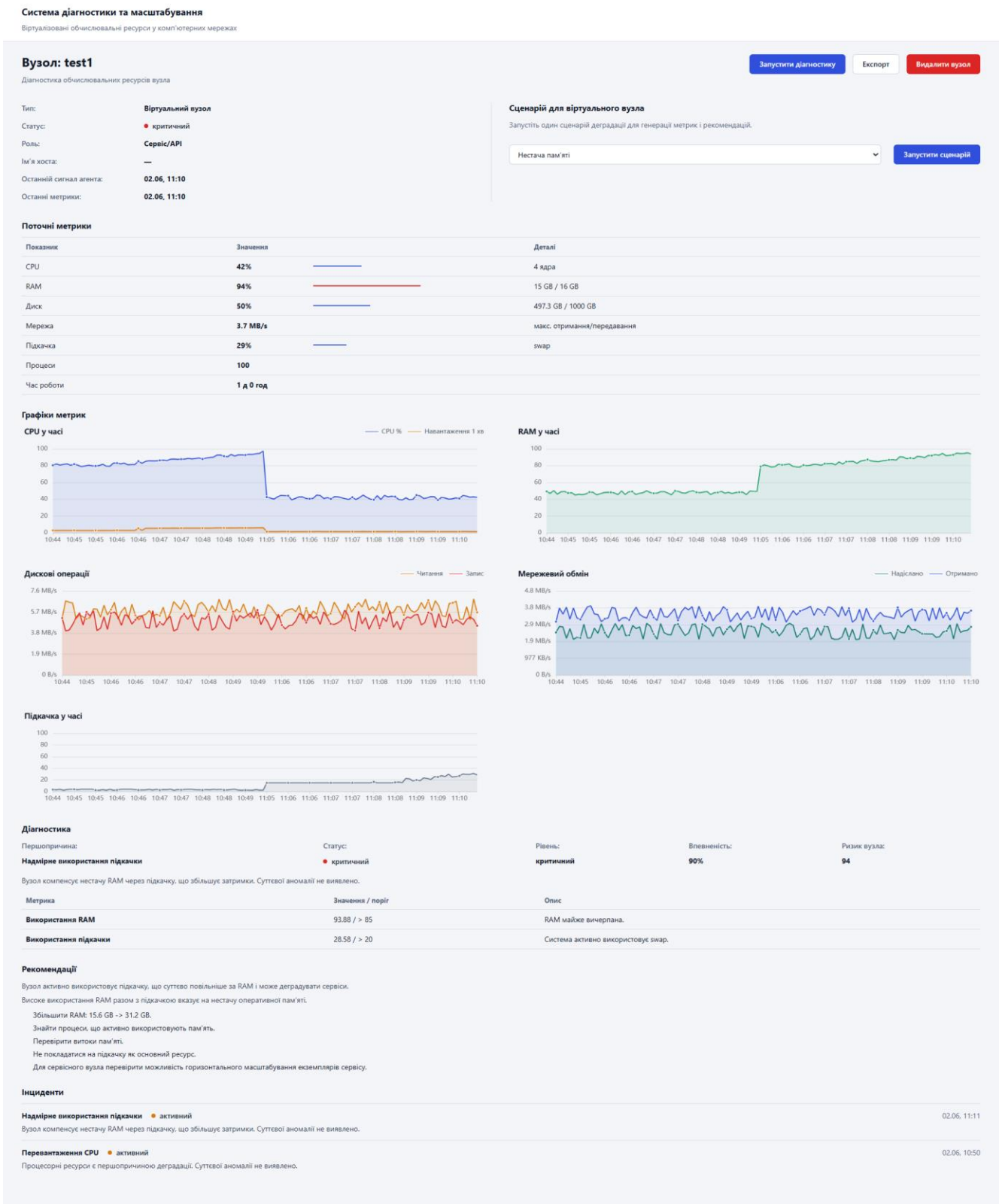


Рисунок. Г.3 – Сторінка аналізу стану вузла з деградацією нестачі пам'яті

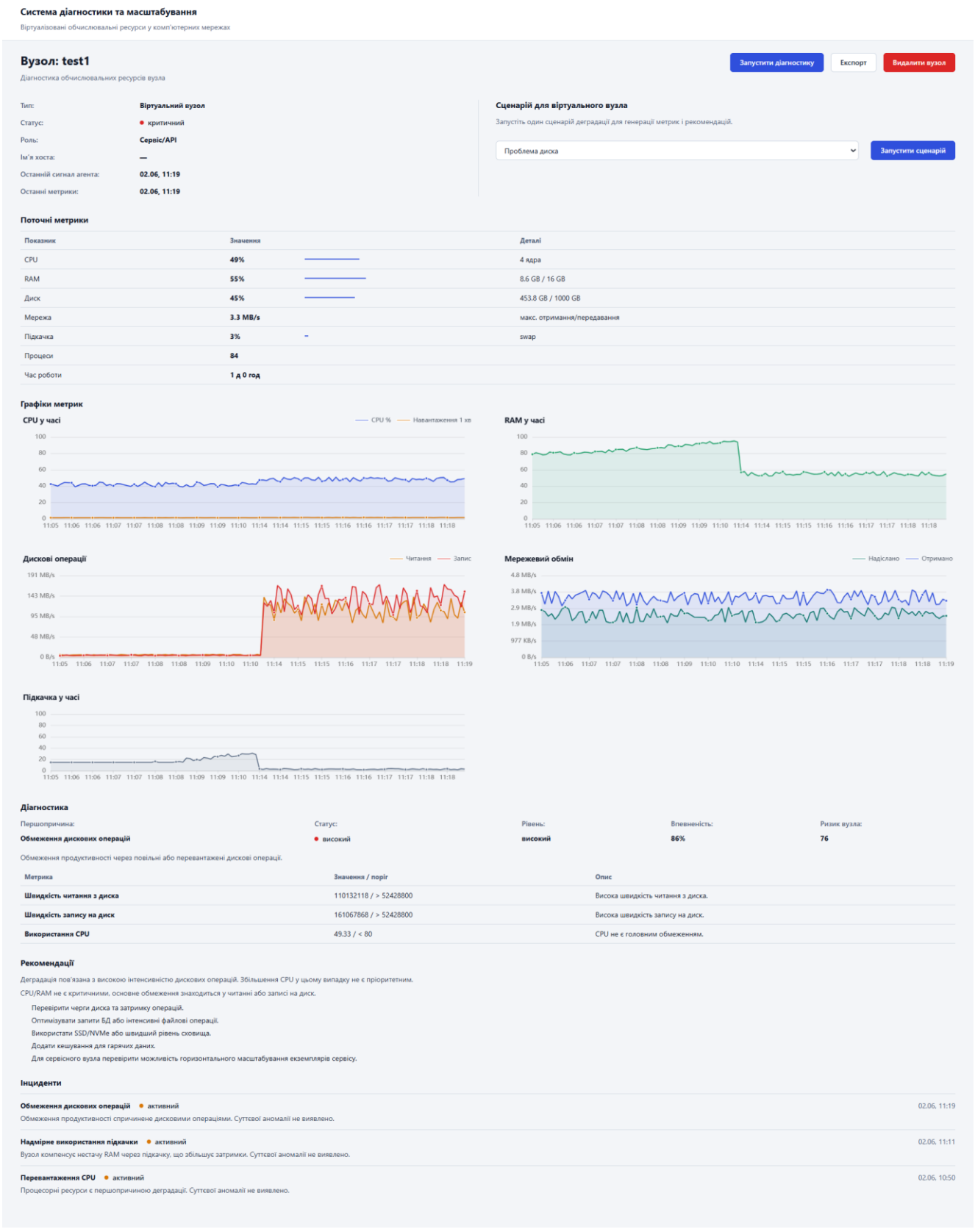


Рисунок. Г.4 – Сторінка аналізу стану вузла з деградацією проблеми диска

ДОДАТОК Д

(обов'язковий)

