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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «ПРОГРАМНО-АПАРАТНИЙ КОМПЛЕКС ДЛЯ СПОСТЕРЕЖЕННЯ ЗА ПРАЦЕЗДАТНІСТЮ СЕРВЕРІВ»

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

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

_____ *Іван ШАПОВАЛОВ*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КІД-41
Іван ШАПОВАЛОВ
Ім'я, ПРІЗВИЩЕ
Керівник: Віра БАБЕНКО
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент:
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут Інформаційних технологій

Кафедра 123 Комп'ютерної інженерії
Ступінь вищої освіти Бакалавр
Спеціальність 123 Комп'ютерної інженерії
Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедрою Наталія ЛАЩЕВСЬКА
Ім'я, ПРІЗВИЩЕ
«_____» _____ 2026р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Шаповалов Іван Олексійович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Програмно-апаратний комплекс для спостереження за працездатністю серверів
керівник кваліфікаційної роботи Віра БАБЕНКО,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» 02 2026р. № 51
2. Строк подання кваліфікаційної роботи «08» 05 2026р.
3. Вихідні дані до кваліфікаційної роботи: Науково-технічна література, методи системного аналізу, мережеві протоколи передачі даних, математичний апарат, емпіричні методи
Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Детальний аналіз та огляд досліджуваної сфери
2. Архітектурне проєктування та практична розробка комплексної системи моніторингу технічного стану серверної інфраструктури
3. Практична реалізація та програмно-апаратна розробка комплексу моніторингу технічного стану серверів
4. Перелік ілюстративного матеріалу: *презентація*
5. Дата видачі завдання «09» 02 2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	25.02-24.03.2026	Виконано
2	Обробка матеріалу	23.03-05.04.2026	Виконано
3	Розробка теоретичної частини	06.04-10.04.2026	Виконано
4	Розробка практичної частини	11.04-20.04.2026	Виконано
5	Висновки за результатами аналізу	21.04-06.05.2026	Виконано
6	Оформлення пояснювальної записки	07.05-08.05.2026	Виконано
7	Розробка презентації	09.05-11.05.2026	Виконано
8	Передзахист	27.05-29.05.2026	Виконано
9	Здача в деканат	02.06-05.06.2026	Виконано

Здобувач вищої освіти _____ Іван ШАПОВАЛОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Віра БАБЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 60 стор., 27 рис., 26 джерел.

Мета дослідження – розробка програмно-апаратного комплексу для безперервного спостереження за працездатністю та технічним станом серверного обладнання.

Об'єкт дослідження – процес моніторингу технічного стану та продуктивності серверної інфраструктури.

Предмет дослідження – апаратні та програмні засоби збору, аналізу та візуалізації телеметричних даних серверів.

Короткий зміст роботи: Сучасні корпоративні системи та вебдодатки критично залежать від безперебійної роботи серверного обладнання. Будь-який простій (downtime) або деградація продуктивності серверів призводять до зупинки бізнес-процесів та фінансових втрат. Тому постає гостре питання своєчасного виявлення апаратних чи програмних проблем ще до того, як вони спричинять критичний збій. Це досягається шляхом впровадження систем безперервного моніторингу.

Розроблений програмно-апаратний комплекс забезпечує автоматизований збір метрик (завантаження процесора, використання оперативної пам'яті, стан дискової підсистеми, температурні показники) та їх аналіз у режимі реального часу. У роботі досліджено принципи побудови архітектури таких систем, вибрано оптимальні протоколи передачі даних (SNMP, SSH, ICMP) та реалізовано механізми миттєвого сповіщення адміністраторів про нештатні ситуації. На основі проведеного дослідження доведено ефективність використання комплексного підходу (поєднання апаратних датчиків та програмних агентів) для забезпечення високої відмовостійкості серверної інфраструктури.

КЛЮЧОВІ СЛОВА: МОНІТОРИНГ СЕРВЕРІВ, ПРОГРАМНО-АПАРАТНИЙ КОМПЛЕКС, ТЕЛЕМЕТРІЯ, ВІДМОВОСТІЙКІСТЬ, ІНФРАСТРУКТУРА, ZABBIX, PROMETHEUS, SNMP, АЛЕРТИНГ.

ABSTRACT

Textual part of the qualification work for the bachelor's degree: 60 pages, 27 figures, 26 sources.

The aim of the work is the development of a hardware-software complex for continuous monitoring of the performance and technical condition of server equipment.

Object of research – The process of monitoring the technical condition and performance of server infrastructure.

Subject of research – Hardware and software tools for collecting, analyzing, and visualizing server telemetry data.

Summary of the work: Modern corporate systems and web applications critically depend on the uninterrupted operation of server equipment. Any downtime or performance degradation of servers leads to the halt of business processes and financial losses. Therefore, there is an urgent need for timely detection of hardware or software problems before they cause a critical failure. This is achieved by implementing continuous monitoring systems.

The developed hardware-software complex provides automated collection of metrics (CPU load, RAM usage, disk subsystem status, temperature indicators) and their analysis in real-time. The paper explores the principles of building the architecture of such systems, selects optimal data transfer protocols (SNMP, SSH, ICMP), and implements mechanisms for instant notification of administrators about abnormal situations. Based on the conducted research, the effectiveness of using a comprehensive approach (combining hardware sensors and software agents) to ensure high fault tolerance of the server infrastructure is proven.

KEYWORDS: SERVER MONITORING, HARDWARE-SOFTWARE COMPLEX, TELEMETRY, FAULT TOLERANCE, INFRASTRUCTURE, ZABBIX, PROMETHEUS, SNMP, ALERTING..

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

ПОДАННЯ
ГОЛОВІ ДЕРЖАВНОЇ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня бакалавра

Направляється здобувач Шаповалов І. О. до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю 123 Комп'ютерна інженерія
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)
на тему: «Програмно-апаратний комплекс для спостереження за працездатністю серверів».
Кваліфікаційна робота і рецензія додаються.

Директор ННІ _____ Катерина НЕСТЕРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач Шаповалов І. О. показав відмінну теоретичну підготовку, уміння володіти та розуміння новими комп'ютерними технологіями, користуватися навчальною і науково-технічною літературою. Працюючи над завданнями, які були поставлені керівником, проявив ініціативність, сумлінність та хист до інженерної роботи.
Все це дозволяє оцінити кваліфікаційну роботу студента Шаповалова Івана Олексійовича на оцінку «відмінно», та присвоїти їй кваліфікацію «бакалавр з комп'ютерної інженерії».

Керівник кваліфікаційної роботи _____ Віра БАБЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

«__» _____ 2026 року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач *Шаповалов І. О.* допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедрою Комп'ютерної інженерії _____ Наталія ЛАЩЕВСЬКА
(назва) *(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну бакалаврську роботу

здобувача вищої освіти Шаповалов Іван Олексійович
(прізвище, ім'я, по батькові)
на тему: «Програмно-апаратний комплекс для спостереження за працездатністю серверів»

Актуальність:

Бакалаврська кваліфікаційна робота присвячена вирішенню надзвичайно важливого практичного завдання — забезпеченню відмовостійкості IT-інфраструктури шляхом безперервного контролю технічного стану серверного обладнання. В умовах тотальної цифровізації бізнесу та переходу до хмарних технологій, ціна простою серверів зростає експоненційно, що робить тему розробки комплексних систем моніторингу високоактуальною. Автор пропонує ефективне поєднання програмних та апаратних засобів для створення цілісної картини працездатності систем..

Позитивні сторони:
1. У роботі представлено ґрунтовний аналіз існуючих рішень на ринку, на основі якого зроблено логічний та обґрунтований вибір технологічного стеку для власної розробки.
2. Автор продемонстрував високий рівень інженерної підготовки, вдало реалізувавши архітектуру збору даних. Особливо варто відзначити розробку модуля сповіщень, який гнучко налаштовується під різні критичні пороги метрик
3. Практична цінність роботи підтверджується успішним тестуванням розробленого комплексу на реальних серверних вузлах.

Недоліки:
1. У роботі дещо недостатньо уваги приділено питанням безпеки самого процесу моніторингу. Передача телеметрії та віддалене виконання діагностичних команд вимагають суворого шифрування та контролю доступу, щоб агенти моніторингу не стали вразливістю для хакерських атак.
2. Бажано було б розширити блок тестування комплексу в умовах високого мережевого навантаження або штучного "шторму подій", щоб перевірити, наскільки стабільно центральний сервер обробляє величезний потік вхідних метрик від великої кількості клієнтів одночасно.

Висновок: *кваліфікаційна бакалаврська робота заслуговує оцінку " відмінно", а здобувач Шаповалов І. О. заслуговує присвоєння кваліфікації: бакалавра з комп'ютерної інженерії.*

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

ЗМІСТ

ВСТУП.....10
1 ДЕТАЛЬНИЙ АНАЛІЗ ТА ОГЛЯД ДОСЛІДЖУВАНОЇ СФЕРИ.....11
1.1 Формування концептуальних засад відстеження технічних параметрів серверного обладнання.....11
1.2 Аналіз та порівняльна характеристика сучасних інструментів та платформ для контролю стану серверів.....14
1.3 Детальний огляд архітектури та функціональних можливостей системи Nagios.....16
1.4 Детальний розгляд екосистеми моніторингу Prometheus.....20
2 АРХІТЕКТУРНЕ ПРОЄКТУВАННЯ ТА ПРАКТИЧНА РОЗРОБКА КОМПЛЕКСНОЇ СИСТЕМИ МОНІТОРИНГУ ТЕХНІЧНОГО СТАНУ СЕРВЕРНОЇ ІНФРАСТРУКТУРИ.....23
2.1 Формування комплексу функціональних та нефункціональних вимог до проєктованої системи моніторингу.....23
2.2 Структурна організація та базова архітектура розроблюваної системи моніторингу.....27
2.3 Обґрунтування вибору оптимальної архітектури та сучасного стеку технологій для розробки комплексної системи моніторингу.....33
3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ПРОГРАМНО-аПАРАТНА РОЗРОБКА КОМПЛЕКСУ МОНІТОРИНГУ ТЕХНІЧНОГО СТАНУ СЕРВЕРІВ.....39
3.1 Проєктування та фізична реалізація структури бази даних системи моніторингу.....39
3.2 Проєктування, програмна реалізація та інтеграція інтерактивного Telegram-бота як головного інтерфейсу користувача.....44
3.3 Комплексне тестування та верифікація працездатності розробленої системи 49

3.4 Проєктування та інтеграція апаратного модуля моніторингу на базі одноплатного мікрокомп'ютера Raspberry Pi..... 52
ВИСНОВКИ..... 56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 57
Додаток А 60

ВСТУП

У наш час інформаційні технології та серверні потужності є фундаментом діяльності практично будь-якої сучасної організації. Серверна інфраструктура забезпечує роботу баз даних, корпоративних порталів, хмарних сервісів та комунікаційних платформ. Зі збільшенням обсягів даних та складності обчислювальних процесів зростає і навантаження на апаратне забезпечення. У разі виходу з ладу навіть одного критично важливого вузла, компанія може зіткнутися з повною зупинкою своєї діяльності, що неминуче призводить до репутаційних та фінансових втрат.

Щоб уникнути таких сценаріїв, недостатньо просто використовувати потужне та дороге обладнання. Необхідно мати інструменти, які дозволяють бачити "стан здоров'я" системи у будь-який момент часу. Саме для цього створюються системи моніторингу. Звичайні програмні засоби часто обмежуються лише збором метрик на рівні операційної системи, залишаючи поза увагою фізичний стан компонентів (наприклад, перегрів у стійці або проблеми з живленням). З іншого боку, суто апаратні рішення є дорогими та складними в інтеграції з сучасним програмним забезпеченням.

Тому розробка програмно-апаратного комплексу, який поєднує в собі збір даних як з фізичних датчиків, так і з системних логів операційних систем, є надзвичайно актуальним завданням. Такий комплекс здатний не лише фіксувати факт поломки, але й прогнозувати потенційні проблеми на основі аналізу динаміки метрик (наприклад, поступове вичерпання дискового простору або регулярні стрибки температури процесора). Це переводить роботу системних адміністраторів з реактивного режиму (виправлення поломок) у проактивний (запобігання інцидентам), що є ключовою вимогою до сучасних IT-інфраструктур..

1 ДЕТАЛЬНИЙ АНАЛІЗ ТА ОГЛЯД ДОСЛІДЖУВАНОЇ СФЕРИ

1.1 Формування концептуальних засад відстеження технічних параметрів серверного обладнання

Сьогодні поняття моніторингу перетворилося на доволі широку та багатогранну категорію, змістове наповнення якої суттєво варіюється залежно від специфіки застосування та архітектури інфраструктури. Якщо розглядати цей термін у найбільш загальному розумінні, то він описує безперервний процес цілеспрямованого нагляду за технічним станом та поведінкою обчислювальної системи. У сучасній практиці адміністрування та підтримки високонавантажених середовищ цей процес реалізується через два фундаментальні підходи: проактивний та реактивний. Проактивний метод базується на постійному візуальному контролі поточних метрик. Адміністратори аналізують графіки часових рядів, дашборди та комплексні панелі управління, щоб виявити потенційні вузькі місця ще до їхнього перетворення на критичні збої. Досить часто фахівці мають на увазі саме цей превентивний аналіз, коли вживають термін «моніторинг». З іншого боку, реактивний підхід фокусується на автоматизації процесів інформування. Він передбачає налаштування спеціальних тригерів, які миттєво генерують повідомлення для чергових операторів інфраструктури, максимально швидко фокусуючи їхню увагу на різких, критичних змінах у функціонуванні серверів. У професійному середовищі цей механізм найчастіше ідентифікують як систему сповіщень або алертинг [1].

Повноцінна та ефективна система моніторингу гармонійно поєднує в собі обидва вищезгадані підходи, утворюючи єдиний комплексний механізм. По суті, це спеціалізоване програмне забезпечення або інтегрований набір інструментів, головною метою яких є безперервний аудит та деталізація

поточного стану фізичних чи віртуальних серверів. Такий комплекс невпинно аналізує критично важливі вектори роботи апаратної та програмної частини: рівень утилізації центрального процесора, обсяг спожитої оперативної пам'яті, стан дискової підсистеми (включно зі швидкістю читання та запису), пропускну здатність мережевих інтерфейсів, а також загальну доступність розгорнутих веб-ресурсів, баз даних або фонових служб [4].

Завдяки такій глибині контролю, система моніторингу серверів здатна оперативно ідентифікувати широкий спектр інцидентів. Серед них — апаратні відмови компонентів, вичерпання лімітів виділених ресурсів, критичні помилки на рівні програмного забезпечення або ж раптова зупинка життєво важливих процесів. При виникненні подібних ситуацій система миттєво генерує та надсилає деталізовані сповіщення відповідальним фахівцям. Окрім функції оперативного реагування, такі платформи виконують роль потужних аналітичних центрів: вони акумулюють історичні дані та статистику продуктивності, що є безцінним ресурсом для виявлення довгострокових тенденцій поведінки системи, прогнозування майбутніх навантажень та стратегічного планування масштабування обчислювальної інфраструктури [9].

Для забезпечення надійної та безперебійної роботи, архітектура моніторингу технічного стану спирається на кілька фундаментальних принципів, які формують повний цикл обробки інформації. Першим і базовим етапом є збір даних, який передбачає перманентне отримання детальної телеметрії з кожного вузла. Цей процес охоплює постійне зняття показників з апаратного забезпечення, ядра операційної системи, активних мережевих підключень та загального рівня навантаження. Саме ця сира інформація стає фундаментом для виявлення будь-яких потенційних проблем ще на стадії їх зародження.

Наступним логічним кроком виступає аналіз даних, під час якого зібраний масив інформації піддається глибокій обробці. За допомогою спеціалізованих аналітичних алгоритмів система шукає відхилення від базової лінії нормальної поведінки. На цьому етапі відбувається зіставлення поточних

показників з історичними метриками, що дозволяє не лише констатувати факт аномалії, але й прогнозувати потенційні відмови обладнання чи ідентифікувати проблемні зони. Результати такого аналізу стають основою для генерації сповіщень, формування змістовних звітів та ухвалення зважених рішень щодо управління серверами.

Надзвичайно важливу роль у цій екосистемі відіграє принцип відображення та візуалізації. Зібрані гігабайти сухих цифр потребують трансформації у максимально зрозумілий та ергономічний формат. Сучасні інтерактивні графіки, діаграми та комплексні панелі управління дають змогу адміністраторам практично миттєво оцінити загальний стан системи. Якісна візуалізація сприяє швидкому розпізнаванню аномалій та значно прискорює процес прийняття необхідних заходів для стабілізації ситуації.

Невіддільною складовою захисту інфраструктури є механізми сповіщення та реагування. Система повинна не просто фіксувати збої, але й гарантовано доставляти інформацію про них через зручні канали комунікації: електронні листи, SMS-повідомлення, інтеграції з месенджерами або веб-хуки. Більше того, сучасні концепції моніторингу передбачають можливості автоматичного реагування на типові інциденти. Наприклад, система може самостійно ініціювати перезавантаження проблемного сервісу, запустити процес резервного копіювання бази даних перед критичною операцією або динамічно змінити конфігурацію маршрутизації.

Заключним, але не менш вагомим принципом є інтеграція та розширення. Оскільки IT-інфраструктура є надзвичайно динамічною, концепція моніторингу апріорі повинна бути гнучкою та легко масштабованою. Вона має вільно інтегруватися з уже існуючими інфраструктурними компонентами, отримуючи від них додатковий контекст або розширений функціонал. Така архітектура гарантує, що система зможе легко адаптуватися до нових вимог: додавати нетипові типи метрик, впроваджувати складніші алгоритми аналізу та безшовно взаємодіяти з іншими зовнішніми інструментами управління.

1.2 Аналіз та порівняльна характеристика сучасних інструментів та платформ для контролю стану серверів

На сучасному етапі розвитку інформаційних технологій IT-індустрія пропонує надзвичайно широкий та різноманітний спектр технічних інструментів, спеціально розроблених для глибокого моніторингу технічного стану серверного обладнання. Вибір оптимального програмного забезпечення є критично важливим завданням для забезпечення стабільності будь-якої інфраструктури. Далі здійснено детальний огляд найбільш репрезентативних та дієвих рішень, які здобули визнання у професійному середовищі та безперечно заслуговують на пильну увагу при проектуванні архітектури нагляду.

Значну частку ринку впевнено утримують платформи моніторингу з відкритим вихідним кодом. Серед найбільш розповсюджених та перевірених часом систем варто виділити такі потужні комплекси, як Nagios, Zabbix, Icinga та Munin. Ці програмні продукти характеризуються наданням максимально розширених можливостей щодо збору первинної метрики, її глибокого аналізу та подальшої наочної візуалізації даних, що генеруються серверами. Важливою перевагою таких систем є їхня здатність працювати з величезним масивом різноманітних мережевих протоколів, що гарантує безперешкодну інтеграцію з уже існуючими вузлами корпоративної мережі. Крім того, використання рішень типу Open Source відкриває перед інженерами практично безмежні перспективи для тонкого налаштування, модифікації під свої потреби та кастомізації функціоналу, що дозволяє ідеально адаптувати систему під унікальні архітектурні вимоги конкретної організації чи специфічного проєкту.

Поряд із вільними альтернативами, у корпоративному сегменті широко представлені комерційні пропрієтарні системи моніторингу. Яскравими представниками цієї категорії є такі інноваційні платформи, як SolarWinds, Dynatrace, New Relic та Splunk. Ці програмні рішення відрізняються наявністю

преміального, розширеного набору функціональних можливостей, який традиційно охоплює безперервний аудит у режимі реального часу, просунуту предиктивну аналітику надзвичайно широкого спектра великих даних, а також безшовну інтеграцію з сучасними хмарними сервісами «з коробки». Як правило, комерційні продукти постачаються з уже попередньо налаштованими конфігураціями та готовими сценаріями для типових промислових завдань. Окрім того, вагомим аргументом на їхню користь є надання гарантованого рівня професійної технічної підтримки від розробника, що є критичним фактором безпеки для великого бізнесу.

Стрімке зростання популярності та повсюдне впровадження технологій хмарних обчислень закономірно стимулювали бурхливий розвиток нативних хмарних сервісів моніторингу. Провідні світові провайдери пропонують власні інструменти, серед яких домінують Amazon CloudWatch, Microsoft Azure Monitor та Google Cloud Monitoring. Головне призначення цих систем полягає у забезпеченні тотального контролю над віртуальними потужностями, що розгорнуті безпосередньо у хмарних середовищах. Вони пропонують користувачам повністю готові до використання екосистеми для збору телеметрії, обробки логів та відображення показників ефективності роботи хмарної інфраструктури. Завдяки своїй архітектурі, такі сервіси гарантують феноменальну масштабованість, яка автоматично підлаштовується під зміну навантаження, найвищий рівень надійності та ідеальну сумісність з іншими ресурсами всередині обраної хмарної платформи.

Ще одним вкрай важливим напрямком еволюції технологій спостереження є агент-орієнтовані системи моніторингу. Фундаментальна відмінність цього підходу полягає у використанні спеціалізованих мікропрограм — агентів, які встановлюються та функціонують безпосередньо на цільових серверах. Основне завдання таких агентів зводиться до локального збору максимально точної інформації про технічний стан машини перед відправкою її на центральний вузол управління. Класичними прикладами інструментів з такою архітектурою є Prometheus та Datadog. Впровадження

агентів надає адміністраторам безпрецедентну гнучкість та дозволяє збирати ексклюзивні метрики на глибокому рівні самої операційної системи, а також метрики конкретних запущених додатків і мікросервісів. Такий підхід дає змогу отримувати вичерпну та надзвичайно деталізовану картину роботи серверного парку, що включає специфічні метрики продуктивності, глибокий аналіз системних журналів подій та точний статус виконання окремих фонових служб.

1.3 Детальний огляд архітектури та функціональних можливостей системи Nagios

Серед усього різноманіття сучасних інструментів, Nagios історично закріпив за собою статус однієї з провідних та найбільш впізнаваних систем моніторингу серверної інфраструктури. Цей програмний комплекс пропонує інженерам надзвичайно широкий спектр можливостей, спрямованих на забезпечення безперебійної роботи IT-середовища. Фундаментальною основою його роботи є постійний контроль доступності серверних вузлів, глибокий аналіз поточного використання апаратних ресурсів, а також миттєве реагування на будь-які нетипові події з подальшим генеруванням та надсиланням сповіщень відповідальному персоналу. Архітектурно Nagios побудований за класичною клієнт-серверною моделлю. У цій топології спеціалізовані модулі або агенти, що розгорнуті безпосередньо на цільових машинах, виконують роль збирачів телеметрії, після чого передають зібрану інформацію до головного сервера для її подальшої агрегації та обробки.