Текст системного програмного забезпечення

```
class MetricsCollector:
    def __init__(self) -> None:
        self.last_time = None
        self.last_disk = None
        self.last_net = None

    def collect(self, node_id: int, token: str) -> dict:
        now = time.time()
        cpu = psutil.cpu_percent(interval=0.2)
        ram = psutil.virtual_memory()
        swap = psutil.swap_memory()
        disk = psutil.disk_usage(Path.home().anchor or "/")
        disk_io = psutil.disk_io_counters()
        net = psutil.net_io_counters()
        elapsed = max(now - self.last_time, 1) if self.last_time else 1

        disk_read_rate = disk_write_rate = network_sent_rate = network_recv_rate = 0.0
        if self.last_disk and disk_io:
            disk_read_rate = max((disk_io.read_bytes - self.last_disk.read_bytes) / elapsed, 0)
            disk_write_rate = max((disk_io.write_bytes - self.last_disk.write_bytes) / elapsed, 0)
        if self.last_net and net:
            network_sent_rate = max((net.bytes_sent - self.last_net.bytes_sent) / elapsed, 0)
            network_recv_rate = max((net.bytes_recv - self.last_net.bytes_recv) / elapsed, 0)

        self.last_time = now
        self.last_disk = disk_io
        self.last_net = net
        load1, load5, load15 = self._load_average()
        return {
            "node_id": node_id,
            "token": token,
            "cpu_usage_percent": round(cpu, 2),
            "cpu_core_count": psutil.cpu_count(logical=True) or 1,
```

```
                "load_average_1m":
load1,
                "load_average_5m":
load5,
                "load_average_15m":
load15,
                "ram_total_mb":
self._mb(ram.total),
                "ram_used_mb":
self._mb(ram.used),
                "ram_usage_percent":
round(ram.percent, 2),
                "ram_available_mb":
self._mb(ram.available),
                "swap_total_mb":
self._mb(swap.total),
                "swap_used_mb":
self._mb(swap.used),

"swap_usage_percent":
round(swap.percent, 2),
                "disk_total_gb":
self._gb(disk.total),
                "disk_used_gb":
self._gb(disk.used),

"disk_usage_percent":
round(disk.percent, 2),
                "disk_read_bytes":
float(disk_io.read_bytes)  if  disk_io
else 0.0,
                "disk_write_bytes":
float(disk_io.write_bytes)  if  disk_io
else 0.0,
                "disk_read_rate":
round(disk_read_rate, 2),
                "disk_write_rate":
round(disk_write_rate, 2),

"network_bytes_sent":
float(net.bytes_sent) if net else 0.0,

"network_bytes_recv":
float(net.bytes_recv) if net else 0.0,
                "network_sent_rate":
round(network_sent_rate, 2),
                "network_recv_rate":
round(network_recv_rate, 2),
                "process_count":
len(psutil.pids()),
                "uptime_seconds":
round(now - psutil.boot_time(), 2),

"temperature_celsius":
self._temperature(),
                "custom_json":
{"agent_version":       AGENT_VERSION,
"hostname": socket.gethostname()},
            }

        @staticmethod
        def _mb(value: int | float)
-> float:
            return
round(float(value) / 1024 / 1024, 2)

        @staticmethod
        def _gb(value: int | float)
-> float:
            return
round(float(value)  /  1024  /  1024  /
1024, 2)
```

```
        @staticmethod
        def _load_average() -> tuple[float | None, float | None, float | None]:
            try:
                load = psutil.getloadavg()
                return round(load[0], 2), round(load[1], 2), round(load[2], 2)
            except (AttributeError, OSError):
                return None, None, None

        @staticmethod
        def _temperature() -> float | None:
            try:
                temps = psutil.sensors_temperatures(fahrenheit=False)
            except (AttributeError, OSError):
                return None
            readings = [entry.current for entries in temps.values() for entry in entries if entry.current is not None]
            return round(max(readings), 2) if readings else None


    def post_json(session: requests.Session, url: str, payload: dict, retries: int = 3) -> bool:
        for attempt in range(1, retries + 1):
            try:
                response = session.post(url, json=payload, timeout=10)
                response.raise_for_status()
                return True
            except requests.RequestException as exc:
                wait = min(2**attempt, 20)
                LOG.warning("Request failed (%s/%s): %s; retry in %ss", attempt, retries, exc, wait)
                time.sleep(wait)
        return False


    # ...

    def main() -> None:
        args = parse_args()

        logging.basicConfig(level=getattr(logging, args.log_level.upper(), logging.INFO), format="%(asctime)s %(levelname)s %(message)s")
        server = args.server.rstrip("/")
        heartbeat_url = f"{server}/api/heartbeat"
        metrics_url = f"{server}/api/metrics"
        session = requests.Session()
        collector = MetricsCollector()
```

```
        LOG.info("Starting AutoInfraDiag agent for node_id=%s server=%s", args.node_id, server)
        while True:
            heartbeat = {
                "node_id": args.node_id,
                "token": args.token,
                "hostname": socket.gethostname(),
                "os_name": platform.system(),
                "os_version": platform.platform(),
                "agent_version": AGENT_VERSION,
            }
            post_json(session, heartbeat_url, heartbeat)
            payload = collector.collect(args.node_id, args.token)
            ok = post_json(session, metrics_url, payload)
            LOG.info("Metrics sent" if ok else "Metrics delivery failed after retries")
        time.sleep(max(args.interval, 1))


    # app/api/metrics.py
    @router.post("/metrics", response_model=ResourceMetricRead)
    def ingest_agent_metric(payload: AgentMetricsRequest, db: Session = Depends(get_db)):
        node = db.get(models.Node, payload.node_id)
        if not node:
            raise HTTPException(status_code=404, detail="Node not found")
        validate_agent_token(db, payload.node_id, payload.token)
        data = payload.model_dump(exclude={"token"})
        return create_metric(db, data, run_analysis=True)


    @router.post("/heartbeat", response_model=NodeRead)
    def ingest_agent_heartbeat(payload: AgentHeartbeatRequest, db: Session = Depends(get_db)):
        return heartbeat(db, payload.model_dump())


    # app/services/agent_service.py
    def create_token_for_node(db: Session, node_id: int, server_url: str = "http://localhost:8000") -> dict:
        node = db.get(models.Node, node_id)
        if not node:
            raise HTTPException(status_code=404, detail="Node not found")
        token = generate_token()
        record = models.AgentToken(node_id=node_id, token_hash=hash_token(token),
```

```
token_preview=preview_token(token),
is_active=True)
          db.add(record)
          db.commit()
          db.refresh(record)
          command                    =
install_command(server_url,   node_id,
token)
          return     {"node":     node,
"token":    token,    "token_preview":
record.token_preview,
"install_command": command}



      def     validate_agent_token(db:
Session, node_id: int, token: str) ->
models.AgentToken:
          token_hash                 =
hash_token(token)
          record = db.scalar(

select(models.AgentToken).where(

models.AgentToken.node_id == node_id,

models.AgentToken.token_hash        ==
token_hash,

models.AgentToken.is_active.is_(True),
              )
          )
          if not record:
              raise
HTTPException(status_code=status.HTTP_
401_UNAUTHORIZED,      detail="Invalid
agent token")
          record.last_used_at        =
datetime.utcnow()
          return record


      def    heartbeat(db:    Session,
payload: dict) -> models.Node:
          validate_agent_token(db,
payload["node_id"], payload["token"])
          node  =  db.get(models.Node,
payload["node_id"])
          if not node:
              raise
HTTPException(status_code=404,
detail="Node not found")
          node.status = "online"
          node.last_heartbeat_at     =
datetime.utcnow()
          node.hostname              =
payload.get("hostname")            or
node.hostname
          node.os_name               =
payload.get("os_name") or node.os_name
          node.os_version            =
payload.get("os_version")          or
node.os_version
          node.agent_version         =
payload.get("agent_version")       or
node.agent_version
          db.commit()
          db.refresh(node)
          return node


      def  install_command(server_url:
str,  node_id:  int,  token:  str  =
"TOKEN") -> str:
```

```
        server                      =
normalize_server_url(server_url)
    return f"python agent.py --server
{server} --node-id {node_id} --token
{token} --interval 5"


    def create_metric(db: Session,
payload: dict, run_analysis: bool =
True) -> models.ResourceMetric:
        data = dict(payload)
        if                      not
data.get("timestamp"):
            data.pop("timestamp",
None)
        node_id = data["node_id"]
        prev   =   latest_metric(db,
node_id)
        metric                      =
models.ResourceMetric(**data)
        _fill_rates(metric, prev)
        db.add(metric)
        node  =  db.get(models.Node,
node_id)
        if node:
            node.status           =
_status_for_metric(metric)
            node.last_heartbeat_at =
datetime.utcnow()
        db.commit()
        db.refresh(metric)

        if run_analysis:
            from
app.services.diagnosis_engine   import
run_diagnostics_for_node
run_diagnostics_for_node(db, node_id)
        return metric


    def          _fill_rates(metric:
models.ResourceMetric,          prev:
models.ResourceMetric | None) -> None:
        if not prev:
            metric.disk_read_rate  =
metric.disk_read_rate or 0
            metric.disk_write_rate =
metric.disk_write_rate or 0
            metric.network_sent_rate
= metric.network_sent_rate or 0
            metric.network_recv_rate
= metric.network_recv_rate or 0
            return
        now  =  metric.timestamp  or
datetime.utcnow()
        prev_ts = prev.timestamp or
now
        elapsed    =    max((now    -
prev_ts).total_seconds(), 1)
        pairs = [
            ("disk_read_rate",
"disk_read_bytes"),
            ("disk_write_rate",
"disk_write_bytes"),
            ("network_sent_rate",
"network_bytes_sent"),
            ("network_recv_rate",
"network_bytes_recv"),
        ]
        for rate_attr, total_attr in
pairs:
```

```
            if       getattr(metric,
rate_attr) is None:
                current            =
getattr(metric, total_attr) or 0
                previous           =
getattr(prev, total_attr) or 0
                setattr(metric,
rate_attr, max((current - previous) /
elapsed, 0))



     def   _status_for_metric(metric:
models.ResourceMetric) -> str:
         risk = max(
             metric.cpu_usage_percent
or 0,
             metric.ram_usage_percent
or 0,

metric.swap_usage_percent or 0,

metric.disk_usage_percent or 0,
             90                    if
(metric.temperature_celsius or 0) > 85
else 0,
         )
    return status_from_risk(risk)


     def run_diagnostics_for_node(db:
Session,     node_id:     int)     ->
models.Diagnostic:
         node  =  db.get(models.Node,
node_id)
         metrics = list(
             reversed(
                 db.scalars(
select(models.ResourceMetric)

.where(models.ResourceMetric.node_id
== node_id)

.order_by(desc(models.ResourceMetric.t
imestamp),
desc(models.ResourceMetric.id))