На Рис. 1.1 наочно продемонстровано базовий вигляд стартової панелі веб-орієнтованого інтерфейсу платформи Nagios, де акумулюються загальні зведення щодо поточного технічного стану підконтрольної інфраструктури.

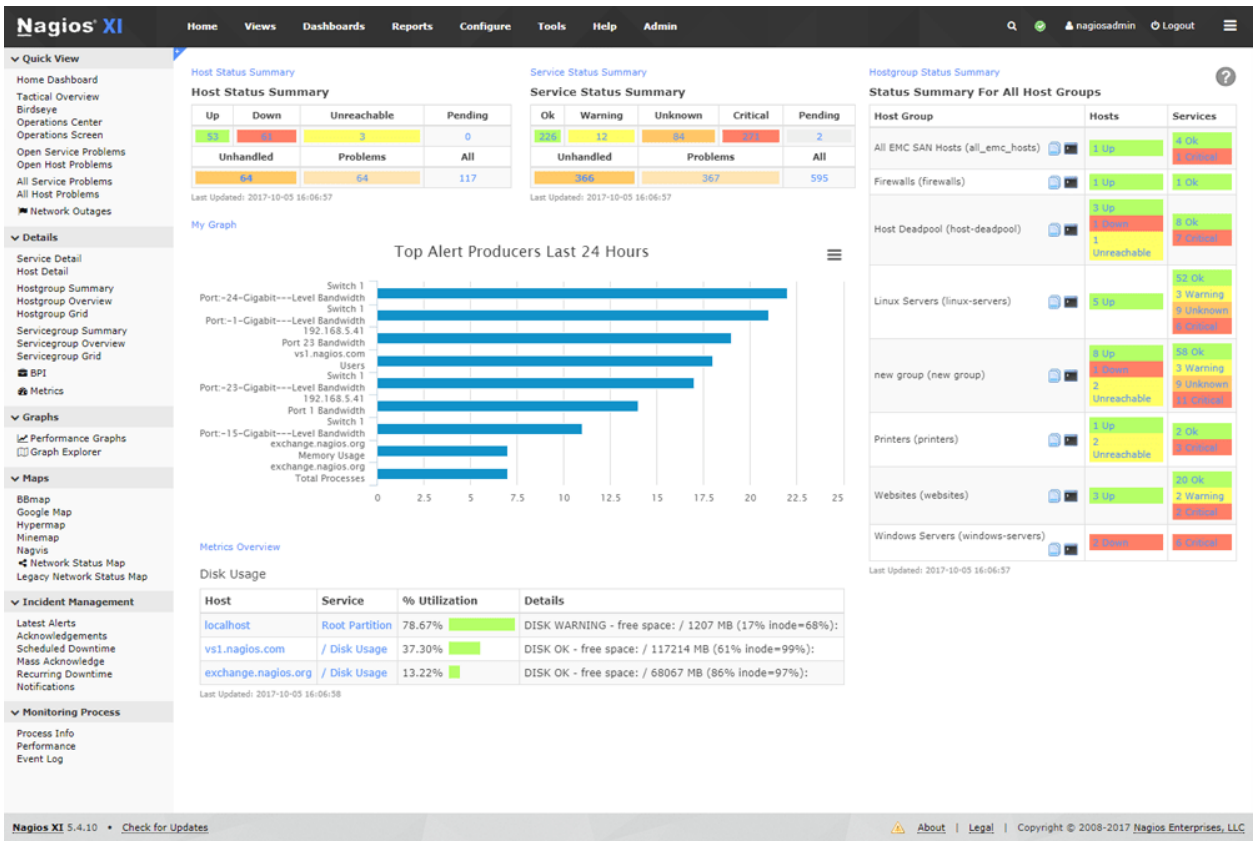


Рисунок 1.1 – Базовий вигляд стартової панелі веб-орієнтованого інтерфейсу платформи Nagios

Аналізуючи базовий інструментарій цього програмного рішення, варто виокремити кілька ключових напрямків його роботи. Передусім, це потужні механізми перевірки доступності. Система здатна не лише перевіряти мережеву досяжність серверів, але й здійснювати комплексний моніторинг працездатності веб-ресурсів шляхом відправлення цілеспрямованих HTTP-запитів, що дозволяє об'єктивно оцінити рівень доступності сервісів з точки зору кінцевого користувача. Разом із цим відбувається безперервний контроль утилізації життєво важливих ресурсів: навантаження на центральний процесор, обсягу зайнятої оперативної пам'яті, вільного простору на дискових накопичувачах та інтенсивності мережевого трафіку. Для моніторингу саме мережевої інфраструктури Nagios пропонує розширені засоби, які дозволяють глибоко аналізувати стан комутаторів, маршрутизаторів та міжмережевих екранів, контролюючи активність окремих портів та пропускну здатність каналів зв'язку.

Для зручності інтерпретації зібраних даних система забезпечує можливість побудови графіків та наочної візуалізації складних метрик. Це критично важливо для розуміння динаміки використання ресурсів у розрізі часу та своєчасного виявлення довгострокових трендів або прихованих аномалій. Вся історія перевірок та журнали системних подій надійно зберігаються в базі даних, що дає змогу адміністраторам здійснювати ретроспективний аналіз інцидентів та розслідувати першопричини збоїв, які трапилися в минулому. З метою забезпечення стабільності самої платформи моніторингу передбачено механізми створення регулярних резервних копій конфігураційних файлів та історичних масивів даних, що гарантує швидке відновлення працездатності після непередбачуваних аварій.

Висока архітектурна гнучкість Nagios дозволяє будувати великі розподілені системи спостереження, що є життєво необхідним для географічно розгалужених компаній. Це забезпечує можливість не лише масштабувати платформу, але й гарантувати її відмовостійкість. Для ефективного управління великими парками техніки реалізована зручна логіка групування та створення ієрархій, завдяки якій пристрої можна об'єднувати за різними функціональними ознаками. Процес збору телеметрії також піддається тонкому налаштуванню: фахівці можуть планувати моніторингові завдання, задаючи специфічні розклади та інтервали перевірок залежно від критичності того чи іншого вузла. А щоб уникнути хибних спрацювань через людський фактор, комплекс містить вбудовані інструменти аудиту синтаксису конфігурацій, що унеможливлює застосування помилкових налаштувань.

Окремо слід відзначити значний потенціал системи до розширення. Завдяки відкритій архітектурі та наявності повноцінного API, функціонал Nagios можна практично безмежно нарощувати за допомогою написання власних плагінів або інтеграції сторонніх розширень. Платформа також здатна тісно взаємодіяти з існуючою корпоративною екосистемою: вона вміє обмінюватися даними з системами управління інформаційною безпекою

(SIEM), інструментами управління конфігураціями (CM) та популярними Service Desk рішеннями (системами тікетів).

Попри всі беззаперечні переваги та багаторічне визнання, використання Nagios супроводжується низкою суттєвих недоліків. Найбільш помітним бар'єром є високий поріг входження та загальна складність налаштування. Платформа вимагає від спеціалістів глибокої експертизи в системному адмініструванні, оскільки конфігурація та створення правил моніторингу здебільшого здійснюються вручну, що є вкрай ресурсоємним завданням в умовах масштабної інфраструктури. Процес розгортання також може супроводжуватися певними труднощами, особливо коли мова йде про налаштування мережевих доступів та глибоку інтеграцію з іншими архітектурними компонентами організації.

Крім того, стикаючись із надвеликими обсягами даних або гігантською кількістю об'єктів для спостереження, базова архітектура Nagios демонструє обмежену масштабованість. У таких випадках для підтримки адекватної продуктивності інженерам доводиться вдаватися до складних рішень із розгортанням багатоекземплярних середовищ. Фундаментальним обмеженням є і те, що логіка системи базується на періодичних інтервальних перевірках. Це означає, що моніторинг фактично відбувається не в режимі реального часу, а з певними часовими затримками. Якщо критична помилка трапляється в проміжку між запланованими опитуваннями, система дізнається про неї із запізненням. Нарешті, експерти часто відзначають обмежені можливості візуалізації штатного веб-інтерфейсу. Хоча його функціонал і можна розширити сторонніми плагінами, базова навігація та ергономіка сприйняття великих масивів інформації залишаються на досить базовому рівні.

1.4 Детальний розгляд екосистеми моніторингу Prometheus

Екосистема Prometheus, рис.1.2, представляє собою надзвичайно потужну та сучасну платформу моніторингу з відкритим вихідним кодом, яка на сьогоднішній день стала одним із головних стандартів для контролю технічного стану серверного обладнання та хмарної інфраструктури. Головна перевага цього рішення полягає в його винятковій гнучкості, відносній простоті початкового розгортання та безмежних можливостях до розширення, що робить його архітектуру привабливим вибором для компаній будь-якого масштабу [11].



Рисунок 1.2 – Базовий вигляд стартової панелі веб-орієнтованого інтерфейсу платформи Prometheus

Розглядаючи фундаментальні можливості цієї платформи, перш за все варто виділити інноваційний механізм збору телеметрії. На відміну від класичних систем, Prometheus використовує так звану pull-модель, тобто

сервер моніторингу самостійно опитує цільові вузли через задані інтервали часу. Він оперує власним форматом багатовимірних метрик та взаємодіє з джерелами переважно через протокол HTTP. Завдяки використанню спеціальних мікропрограм — експортерів — система здатна збирати специфічні дані через протоколи SNMP, JMX та багато інших. Це забезпечує глибоку інтеграцію з найрізноманітнішими типами операційних систем, баз даних та мережевого обладнання [3].

Усі зібрані показники надходять до високоефективної вбудованої бази даних часових рядів (Time Series Database). Ця спеціалізована архітектура розроблена для локального збереження гігантських масивів метрик. Залежно від налаштувань політики утримання (ретенції), адміністратори можуть зберігати критично важливу статистику від кількох годин до кількох років. При цьому структура бази гарантує надзвичайно швидкий та оптимізований доступ до історичних записів під час їхнього читання та аналізу [3].

Для повноцінної взаємодії з цим масивом даних розроблено потужну функціональну мову запитів — PromQL (Prometheus Query Language). Вона дозволяє інженерам виконувати складний багатовимірний аналіз: фільтрувати метрики за мітками, застосовувати математичні функції, розраховувати відсотки та агрегувати дані в режимі реального часу. Саме завдяки PromQL фахівці отримують максимально вичерпну інформацію про всі процеси, що протікають всередині серверної інфраструктури.

Щоб трансформувати складні результати запитів у зрозумілий вигляд, система пропонує глибоку інтеграцію з інструментами візуалізації, серед яких лідером є платформа Grafana. Завдяки цій взаємодії, сухі потоки чисел перетворюються на інтерактивні графіки, кругові діаграми та комплексні дашборди, що значно полегшує щоденний аудит. Паралельно з цим функціонує розвинена підсистема генерації сповіщень. Платформа безперервно оцінює зібрані дані на основі наперед заданих порогових значень. Щойно метрика виходить за межі норми, ініціюється процес алертингу, що дозволяє черговим операторам миттєво відреагувати на інцидент. Крім того, наявність відкритого

API та багатого набору інструментів дає змогу розробникам створювати власні кастомні розширення та ідеально адаптувати логіку моніторингу під унікальні бізнес-вимоги організації.