                     .limit(30)
                 ).all()
             )
         )
         diagnostic_data            =
_diagnose(node, metrics, db)
         diagnostic                 =
models.Diagnostic(node_id=node_id,
**diagnostic_data)
         db.add(diagnostic)
         if node:
             node.status            =
status_from_risk(diagnostic_data["risk
_score"])
         db.commit()
         db.refresh(diagnostic)

         if diagnostic.diagnosis_type
== "healthy":
             from
app.services.incident_service   import
resolve_open_incidents_for_node


resolve_open_incidents_for_node(db,
node_id)
         else:
```

```
            from
app.services.incident_service   import
create_incident_from_diagnostic
            from
app.services.recommendation_engine
import
create_recommendation_for_diagnostic



create_incident_from_diagnostic(db,
diagnostic)

create_recommendation_for_diagnostic(d
b, diagnostic)
          return diagnostic


      def  _diagnose(node:  models.Node
|            None,            metrics:
list[models.ResourceMetric],       db:
Session) -> dict:
          if not metrics:
              return
_result("unknown_degradation",
"Недостатньо даних", 10, 20, "Для вузла
ще немає resource metrics.", [])

          latest = metrics[-1]
          avg_cpu    =    _avg(metrics,
"cpu_usage_percent")
          avg_ram    =    _avg(metrics,
"ram_usage_percent")
          avg_swap   =   _avg(metrics,
"swap_usage_percent")
          anomaly                    =
detect_anomaly(metrics)
          evidence: list[dict] = []
          overcommit                 =
_overcommit_evidence(node, db)
          if overcommit:

evidence.extend(overcommit)
              return _result(

"resource_overcommit",
                  "Resource
overcommit",
                  82,
                  88,
                  "Сума      виділених
ресурсів перевищує доступні ліміти або
ліміти конкретного вузла.",
                  evidence,
                  anomaly,
              )

          if
(latest.temperature_celsius or 0) > 80:

evidence.append(_ev("temperature_celsi
us",  latest.temperature_celsius,  ">
80", "Температура перевищує безпечний
поріг."))
              return
_result("thermal_risk",       "Thermal
risk",
clamp((latest.temperature_celsius   or
80) + 5), 92, "Виявлено ризик перегріву
або  thermal  throttling.",  evidence,
anomaly)

          if (latest.ram_usage_percent
or       0)       >       85       and
(latest.swap_usage_percent or 0) > 20:
```

```
            evidence.extend(
                [
                    _ev("ram_usage_percent", latest.ram_usage_percent, "> 85", "RAM майже вичерпана."),
                    _ev("swap_usage_percent", latest.swap_usage_percent, "> 20", "Система активно використовує swap."),
                ]
            )
            return _result("swap_pressure", "Swap pressure", max(latest.ram_usage_percent or 0, 78), 90, "Вузол компенсує нестачу RAM через swap, що збільшує затримки.", evidence, anomaly)

        if (latest.ram_usage_percent or 0) > 90 and (latest.ram_available_mb or 0) < max((latest.ram_total_mb or 4096) * 0.12, 512):
            evidence.extend(
                [
                    _ev("ram_usage_percent", latest.ram_usage_percent, "> 90", "Використання RAM перевищує критичний поріг."),
                    _ev("ram_available_mb", latest.ram_available_mb, "< 12% total або < 512 MB", "Доступної пам'яті замало для стабільної роботи."),
                ]
            )
            return _result("memory_pressure", "Memory pressure", latest.ram_usage_percent or 85, 91, "Оперативної пам'яті недостатньо для поточного workload.", evidence, anomaly)

        if (latest.cpu_usage_percent or 0) > 85:
            evidence.extend(
                [
                    _ev("cpu_usage_percent", latest.cpu_usage_percent, "> 85", "CPU використовується майже повністю."),
                    _ev("load_average_1m", latest.load_average_1m, "informational", "Load average використовується як додатковий доказ, якщо доступний."),
                ]
            )
            return _result("cpu_saturation", "CPU saturation", latest.cpu_usage_percent or 85, 92, "Процесорні ресурси є першопричиною деградації.", evidence, anomaly)

        disk_rate = max(latest.disk_read_rate or 0, latest.disk_write_rate or 0)
        if disk_rate > DISK_RATE_HIGH and (latest.cpu_usage_percent or 0) < 80:
```

```
            evidence.extend(
                [

_ev("disk_read_rate",
latest.disk_read_rate,            f">
{DISK_RATE_HIGH}", "Висока швидкість
читання з диска."),

_ev("disk_write_rate",
latest.disk_write_rate,           f">
{DISK_RATE_HIGH}", "Висока швидкість
запису на диск."),

_ev("cpu_usage_percent",
latest.cpu_usage_percent, "< 80", "CPU
не є головним bottleneck."),
                ]
            )
            return
_result("disk_io_bottleneck",    "Disk
I/O bottleneck", 76, 86, "Обмеження
продуктивності спричинене дисковими
операціями.", evidence, anomaly)

        net_rate                  =
max(latest.network_sent_rate   or   0,
latest.network_recv_rate or 0)
        if        net_rate        >
NETWORK_RATE_HIGH                 and
(latest.cpu_usage_percent or 0) < 80
and (latest.ram_usage_percent or 0) <
85:
            evidence.extend(
                [

_ev("network_sent_rate",
latest.network_sent_rate,         f">
{NETWORK_RATE_HIGH}",          "Висока
швидкість вихідного трафіку."),

_ev("network_recv_rate",
latest.network_recv_rate,         f">
{NETWORK_RATE_HIGH}",          "Висока
швидкість вхідного трафіку."),

_ev("cpu_usage_percent",
latest.cpu_usage_percent, "< 80", "CPU
не є першопричиною."),
                ]
            )
            return
_result("network_pressure",   "Network
pressure",     72,     84,     "Основне
навантаження припадає на мережевий