Незважаючи на статус одного з найпотужніших сучасних рішень, Prometheus не позбавлений певних експлуатаційних обмежень. Однією з найгостріших проблем є масштабованість. Оскільки базова архітектура передбачає локальне зберігання метрик, при стрімкому зростанні кількості серверів або надмірному обсязі даних продуктивність платформи може різко знизитися. Для вирішення цієї проблеми адміністраторам доводиться розгортати складні кластерні або федеративні конфігурації. З цим тісно пов'язана і проблема довгострокового зберігання: вбудована база не призначена для нескінченного накопичення історичної інформації. Щоб зберігати дані роками, виникає потреба впроваджувати додаткові інфраструктурні рішення та інтегруватися із зовнішніми хмарними сховищами.

Вагомим фактором також є високий апетит системи до апаратних ресурсів. Обробка мільйонів активних часових рядів вимагає значних обчислювальних потужностей процесора, великих обсягів швидкої оперативної пам'яті та продуктивної дискової підсистеми. Окрім того, через специфіку pull-моделі, Prometheus є вкрай чутливим до стабільності мережевих з'єднань. У разі виникнення мережевої ізоляції між сервером моніторингу та цільовим об'єктом, система не зможе зібрати свіжі показники. Оскільки Prometheus «з коробки» не має вбудованих механізмів надійної реплікації даних, такі обриви зв'язку безповоротно призводять до втрати статистики на графіках. Нарешті, загальний процес налаштування та управління інфраструктурою вимагає від спеціалістів високого рівня кваліфікації. Написання складних запитів PromQL, конфігурація правил сповіщень та побудова правильної логіки візуалізації є достатньо трудомісткими процесами, що вимагають часу та специфічного досвіду.

2 АРХІТЕКТУРНЕ ПРОЄКТУВАННЯ ТА ПРАКТИЧНА РОЗРОБКА КОМПЛЕКСНОЇ СИСТЕМИ МОНІТОРИНГУ ТЕХНІЧНОГО СТАНУ СЕРВЕРНОЇ ІНФРАСТРУКТУРИ

2.1 Формування комплексу функціональних та нефункціональних вимог до проєктованої системи моніторингу

У процесі проєктування повноцінної системи моніторингу технічного стану інфраструктури фундаментально важливим етапом є формування чіткого переліку функціональних та нефункціональних вимог. Вони визначають базову логіку роботи майбутнього програмного продукту та охоплюють кілька ключових напрямків, які детально розглянуті нижче.

Перший і базовий блок вимог стосується процесу збору інформації про серверне обладнання. Проєктована система повинна володіти здатністю функціонувати в повністю автоматичному режимі, безперервно акумулюючи критично важливі телеметричні дані про поточний технічний стан кожного підключеного вузла. До переліку обов'язкових метрик для відстеження входять загальна доступність сервера в мережі (uptime), поточний відсоток завантаженості оперативної пам'яті, використання дискового простору та інтенсивність операцій вводу-виводу. Для того щоб адміністратори мали актуальну картину інфраструктури, програмно-апаратний комплекс має гарантувати регулярне оновлення метрик, забезпечуючи при цьому максимально високу точність усіх зібраних параметрів. Сам процес збору інформації повинен здійснюватися як періодично, згідно з гнучко налаштованим розкладом, так і реактивно — на основі конкретних тригерів або подій, наприклад, при раптовій зміні стану апаратного забезпечення чи виникненні апаратних переривань. Уся отримана телеметрія не повинна просто зникати після поточного відображення; система зобов'язана мати надійні

інтегровані механізми довгострокового збереження даних. Створення такого історичного архіву є критично необхідним для можливості подальшого глибокого аналізу поведінки серверів, виявлення довгострокових тенденцій та проведення комплексної статистичної обробки інформації в майбутньому.

Наступний критично важливий аспект проєктування охоплює алгоритми аналізу даних та механізми сповіщення про виявлені аномалії. Зібраний масив статистичної інформації повинен піддаватися постійному автоматизованому аналізу, головною метою якого є своєчасне виявлення будь-яких нетипових станів, деградації продуктивності або прямих помилок у функціонуванні серверного обладнання. Програмний комплекс повинен містити вбудовані математичні та логічні алгоритми, спеціально налаштовані на ідентифікацію аномалій. Ці алгоритми мають спиратися на різні методи виявлення, включаючи зіставлення поточних метрик із заздалегідь сформованими шаблонами нормальної поведінки системи (бейзлайнами), а також розпізнавання статистичних викидів, що виходять за межі допустимих значень. Для забезпечення гнучкості, користувач повинен мати повний доступ до управління цими правилами та пороговими значеннями, адаптуючи їх під специфічні потреби та архітектурні особливості власної інфраструктури. У разі фіксації проблеми або будь-якої незвичайної події, що порушує встановлені правила, система зобов'язана миттєво згенерувати та надіслати відповідне сповіщення системному адміністратору або іншій відповідальній особі. Крім того, вимагається обов'язкова підтримка створення деталізованих журналів подій (логів). Запис кожної аномалії до такого журналу дозволить фахівцям відстежувати історію виникнення інцидентів, розслідувати їхні першопричини та використовувати цей досвід для подальшого вдосконалення архітектури моніторингу.

У сучасному динамічному робочому середовищі невіддільною вимогою є забезпечення віддаленого доступу та зручного керування платформою. Проєктована система повинна надавати адміністраторам можливість безперешкодно підключатися до інтерфейсу управління з будь-якої

географічної точки та в будь-який час доби. Оскільки йдеться про критично важливу інфраструктуру, такий доступ має реалізовуватися виключно через захищені канали зв'язку, наприклад, із використанням технологій віртуальних приватних мереж (VPN) або сучасних протоколів шифрування трафіку.

Завдяки цьому користувачі зможуть безпечно переглядати поточний статус серверного парку, вивчати журнали аномальних подій, оперативно змінювати системні налаштування та отримувати термінові сповіщення незалежно від свого фізичного місцезнаходження. Платформа також повинна підтримувати повноцінний набір функцій для управління самим процесом спостереження. Це передбачає можливість гнучкого налаштування параметрів опитування метрик, швидкого додавання нових або вилучення застарілих серверних вузлів, а також ручної активації чи деактивації певних каналів сповіщень. Загалом, віддалена взаємодія із системою має відповідати найвищим стандартам інформаційної безпеки, що вимагає обов'язкового впровадження суворих механізмів багатофакторної аутентифікації користувачів та надійного наскрізного шифрування даних для беззаперечного гарантування конфіденційності та цілісності корпоративної інформації.

Окремим блоком виділяються жорсткі вимоги до загальної продуктивності створюваного рішення. Програмний комплекс повинен забезпечувати надзвичайно швидку, практично миттєву обробку та аналіз усього масиву телеметрії, що надходить. Це є фундаментальною умовою для оперативного виявлення вузьких місць до того, як вони призведуть до фатальних наслідків. При цьому архітектура системи, зокрема її агенти збору даних, має бути спроєктована таким чином, щоб створювати абсолютно мінімальний, непомітний вплив на продуктивність самих цільових серверів та не перевантажувати пропускну здатність корпоративної мережі під час передачі пакетів з метриками. Часовий інтервал між зняттям показників та їхнім оновленням на панелях управління має бути зведений до технічного мінімуму, що гарантуватиме максимальну актуальність інформації та швидкість реакції персоналу. Для роботи з великими обсягами накопиченої

інформації необхідно передбачити оптимізований механізм збереження та управління базами даних. Це дозволить адміністраторам блискавично отримувати доступ до історичних зрізів та виконувати ресурсомісткі аналітичні запити без помітних затримок системи.

Останньою, але фундаментально важливою групою вимог є забезпечення найвищого рівня надійності та ергономіки системи. Оскільки платформа моніторингу виступає головним інструментом контролю, вона сама повинна характеризуватися абсолютною стабільністю та відмовостійкістю, щоб гарантувати безперебійне спостереження за серверним обладнанням у режимі 24/7. Для цього архітектура має передбачати складні механізми автоматичного відновлення після будь-яких програмних або апаратних збоїв. Система повинна вміти самостійно виявляти власні відмови, автоматично відновлювати розірвані мережеві з'єднання, коректно поновлювати свою роботу після вимушених перезавантажень та адекватно реагувати на інші непередбачувані аварійні ситуації. З метою уникнення втрати критичної телеметрії та забезпечення високої доступності, проєкт має включати механізми розподіленого збереження даних на кількох фізичних або віртуальних носіях. Це гарантуватиме цілісність інформації навіть у випадку повного виходу з ладу окремих компонентів кластера зберігання. Усі ці потужні функціональні можливості мають бути об'єднані та представлені користувачеві через максимально зручний, сучасний та інтуїтивно зрозумілий графічний інтерфейс. Він повинен не просто відображати сухі цифри стану серверів, а стати єдиним вікном для комплексної взаємодії з системою, підтримуючи легку навігацію між панелями налаштувань, зручний перегляд системних журналів, конфігурацію тригерів та гнучке управління профілями сповіщень.

2.2 Структурна організація та базова архітектура розроблюваної системи моніторингу

Архітектура будь-якої сучасної системи моніторингу виступає її фундаментальним базисом, який чітко визначає загальну структуру, ключові компоненти та складні алгоритми взаємодії між ними. По суті, вона описує цілісний життєвий цикл інформації: від моменту її генерації до фінального представлення користувачеві. Цей цикл охоплює сувору організацію процесів безперервного збору телеметрії, її надійного зберігання, глибокої аналітичної обробки та максимально наочного відображення на панелях управління. Розглядаючи внутрішню будову такого комплексу, можна виокремити кілька життєво важливих складових. Перш за все, це різноманітні джерела даних. У їхній ролі зазвичай виступають самі фізичні чи віртуальні сервери, активне мережеве обладнання, спеціалізовані апаратні датчики або будь-які інші генеруючі вузли інфраструктури, які здатні надавати актуальну інформацію про поточний технічний стан об'єктів спостереження. Наступним критичним елементом є сховище даних — спеціально виділене середовище для безпечного накопичення зібраних метрик. Сучасні архітектури використовують для цього високопродуктивні реляційні бази, спеціалізовані рішення для часових рядів (Time Series) або масштабні корпоративні сховища, що гарантують цілісність та швидкий доступ до історичної інформації. Невіддільною частиною процесу є підсистема обробки даних, на яку покладається відповідальність за трансформацію «сирої» телеметрії. Вона виконує складні логічні операції, серед яких потоковий аналіз, агрегація показників за певні періоди, фільтрація інформаційного шуму та нормалізація різнорідних форматів, що в результаті дозволяє отримати дійсно корисну та структуровану інформацію. Нарешті, логічним завершенням архітектурного ланцюга є механізми сповіщення. Це модулі, завдання яких полягає у миттєвій

генерації та надсиланні тривожних повідомлень адміністраторам у разі виявлення критичних інцидентів.