throughput.", evidence, anomaly)

        sustained_window = metrics[-
8:] if len(metrics) >= 8 else metrics
        if    _avg(sustained_window,
"cpu_usage_percent")    <    15    and
_avg(sustained_window,
"ram_usage_percent") < 30:
            evidence.extend(
                [

_ev("cpu_usage_percent_avg",
round(avg_cpu,   2),   "<   15",   "CPU
тривалий      час      майже      не
використовується."),

_ev("ram_usage_percent_avg",
round(avg_ram,   2),   "<   30",   "RAM
allocation           використовується
частково."),
```

```
                ]
            )
            return
_result("underutilization",
"Underutilization", 28, 85, "Вузол має
надлишково виділені ресурси.",
evidence, anomaly)

        if
(latest.service_latency_ms or 0) > 1000
or (latest.service_error_rate or 0) >
5:
            evidence.extend(
                [

_ev("service_latency_ms",
latest.service_latency_ms, "> 1000",
"Сервіс відповідає повільно."),

_ev("service_error_rate",
latest.service_error_rate, "> 5",
"Підвищена частка помилок."),
                ]
            )
            return
_result("service_degradation",
"Service degradation", 68, 70,
"Сервісні показники деградують, але
ресурсна першопричина не
підтверджена.", evidence, anomaly)

        if anomaly.get("is_anomaly")
and anomaly.get("anomaly_score", 0) >
60:

evidence.append(_ev("anomaly_score",
anomaly["anomaly_score"], "> 60",
anomaly.get("comment", "Виявлена
аномалія.")))
            return
_result("unknown_degradation",
"Unknown degradation",
anomaly["anomaly_score"], 58, "Метрики
виглядають нетипово, але конкретний
bottleneck не підтверджено.",
evidence, anomaly)

        evidence.extend(
            [

_ev("cpu_usage_percent",
latest.cpu_usage_percent, "<= 85",
"CPU у межах допустимого."),

_ev("ram_usage_percent",
latest.ram_usage_percent, "<= 90",
"RAM у межах допустимого."),

_ev("swap_pressure_condition", f"RAM
{latest.ram_usage_percent}%, swap
{latest.swap_usage_percent}%", "RAM >
85% і swap > 20%", "Swap pressure не
підтверджено, бо умова RAM+swap не
виконана."),
            ]
        )
        return _result("healthy",
"No critical degradation", 12, 78,
"Критичних ознак деградації ресурсів не
виявлено.", evidence, anomaly)

    def _ev(metric_name: str,
current_value: object, threshold:
object, explanation: str) -> dict:
```

```
        return {"metric": metric_name, "current_value": current_value, "threshold": threshold, "explanation": explanation}


    def _result(dtype: str, root: str, risk: float, confidence: float, explanation: str, evidence: list[dict], anomaly: dict | None = None) -> dict:
        risk_score = clamp(float(risk))
        return {
            "diagnosis_type": dtype,
            "root_cause": root,
            "severity": severity_from_risk(risk_score),
            "confidence": clamp(confidence),
            "risk_score": risk_score,
            "explanation": f"{explanation} {anomaly.get('comment') if anomaly else ''}".strip(),
            "evidence_json": {"evidence": evidence, "anomaly": anomaly or {}},
            "status": "open",
        }


    def _overcommit_evidence(node: models.Node | None, db: Session) -> list[dict]:
        if not node:
            return []
        evidence: list[dict] = []
        if node.allocated_cpu_cores and node.max_cpu_cores and node.allocated_cpu_cores > node.max_cpu_cores:
            evidence.append(_ev("allocated_cpu_cores", node.allocated_cpu_cores, f"<= {node.max_cpu_cores}", "CPU allocation перевищує ліміт вузла."))
        if node.allocated_ram_mb and node.max_ram_mb and node.allocated_ram_mb > node.max_ram_mb:
            evidence.append(_ev("allocated_ram_mb", node.allocated_ram_mb, f"<= {node.max_ram_mb}", "RAM allocation перевищує ліміт вузла."))
        if node.environment_id:
            nodes = db.scalars(select(models.Node).where(models.Node.environment_id == node.environment_id)).all()
            total_cpu = sum(n.allocated_cpu_cores or 0 for n in nodes)
            total_cpu_limit = sum(n.max_cpu_cores or n.allocated_cpu_cores or 0 for n in nodes)
            total_ram = sum(n.allocated_ram_mb or 0 for n in nodes)
```

```
            total_ram_limit        =
sum(n.max_ram_mb or n.allocated_ram_mb
or 0 for n in nodes)
            if  total_cpu_limit  and
total_cpu > total_cpu_limit:

evidence.append(_ev("environment_alloc
ated_cpu",       total_cpu,       f"<=
{total_cpu_limit}",   "Сумарний   CPU
allocation    середовища    перевищує
доступний CPU limit."))
            if  total_ram_limit  and
total_ram > total_ram_limit:

evidence.append(_ev("environment_alloc
ated_ram_mb",     total_ram,     f"<=
{total_ram_limit}",   "Сумарний   RAM
allocation    середовища    перевищує
доступний RAM limit."))
    return evidence


     def
create_recommendation_for_diagnostic(d
b:        Session,        diagnostic:
models.Diagnostic)                  ->
models.Recommendation:
         node  =  db.get(models.Node,
diagnostic.node_id)
         latest = db.scalar(

select(models.ResourceMetric)

.where(models.ResourceMetric.node_id
== diagnostic.node_id)

.order_by(desc(models.ResourceMetric.t
imestamp),
desc(models.ResourceMetric.id))
             .limit(1)
         )
         current_cpu                =
node.allocated_cpu_cores if node else
None
         current_ram                =
node.allocated_ram_mb if node else None
         spec                       =
_recommendation_spec(diagnostic, node,
latest)
         _apply_role_context(spec,
node)
         recommendation             =
models.Recommendation(

node_id=diagnostic.node_id,

diagnostic_id=diagnostic.id,

recommendation_type=spec["type"],
             title=spec["title"],

description=spec["description"],

priority=diagnostic.severity,

current_cpu_cores=current_cpu,

recommended_cpu_cores=spec.get("recomm
ended_cpu"),