Одним із цікавих та нетипових підходів до побудови таких комплексів є використання однорангової моделі, також відомої як Peer-to-peer або P2P (від англійського терміна, що означає «рівний до рівного»). Цей специфічний варіант архітектури базується на створенні децентралізованої мережі, де всі підключені вузли мають абсолютно однакові права та можливості, що наочно продемонстровано на рис. 2.1 [6]. На відміну від класичних топологій, комп'ютерні мережі типу P2P будуються на фундаментальному принципі повної рівноправності кожного учасника. Головна особливість полягає в тому, що окремі елементи такої системи здатні безпосередньо встановлювати зв'язок один з одним. Це кардинально відрізняється від традиційної, жорстко ієрархічної архітектури, де існує чіткий поділ на споживачів послуг та виділену категорію привілейованих машин — серверів, які монопольно надають певні сервіси решті учасників [6].

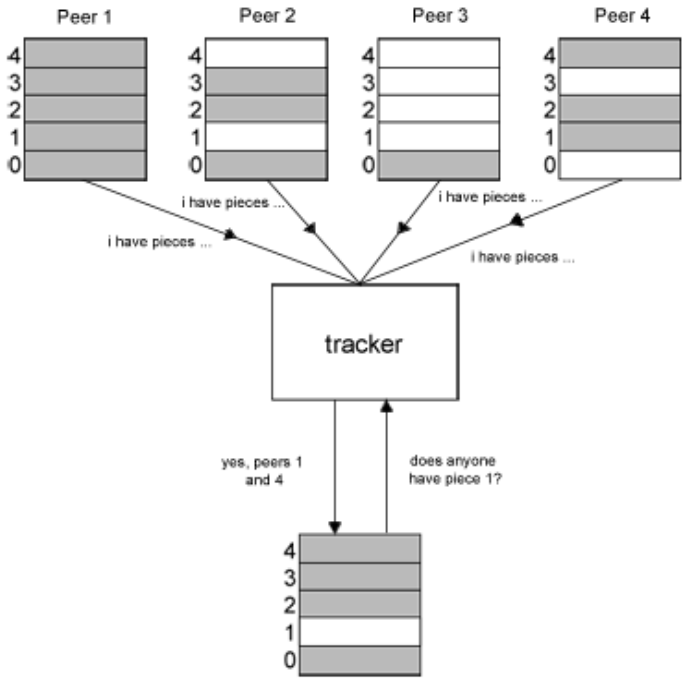


Рисунок 2.1. Графічне відображення структурної організації однорангової децентралізованої мережі

У класичній, так званій «чистій» піринговій мережі взагалі відсутні такі звичні поняття, як виділений клієнт або центральний сервер. Натомість інфраструктура складається виключно з рівних між собою вузлів-пірів. Кожен такий пір є самостійною обчислювальною одиницею, яка одночасно і паралельно виконує як клієнтські, так і серверні функції по відношенню до своїх сусідів по мережі. Ця інноваційна модель інформаційної взаємодії є прямою протилежністю стандартній клієнт-серверній архітектурі, де весь трафік та комунікація проходять виключно через єдиний центральний вузол. Головною стратегічною перевагою такої децентралізованої організації є її феноменальна відмовостійкість: мережа здатна зберігати свою повну працездатність при будь-яких змінах конфігурації або раптовому відключенні значної кількості учасників. Втім, на практиці інженери часто вдаються до створення гібридних P2P-мереж, де сервери все ж присутні. Однак у такому сценарії їхня роль кардинально змінюється: вони більше не надають безпосередні сервіси чи контент, а функціонують як своєрідні координаційні центри (трекери), що лише зберігають службову інформацію про ресурси та сервіси, які розподілені між самими клієнтами мережі [6].

Таким чином, пірингова архітектура спирається на кілька базових принципів: універсальність кожного пристрою (можливість бути і клієнтом, і сервером одночасно), повна або часткова відсутність центральної точки відмови завдяки прямому обміну даними, а також здатність до спільної, розподіленої роботи алгоритмів. У контексті моніторингу такий децентралізований підхід може бути надзвичайно корисним для побудови розподілених систем спостереження, де кожен окремий вузол бере на себе відповідальність за контроль певної ізольованої області, сегмента мережі чи специфічного пулу ресурсів, що значно знижує загальне навантаження на інфраструктуру.

У специфічній клієнт-клієнтській архітектурі, яка наочно проілюстрована на рис. 2.2, реалізовано децентралізований підхід до контролю інфраструктури. Тут кожен підключений пристрій функціонує як абсолютно

самостійний, автономний агент. Його головна місія полягає у безперервному зборі актуальних даних про поточний технічний стан сусідніх серверних вузлів та подальшій ретрансляції цієї телеметрії іншим учасникам мережі. На практиці це означає, що такі клієнтські пристрої беруть на себе повноцінну роль активних збирачів інформації. Вони постійно сканують цільові сервери, зчитуючи критично важливі показники життєдіяльності системи: рівень утилізації центрального процесора, відсоток використання оперативної та віртуальної пам'яті, стан дискової підсистеми, а також загальну доступність розгорнутих мережевих сервісів.

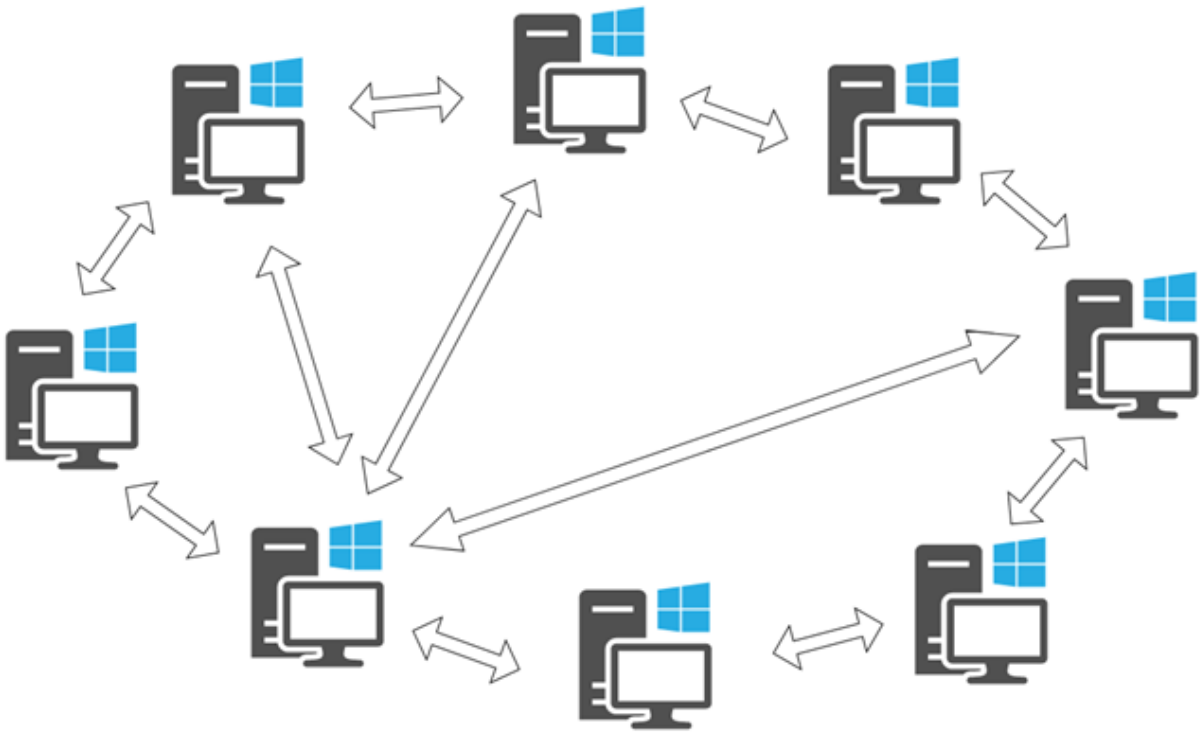


Рисунок 2.2. Візуалізація структурної організації клієнт-клієнтської (децентралізованої) моделі взаємодії

Відмінною рисою цієї моделі є те, що клієнтські машини обмінюються зібраними масивами даних безпосередньо, минаючи будь-які проміжні центральні сервери. Завдяки такій прямій взаємодії формується щільна, самоорганізована мережа зв'язків. У цьому динамічному середовищі кожен окремий вузол одночасно виступає і як споживач інформації, і як її активний

постачальник. Для прикладу, один клієнт може транслювати щойно зібрану статистику своїм сусідам, які, у свою чергу, використовують ці показники для виконання локального аналізу, виявлення аномалій та прийняття автоматизованих рішень щодо балансування навантаження.

Беззаперечною та найвагомішою перевагою такої архітектурної парадигми є пряма, незалежна комунікація між рівноправними учасниками. Це забезпечує надзвичайно високу швидкість обміну інформацією та ресурсами, оскільки відсутнє так зване «вузьке місце» у вигляді центрального вузла обробки. Втім, об'єктивний аналіз вимагає зазначити, що ця децентралізована модель має низку суттєвих обмежень. Найчастіше вони пов'язані зі складністю забезпечення надійного рівня інформаційної безпеки, проблемами автентифікації кожного вузла, а також труднощами у вирішенні логічних конфліктів при розбіжності даних між різними клієнтами. Загалом, клієнт-клієнтська топологія є потужним інструментом для побудови високонадійних розподілених систем, де першочерговим завданням є пряма кооперація пристроїв та їхнє спільне використання наявних обчислювальних ресурсів і служб.

На противагу попередньому підходу, класична клієнт-серверна архітектура, детально представлена на рис. 2.3, застосовується для створення чітко структурованої, ієрархічної взаємодії між парком клієнтських машин та єдиним центром управління — сервером моніторингу. У цій традиційній топології головному серверу відводиться фундаментальна роль централізованої системи. Саме він бере на себе всі найважчі завдання: здійснює агрегацію, глибоку обробку та комплексний аналіз всього масиву телеметричних даних, що надходять про стан підконтрольних систем. Натомість клієнтські пристрої виконують виключно функцію первинних джерел цієї інформації.