current_ram_mb=current_ram,

recommended_ram_mb=spec.get("recommend
ed_ram"),
```

```
expected_effect=spec["expected_effect"],
                reason=spec["reason"],
                action_steps_json=spec["actions"],
                status="new",
            )
            db.add(recommendation)
            db.commit()
            db.refresh(recommendation)
            return recommendation


        def _double(value: int | None, fallback: int) -> int:
            return max((value or fallback) * 2, fallback)


        def _recommendation_spec(diagnostic: models.Diagnostic, node: models.Node | None, latest: models.ResourceMetric | None) -> dict:
            cpu = node.allocated_cpu_cores if node and node.allocated_cpu_cores else (latest.cpu_core_count if latest else 2)
            ram = node.allocated_ram_mb if node and node.allocated_ram_mb else (int(latest.ram_total_mb or 4096) if latest else 4096)
            dtype = diagnostic.diagnosis_type
            if dtype == "cpu_saturation":
                rec_cpu = min(_double(cpu, 2), node.max_cpu_cores if node and node.max_cpu_cores else max(_double(cpu, 2), 4))
                return {
                    "type": "vertical_cpu_scaling",
                    "title": "Рекомендовано збільшити CPU ресурси",
                    "description": "На вузлі виявлено стабільно високе використання CPU та підвищений load average.",
                    "recommended_cpu": rec_cpu,
                    "recommended_ram": ram,
                    "reason": "CPU є основним bottleneck, тому додаткові ядра або application instances мають зменшити чергу виконання.",
                    "expected_effect": "Зниження CPU utilization, load average і затримок сервісів під піковим навантаженням.",
                    "actions": [
                        f"Збільшити CPU cores: {cpu} -> {rec_cpu}.",
                        "Перевірити процеси з найбільшим CPU.",
                        "Розглянути горизонтальне масштабування application instances.",
```

```
                    "Перевірити політики балансування навантаження.",
                ],
            }

        if dtype in ("memory_pressure", "swap_pressure"):
            rec_ram = min(_double(ram, 4096), node.max_ram_mb if node and node.max_ram_mb else max(_double(ram, 4096), 8192))
            return {
                "type": "vertical_ram_scaling",
                "title": "Рекомендовано збільшити RAM ресурси",
                "description": "RAM usage перевищує безпечний поріг або система активно використовує swap.",
                "recommended_cpu": cpu,
                "recommended_ram": rec_ram,
                "reason": "Високе використання пам'яті є підтвердженою першопричиною деградації.",
                "expected_effect": "Менше swap activity, стабільніший response time і нижчий ризик OOM.",
                "actions": [
                    f"Збільшити RAM: {round(ram / 1024, 1)} GB -> {round(rec_ram / 1024, 1)} GB.",
                    "Перевірити memory leaks у сервісах.",
                    "Оптимізувати процеси з найбільшим memory footprint.",
                ],
            }

        if dtype == "resource_overcommit":
            return {
                "type": "overcommit_optimization",
                "title": "Рекомендовано зменшити overcommit ресурсів",
                "description": "Віртуальним вузлам виділено більше CPU/RAM, ніж реально доступно у межах заданих лімітів.",
                "recommended_cpu": cpu,
                "recommended_ram": ram,
                "reason": "Overcommit може призвести до нестабільної продуктивності при одночасному піковому навантаженні.",
                "expected_effect": "Менший ризик contention між VM і прогнозованіша продуктивність.",
                "actions": [
                    "Зменшити overcommit ratio.",
                    "Перенести VM на інший host.",
                    "Додати host resources.",
```

```
                    "Переглянути CPU/RAM quotas для віртуальних вузлів.",
                ],
            }

        return {
            "type": "no_scaling_recommended",
            "title": "Масштабування наразі не рекомендоване",
            "description": "Система не має достатньо доказів, що CPU/RAM scaling вирішить проблему.",
            "recommended_cpu": cpu,
            "recommended_ram": ram,
            "reason": "Першопричина не підтверджена або метрик недостатньо.",
            "expected_effect": "Уникнення необґрунтованих змін ресурсів.",
            "actions": ["Зібрати більше метрик за довший період.", "Перевірити журнали сервісів."],
        }
    # app/services/anomaly_detector.py
    from __future__ import annotations

    import numpy as np

    from app import models


    FEATURES = [
        "cpu_usage_percent",
        "ram_usage_percent",
        "swap_usage_percent",
        "disk_usage_percent",
        "disk_read_rate",
        "disk_write_rate",
        "network_sent_rate",
        "network_recv_rate",
        "temperature_celsius",
    ]


    def detect_anomaly(metrics: list[models.ResourceMetric]) -> dict:
        if len(metrics) < 12:
            return _rule_based(metrics)
        rows = []
        for metric in metrics:
            rows.append([float(getattr(metric, name) or 0) for name in FEATURES])
        data = np.asarray(rows, dtype=float)
        try:
            from sklearn.ensemble import IsolationForest

            model = IsolationForest(contamination=0.15, random_state=42)
            model.fit(data)
            scores = model.decision_function(data)
            latest_score = float(scores[-1])
```

```
            anomaly_score =
round(max(0.0, min(100.0, (0.15 -
latest_score) * 300)), 2)
            return {
                "mode":
"isolation_forest",
                "is_anomaly":
bool(model.predict(data[-1:])[0] == -
1),
                "anomaly_score":
anomaly_score,
                "comment":
"IsolationForest позначив останній
стан як нетиповий." if anomaly_score >
55 else "Суттєвої ML-аномалії не
виявлено.",
            }
        except Exception:
            return
_rule_based(metrics)



    def _rule_based(metrics:
list[models.ResourceMetric]) -> dict:
        latest = metrics[-1] if
metrics else None
        if not latest:
            return {"mode":
"rule_based", "is_anomaly": False,
"anomaly_score": 0, "comment": "Немає
достатньо метрик для аналізу."}
        score = max(
            latest.cpu_usage_percent
or 0,
            latest.ram_usage_percent
or 0,
            latest.swap_usage_percent or 0,
            latest.disk_usage_percent or 0,
            90 if
(latest.temperature_celsius or 0) > 80
else 0,
        )
        return {
            "mode": "rule_based",
            "is_anomaly": score >
85,
            "anomaly_score":
round(float(score), 2),
            "comment": "Використано
rule-based fallback через малу
кількість метрик.",
        }

    #
app/services/capacity_planner.py
    from datetime import timedelta

    from sqlalchemy import desc,
select
    from sqlalchemy.orm import
Session

    from app import models
    from app.utils.time_utils import
utc_now


    TARGETS = {
        "cpu_usage_percent": 85,
        "ram_usage_percent": 85,
        "disk_usage_percent": 90,
```

```
    }


    def run_capacity_planning(db: Session, node_id: int) -> list[models.CapacityForecast]:
        metrics = list(
            reversed(
                db.scalars(
                    select(models.ResourceMetric)
                    .where(models.ResourceMetric.node_id == node_id)
                    .order_by(desc(models.ResourceMetric.timestamp), desc(models.ResourceMetric.id))
                    .limit(80)
                ).all()
            )
        )
        old = db.scalars(select(models.CapacityForecast).where(models.CapacityForecast.node_id == node_id)).all()
        for row in old:
            db.delete(row)
        forecasts = [_forecast_metric(node_id, metrics, name, threshold) for name, threshold in TARGETS.items()]
        if metrics and _avg(metrics[-10:], "cpu_usage_percent") < 15 and _avg(metrics[-10:], "ram_usage_percent") < 30:
            latest = metrics[-1]
            forecasts.append(
                models.CapacityForecast(
                    node_id=node_id,
                    metric_name="underutilization",
                    current_value=max(latest.cpu_usage_percent or 0, latest.ram_usage_percent or 0),
                    predicted_value=None,
                    predicted_threshold_time=None,
                    trend_direction="low_usage",
                    recommendation="Вузол використовує дуже малу частку CPU/RAM. Рекомендовано розглянути downscale або consolidation.",
                )
            )
        for forecast in forecasts:
            db.add(forecast)
        db.commit()
        for forecast in forecasts:
            db.refresh(forecast)
        return forecasts


    def _forecast_metric(node_id: int, metrics: list[models.ResourceMetric], metric_name: str, threshold: float) -> models.CapacityForecast:
```

```
        if len(metrics) < 2:
            return
models.CapacityForecast(
                node_id=node_id,

metric_name=metric_name,

current_value=getattr(metrics[-1],
metric_name, None) if metrics else
None,

predicted_value=None,

predicted_threshold_time=None,

trend_direction="insufficient_data",

recommendation="Недостатньо метрик для
capacity forecast.",
            )
        values                      =
[float(getattr(metric, metric_name) or
0) for metric in metrics]
        current = values[-1]
        slope   =   (values[-1]   -
values[0]) / max(len(values) - 1, 1)
        predicted = max(0, min(100,
current + slope * 12))
        if slope > 0.25:
            trend = "up"
        elif slope < -0.25:
            trend = "down"
        else:
            trend = "stable"

        threshold_time = None
        if slope > 0 and current <
threshold:
            points_to_threshold    =
(threshold - current) / slope
            minutes                =
max(points_to_threshold * 5, 1)
            threshold_time         =
utc_now() + timedelta(minutes=minutes)
            recommendation         =
f"{_label(metric_name)} має висхідний
тренд. Якщо тенденція збережеться,
поріг {threshold}% може бути досягнутий
приблизно через {round(minutes)}
хвилин."
        elif current >= threshold:
            threshold_time         =
utc_now()
            recommendation         =
f"{_label(metric_name)} вже перевищує
поріг      {threshold}%.      Потрібна
діагностика та план масштабування."
        elif trend == "down":
            recommendation         =
f"{_label(metric_name)}   має   спадний
тренд. Додаткове масштабування наразі
не є пріоритетним."
        else:
            recommendation         =
f"{_label(metric_name)}     стабільна.
Продовжуйте   збирати   метрики   для
точнішого прогнозу."

        return
models.CapacityForecast(
            node_id=node_id,
            metric_name=metric_name,
```

```
current_value=round(current, 2),
predicted_value=round(predicted, 2),
predicted_threshold_time=threshold_time,
            trend_direction=trend,
recommendation=recommendation,
        )

    def _avg(metrics: list[models.ResourceMetric], attr: str) -> float:
        if not metrics:
            return 0.0
        return sum(float(getattr(metric, attr) or 0) for metric in metrics) / len(metrics)

    def _label(metric_name: str) -> str:
        return {
            "cpu_usage_percent": "CPU usage",
            "ram_usage_percent": "RAM usage",
            "disk_usage_percent": "Disk usage",
        }.get(metric_name, metric_name)
```