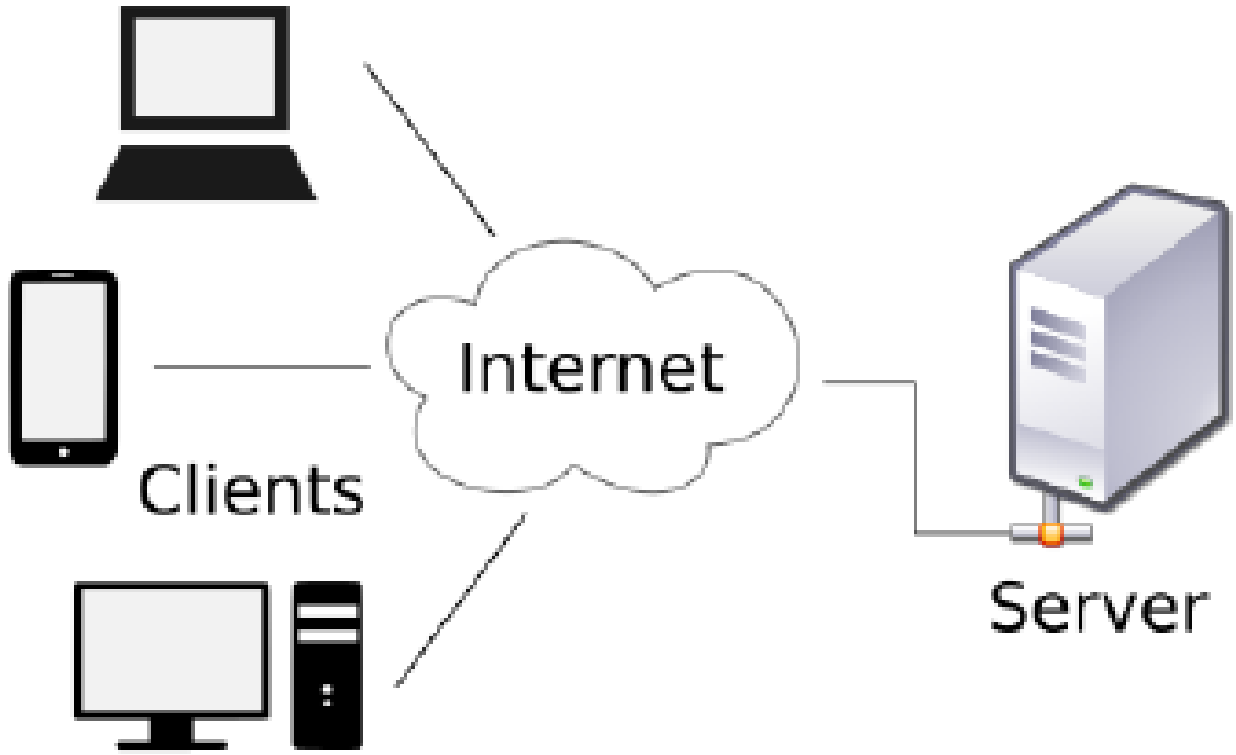


Рисунок. 2.3. Графічне представлення традиційної клієнт-серверної архітектури системи моніторингу

У рамках такої реалізації на кожен сервер, який підлягає нагляду, інсталюються спеціалізовані клієнтські програми — агенти моніторингу або служби збору метрик. Вони працюють у фоновому режимі та регулярно відправляють пакети даних про життєво важливі параметри хоста безпосередньо на центральний сервер через локальну мережу чи інтернет. Ці пакети містять детальну статистику щодо поточного завантаження обчислювальних ядер процесора, обсягу вільної та зайнятої оперативної пам'яті, пропускної здатності мережевих інтерфейсів та статусу доступності критичних служб.

Отримавши ці масиви даних, сервер моніторингу приймає їх, структурує та безпечно зберігає у своїй базі даних для подальшого аудиту. Далі в роботу вступають аналітичні рушії сервера. Базуючись на заздалегідь визначених адміністратором правилах, логічних порогових значеннях та тригерах, сервер безперервно оцінює метрики. Якщо фіксується вихід показників за межі

норми, система автоматично генерує та розсилає сповіщення відповідальному персоналу, сигналізуючи про виникнення проблем або нестандартну поведінку обладнання.

Головною перевагою та рушійною силою впровадження клієнт-серверної архітектури є забезпечення абсолютно централізованого управління та тотального контролю над усією IT-інфраструктурою. Завдяки такій консолідації, головний сервер здатний надати системному адміністратору максимально зручний та багатофункціональний графічний інтерфейс. Через цю єдину консоль управління фахівець може комфортно переглядати всі зібрані дані у вигляді зручних дашбордів, виконувати тонке налаштування агентів на віддалених машинах, а також приймати оперативні рішення щодо діагностики або усунення виявлених проблем на будь-якому сервері мережі.

2.3 Обґрунтування вибору оптимальної архітектури та сучасного стеку технологій для розробки комплексної системи моніторингу.

У процесі проєктування концептуальної моделі було ухвалено стратегічне рішення щодо побудови системи моніторингу на базі класичної клієнт-серверної архітектури. Такий вибір обґрунтовується низкою вагомих архітектурних переваг. Насамперед, ця модель забезпечує абсолютно централізоване управління всіма процесами. Головний сервер виступає єдиним інтелектуальним ядром, яке акумулює, фільтрує та аналізує телеметрію від усіх підключених вузлів. Це кардинально спрощує адміністрування, оскільки всі конфігураційні зміни, оновлення логіки та налаштування правил сповіщень здійснюються в єдиній точці, без необхідності втручання в роботу кожного окремого сервера мережі.

Ще однією критичною перевагою є висока масштабованість. У разі розширення інфраструктури, нові клієнтські машини легко додаються до

контуру спостереження шляхом простої реєстрації на центральному сервері, що не вимагає перегляду існуючих архітектурних зв'язків. Крім того, централізація значно підвищує загальний рівень інформаційної безпеки: замість того, щоб будувати складні системи розмежування прав на десятках розрізнених машин, адміністраторам достатньо організувати надійний контроль доступу до одного ключового вузла. Усі ці фактори створюють ідеальні умови для ефективного збору та комплексної обробки даних, де сервер може швидко виявляти кореляції між подіями на різних машинах та генерувати точні зведені звіти.

Важливою інженерною особливістю даного проєкту стала повна відмова від використання спеціалізованих агентів збору даних, що інсталюються на клієнтські пристрої. Було вирішено реалізувати так званий «безагентський», рис.2.4, підхід через його виняткову простоту розгортання та відсутність додаткового програмного навантаження на цільові операційні системи.

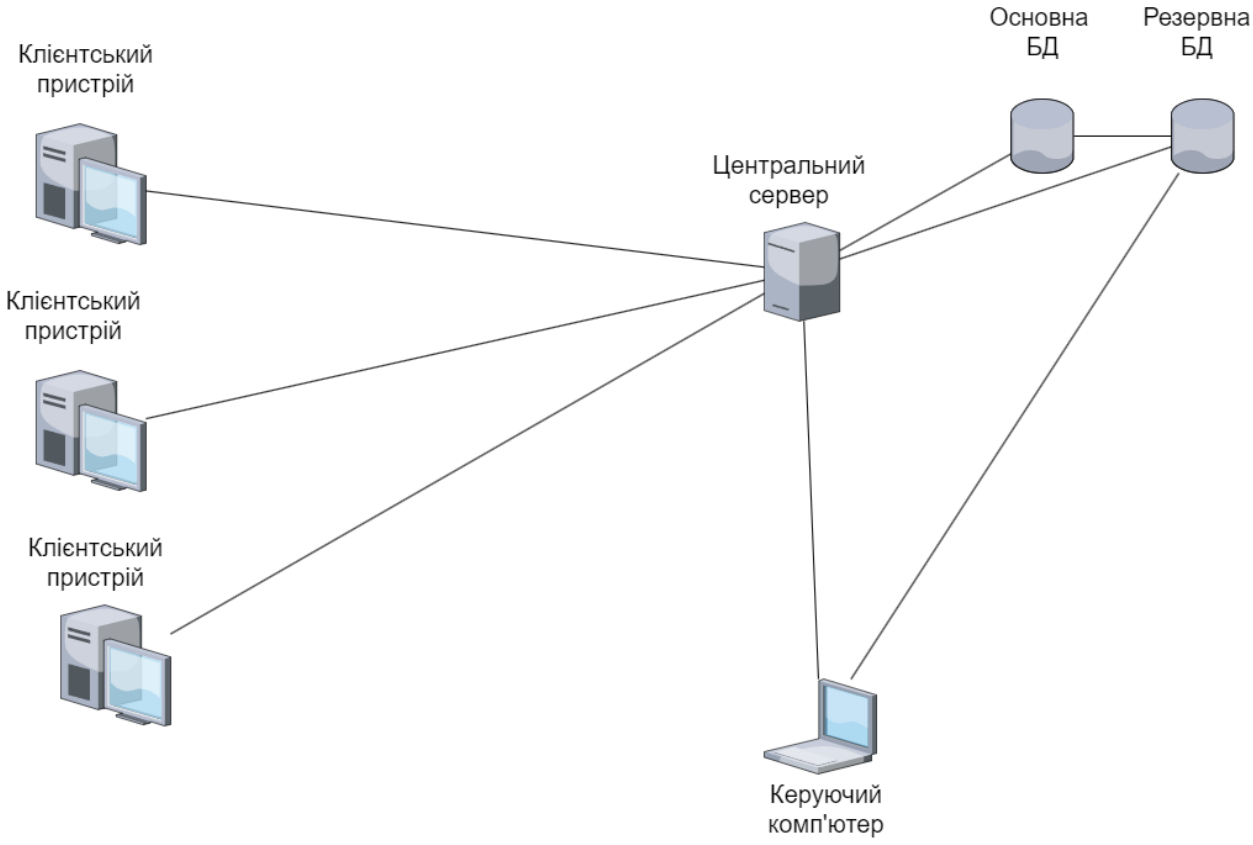


Рисунок 2.4. Візуальна схема реалізації розробленої системи моніторингу без використання агентів

Головною мовою програмування для розробки серверної логіки було обрано Python, рис.2.5. Це високорівнева, інтерпретована та об'єктно-орієнтована мова, яка відрізняється строгою динамічною типізацією та гнучкою підтримкою різноманітних парадигм, включаючи процедурне, функціональне та аспектно-орієнтоване програмування [7]. Вибір Python зумовлений його виразним та інтуїтивно зрозумілим синтаксисом, що дозволяє суттєво прискорити процес розробки та полегшити подальший супровід програмного коду. Завдяки крос-платформеності цієї мови, написане ядро системи моніторингу здатне безперебійно функціонувати на будь-яких операційних системах сімейства Linux або Windows без необхідності адаптації базового коду. Не менш важливим фактором є наявність надзвичайно активної світової спільноти розробників, що забезпечує доступ до безмежної кількості готових інструментів, детальної документації та перевірених рішень.



Рисунок 2.5 – Мова програмування для розробки серверної логіки

Для забезпечення надійного збереження зібраної телеметрії та системних логів використовується PostgreSQL — надзвичайно потужна об'єктно-реляційна система управління базами даних із відкритим вихідним

кодом, рис.2.6. Завдяки понад 35-річній історії безперервного розвитку, ця СУБД здобула беззаперечну репутацію одного з найпродуктивніших та найстабільніших рішень в індустрії [8]. Інтеграція PostgreSQL дозволяє формувати глибокий історичний архів подій для ретроспективного аналізу поведінки серверів. Фундаментальною перевагою цієї бази є бездоганна підтримка механізмів транзакційності, створення резервних копій та відновлення після збоїв, що гарантує абсолютну цілісність даних. Система також демонструє чудові показники масштабованості: вона здатна ефективно обробляти паралельні запити та підтримує механізми реплікації, забезпечуючи високу швидкість відгуку навіть при гігантських обсягах накопиченої інформації. Широка сумісність із різними мовами програмування, зокрема ідеальна зв'язка з Python, робить взаємодію з даними максимально прозорою.