Протокол аналізу звіту подібності експертом

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

Автор: Андрій СТЕПАНЮК

Співавтор:

Назва: Система діагностики та масштабування віртуалізованих обчислювальних ресурсів у комп'ютерних мережах

Експерт: Володимир ГРИГА

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:1.76%

Коефіцієнт подібності 2:0.17%

Мікропробіли: 3

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-06-10 15:00:58.0

Після аналізу Звіту подібності констатую наступне:

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

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

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

Обгрунтування:

2026-06-10 Доцент Андрій Нічепорук

Дата експерт

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 7%**

ID: 274473 Назва: БКР Система діагностики та масштабування віртуалізованих обчислювальних ресурсів у комп'ютерних мережах Додано в БД: 2026-06-09 Автора: Андрій СТЕПАНЮК Керівники: Володимир ГРИГА Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	122381	1043	2227 (2%)	32 (3%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Степанюк Андрій Ігорович

Тема: Система діагностики та масштабування віртуалізованих обчислювальних ресурсів у комп'ютерних мережах.

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___74___

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є розробка системи діагностики та формування рекомендацій щодо масштабування віртуалізованих обчислювальних ресурсів у комп'ютерних мережах.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню та вимогам до кваліфікаційної роботи.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі кваліфікаційної роботи проведено дослідження предметної області: проаналізовано особливості віртуалізованих обчислювальних ресурсів, принципи моніторингу та діагностики серверної інфраструктури, виконано порівняльний аналіз наявних систем спостереження та сформульовано постановку задачі дослідження. В другому розділі кваліфікаційної роботи проведено проєктування системи діагностики та формування рекомендацій щодо масштабування, а саме: розроблено загальну архітектуру системи; обґрунтовано вибір технологій реалізації; розроблено структуру телеметричних даних та формат обміну повідомленнями; спроєктовано алгоритм діагностики та механізм формування рекомендацій; розроблено модель роботи системи та забезпечено її надійність. В третьому розділі кваліфікаційної роботи виконано програмну реалізацію та перевірку роботи системи діагностики, а саме: реалізовано агентську підсистему збору телеметрії; реалізовано серверну частину; реалізовано механізм діагностики та формування рекомендацій; реалізовано панель спостереження; виконано тестування системи та оцінку ефективності запропонованого рішення.

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: недостатня увага аналізу предметної області.

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

7. Відгук про роботу в цілому: Робота виконана на належному технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: добре (B / 86)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Стецюк Микола Васильович, PhD, ст. викладач
к-ф. Кібербезпеки

"10" червня 2026 р. _____(підпис)

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Андрій СТЕПАНЮК
ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи КІ2с-23-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

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

1 травня 2026 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

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

Назва кваліфікаційної роботи Система діагностики та масштабування віртуалізованих обчислювальних ресурсів у комп'ютерних мережах.
Автор Андрій СТЕПАНЮК
Освітня програма Комп'ютерна інженерія та програмування
Рівень вищої освіти перший (бакалаврський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: к.т.н., доцент Володимир ГРИГА

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

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

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;

3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

4) значна частина знайденого плагіату відноситься до списку використаних джерел

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 1,76%; та системою Anti-Plagiarism складає 1,0%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

01.06.2026

Завідувач кафедри — Підпис — Ольга ПАВЛОВА (Ім'я, ПРІЗВИЩЕ)

Гарант освітньої програми — Підпис — Андрій НІЧЕПОРУК (Ім'я, ПРІЗВИЩЕ)

Керівник кваліфікаційної роботи — Підпис — Володимир ГРИГА (Ім'я, ПРІЗВИЩЕ)