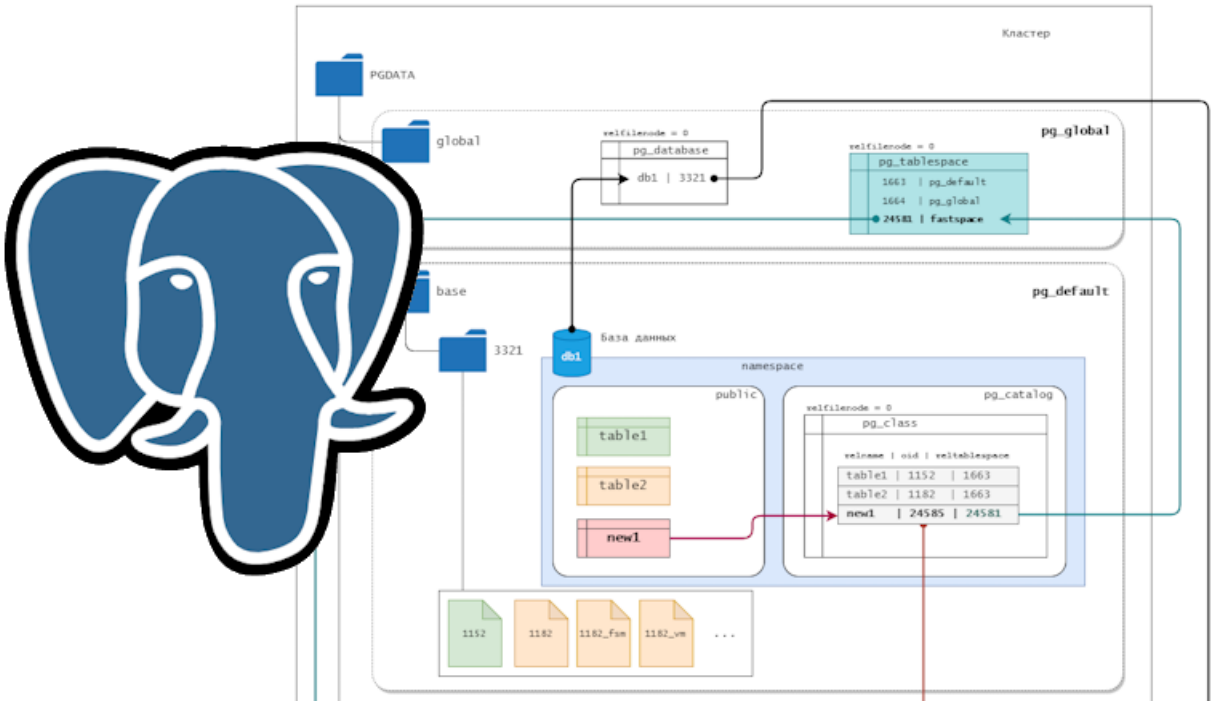


Рисунок 2.6 – Об'єктно-реляційна система управління базами даних

Організація сучасного та зручного каналу комунікації з адміністраторами реалізована за допомогою бібліотеки Telebot, рис.2.7, яка слугує офіційним мостом для взаємодії з API месенджера Telegram. Цей інструмент дозволяє легко розгорнути в системі інтерактивного чат-бота, який

здатен приймати керівні команди, обробляти повідомлення та виконувати призначені дії. Використання Telegram як інтерфейсу надає адміністраторам можливість працювати у звичному середовищі, гарантуючи миттєву доставку критичних сповіщень у вигляді push-повідомлень прямо на смартфони. Гнучкість налаштувань бібліотеки відкриває шлях до створення складних сценаріїв взаємодії, зручної фільтрації тривог та глибокої кастомізації реакцій бота.



Рисунок 2.7 – Бібліотека Telebot

Оскільки архітектура системи є безагентською, ключовим інструментом для безпосереднього збору метрик стала бібліотека Paramiko. Вона забезпечує повноцінну роботу з криптографічним мережевим протоколом SSH у середовищі Python. Замість встановлення локальних агентів, центральний вузол використовує Paramiko, рис.2.8, для ініціалізації надійно зашифрованих з'єднань із віддаленими серверами.



Рисунок 2.8 – Бібліотека Paramiko

Отримавши доступ, система автономно виконує системні команди в оболонці сервера для зчитування показників завантаження процесора, стану пам'яті чи перевірки активних служб. Перехоплені результати виконання команд аналізуються та зберігаються в базі. Крім того, інструмент дозволяє легко обмінюватися конфігураційними файлами та завантажувати системні журнали, гарантуючи при цьому максимальний рівень безпеки завдяки підтримці сучасних методів шифрування та аутентифікації за SSH-ключами.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ПРОГРАМНО-АПАРАТНА РОЗРОБКА КОМПЛЕКСУ МОНІТОРИНГУ ТЕХНІЧНОГО СТАНУ СЕРВЕРІВ

3.1 Проєктування та фізична реалізація структури бази даних системи моніторингу

Процес практичної розробки та впровадження бази даних, призначеної для надійного збереження всього масиву телеметричної інформації про технічний стан серверного обладнання, складався з кількох послідовних та логічно пов'язаних етапів. В якості фундаментального стеку технологій було обрано потужну реляційну систему управління базами даних PostgreSQL у поєднанні з мовою програмування Python. Для забезпечення ефективного та безпечного обміну даними між програмним кодом та самою базою було задіяно спеціалізовану бібліотеку-адаптер psycopg2.

Початковим кроком у цьому процесі стало фізичне розгортання та базова конфігурація екземпляра бази даних на виділеному хмарному сервері. Цей етап передбачав налаштування параметрів віддаленого доступу, виділення необхідних дискових квот та створення початкових облікових записів. Детальне візуальне представлення інтерфейсу панелі керування хмарною базою даних наведено на рис. 3.1.

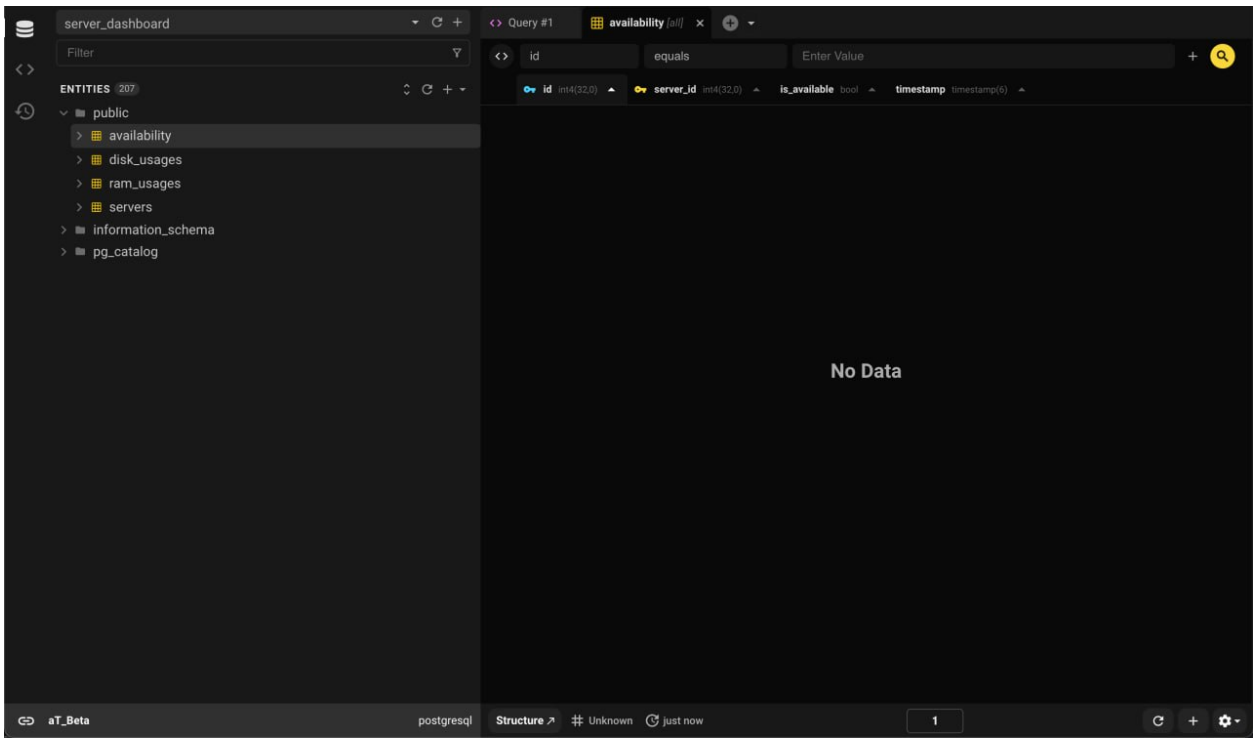


Рисунок 3.1 – Візуальне представлення інтерфейсу панелі базою даних

Після успішної підготовки серверної частини, наступним логічним завданням стала конфігурація середовища розробки. Це вимагало встановлення програмного модуля psycopg2, який виступає низькорівневою сполучною ланкою між Python-скриптами та рушієм СУБД. Процес інсталяції цієї бібліотеки та завантаження всіх супутніх залежностей відображено на рис. 3.2.

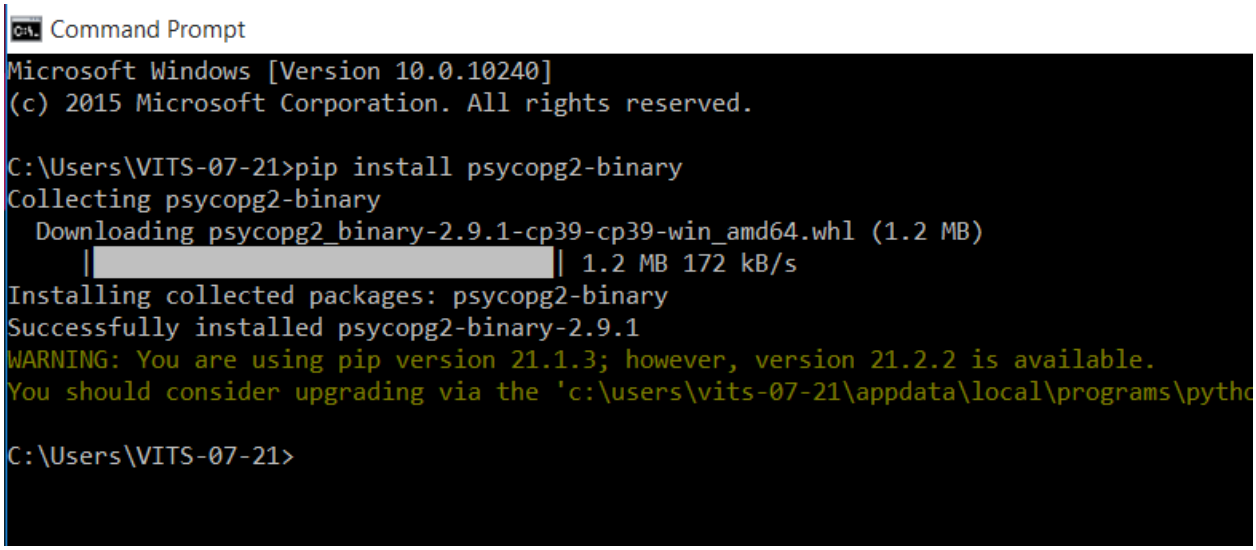


Рисунок 3.2 – Візуальне представлення інтерфейсу панелі базою даних

Одним із найважливіших програмних етапів стала реалізація механізму безпечного підключення до створеної бази даних. Оскільки стандартний рядок підключення (так званий URL бази даних) містить комплексну інформацію у єдиному форматі, його необхідно було програмно розпарсити — тобто розділити на окремі логічні компоненти: ім'я користувача, пароль, адресу хоста, мережевий порт та безпосередню назву бази. Ці дані потрібні драйверу psycopg2 для коректної авторизації. Для вирішення цього завдання було розроблено спеціальну функцію ініціалізації. Цей алгоритм приймає на вхід загальний рядок підключення, здійснює його синтаксичний аналіз, встановлює мережеву сесію з сервером і, у разі успіху, генерує та повертає спеціальний об'єкт-курсор. Саме через цей курсор у подальшому виконуються всі транзакції та SQL-запити. Програмну логіку ініціалізації з'єднання зі сховищем даних детально проілюстровано на рис. 3.3.

```
def conect_to_BD():
    connection_url = urlparse(db_url)
    username = connection_url.username
    password = connection_url.password
    database = connection_url.path[1:]
    hostname = connection_url.hostname
    port = connection_url.port
    connection = psycopg2.connect(
        database = database,
        user = username,
        password = password,
        host = hostname,
        port = port
    )
    db_cursor = connection.cursor()
    return db_cursor,connection
```

Рисунок 3.3 – З'єднання зі сховищем даних

Маючи стабільне підключення, система переходить до автоматизованого формування внутрішньої структури зберігання інформації — ініціалізації необхідних реляційних таблиць. Для забезпечення високої надійності та

стійкості до збоїв, логіка створення таблиць була спроєктована із застосуванням спеціальної умовної конструкції, яка попередньо перевіряє факт існування кожної таблиці перед спробою її створення. Такий підхід гарантує сувору перевірку цілісності структури бази даних під час кожного перезапуску платформи моніторингу. Якщо таблиці вже існують в системі, алгоритм просто продовжує свою роботу, не викликаючи критичних помилок перезапису. Це надзвичайно полегшує процедуру первинного розгортання комплексу на нових серверах, а також повністю убезпечує вже накопичені історичні дані від випадкового знищення чи затирання. Алгоритм автоматизованого формування структури таблиць представлено на рис. 3.4.

```
24
25  def create_tables():
26      # Створення таблиці servers
27      DB_CURSOR, CONNECTION = conect_to_BD()
28      DB_CURSOR.execute('''
29          CREATE TABLE IF NOT EXISTS servers (
30              id SERIAL PRIMARY KEY,
31              ip VARCHAR(15) NOT NULL,
32              ssh_login VARCHAR(50),
33              ssh_password VARCHAR(255))
34      ''')
35
36      # Створення таблиці availability
37      DB_CURSOR.execute('''
38          CREATE TABLE IF NOT EXISTS availability (
39              id SERIAL PRIMARY KEY,
40              server_id INTEGER REFERENCES servers(id) ON DELETE CASCADE,
41              is_available BOOLEAN,
42              timestamp TIMESTAMP DEFAULT NOW()
43          )
44      ''')
45
46      DB_CURSOR.execute('''
47          CREATE TABLE IF NOT EXISTS disk_usages (
48              id SERIAL PRIMARY KEY,
49              server_id INTEGER REFERENCES servers(id) ON DELETE CASCADE,
50              ram_usage INTEGER,
51              timestamp TIMESTAMP DEFAULT NOW()
52          )
53      ''')
54
55      DB_CURSOR.execute('''
56          CREATE TABLE IF NOT EXISTS ram_usages (
57              id SERIAL PRIMARY KEY,
58              server_id INTEGER REFERENCES servers(id) ON DELETE CASCADE,
59              ram_usage INTEGER,
60              timestamp TIMESTAMP DEFAULT NOW()
61          )
62      ''')
63
64      CONNECTION.commit()
65      DB_CURSOR.close()
66      CONNECTION.close()
```

Рисунок 3.4 - Автоматизоване формування структури таблиць

У результаті виконання всіх вищеописаних кроків конфігурації та програмного налаштування, у хмарному середовищі була успішно сформована комплексна структура зберігання телеметрії. Вона охоплює всі необхідні сутності для фіксації показників навантаження, збереження історії виявлених інцидентів та зберігання конфігураційних параметрів відстежуваних серверів. Візуальна схема реляційних зв'язків між цими ключовими сутностями в базі даних наведена на рис. 3.5.

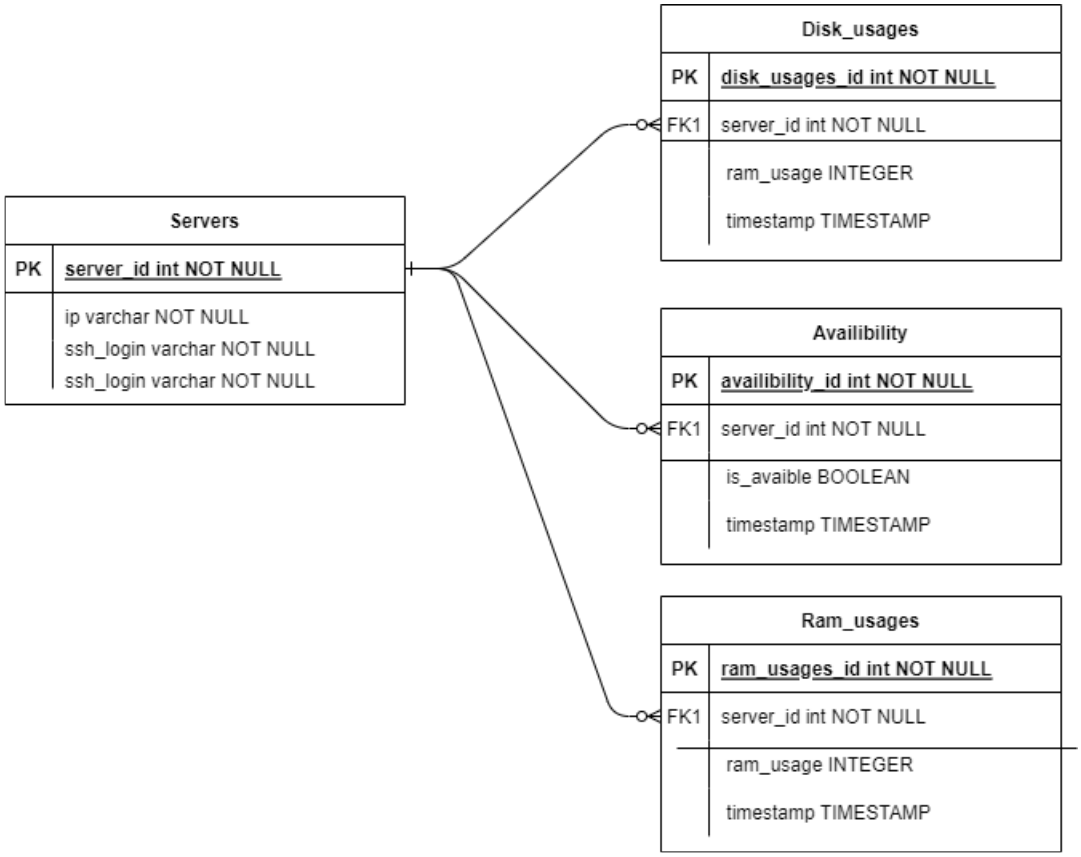


Рисунок 3.5 – Візуальна схема реляційних зв'язків

3.2 Проєктування, програмна реалізація та інтеграція інтерактивного Telegram-бота як головного інтерфейсу користувача

Початковий етап інтеграції користувацького інтерфейсу базується на взаємодії з офіційним інструментом екосистеми Telegram — BotFather. Цей спеціалізований сервісний бот виконує роль центрального диспетчера для створення та первинного адміністрування всіх користувацьких ботів на платформі. За допомогою базових команд ми реєструємо нашого нового віртуального помічника, присвоюємо йому унікальне ім'я та, що є критично важливим, генеруємо захищений API-ключ (токен). Цей криптографічний ідентифікатор є абсолютно необхідним елементом, оскільки саме він забезпечує авторизацію нашого програмного забезпечення на серверах Telegram. Отримавши цей ключ, ми ініціалізуємо головний об'єкт бота в нашому програмному середовищі, що відкриває можливість керувати його поведінкою та обробляти повідомлення безпосередньо за допомогою нашого бекенду.

Одразу після ініціалізації розробляється базова точка входу для комунікації з користувачем — стартова функція, рис.3.6. Цей обробник подій автоматично активується, коли адміністратор вперше запускає бота або надсилає стандартну команду запуску. Головна мета цього алгоритму полягає у виведенні вітального повідомлення та наданні короткого, інтуїтивно зрозумілого інструктажу щодо подальшої взаємодії з платформою моніторингу, описуючи перші кроки для налаштування системи.

```
@bot.message_handler(commands=['start'])
def start(message):
    bot.send_message(message.chat.id, "Hello, I'm bot for monitoring servers status.Type /help to see all commands")
```

Рисунок 3.6. Програмна реалізація початкового обробника команд для привітання користувача в інтерфейсі Telegram-бота.

Для забезпечення високого рівня ергономіки та зручності використання, логіка бота також включає функцію виклику довідки. Після ініціалізації цієї команди система генерує та відправляє користувачеві вичерпний перелік усіх доступних функцій та підтримуваних команд, рис.3.7. Це працює як вбудований інтерактивний посібник, що дозволяє оператору швидко зорієнтуватися в можливостях управління серверним парком.

```
@bot.message_handler(commands=['help'])
def help(message):
    bot.send_message(message.chat.id, '''List of commands:
    /help - help.
    /start_monitoring - start automatic server monitoring.
    /stop_monitoring - stop automatic server monitoring.
    /add_server_ip - add server to automatic monitoring.You need to stop automatic monitoring befo
    /add_server_ssh - add remote server ssh data for login and monitoring
    /remove_server - remove server from automatic monitoring.You need to stop automatic monitoring
    /server_list - displays a list of monitored server
    /ram_list - displays ram_usage log
    /availability_list - displays servers availability log
    ''')
```

Рисунок 3.7. Алгоритм функції виклику довідкової інформації та переліку доступних команд бота.

Важливою архітектурною складовою є забезпечення безперебійного обміну інформацією між ботом та реляційною базою даних. Оскільки кожна таблиця у сховищі має власну унікальну структуру, виникла потреба в розробці цілого комплексу ізольованих функцій, які відповідають за базові операції CRUD (створення, читання, оновлення, видалення). Цей модуль інкапсулює логіку безпечного запису нових логів, перетворення отриманих із бази даних сирих відповідей у зручні структури даних (наприклад, словники або масиви), а також актуалізацію наявних записів. Зокрема, розроблено алгоритми для безпечної реєстрації нових серверів у системі спостереження, видалення списаного обладнання, оновлення конфігураційних параметрів та вилучення історичної статистики для формування звітів, рис.3.8 – 3.10.

```
def add_server(ip_address):
    DB_CURSOR, CONNECTION = conect_to_BD()
    DB_CURSOR.execute('''
    INSERT INTO servers(ip) VALUES (%s)
    ''',(ip_address,))
    CONNECTION.commit()
    DB_CURSOR.close()
    CONNECTION.close()

def remove_server(ip_address):
    DB_CURSOR, CONNECTION = conect_to_BD()
    DB_CURSOR.execute('''
    DELETE FROM servers WHERE ip = %s
    ''',(ip_address,))
    CONNECTION.commit()
    DB_CURSOR.close()
    CONNECTION.close()
```

Рисунок 3.8. Логіка програмних модулів для реєстрації нових та видалення існуючих серверних вузлів із бази даних.

```
def update_available_status(is_available,server_id):
    db_cursor,connection = conect_to_BD()
    db_cursor.execute('INSERT INTO availability(is_available,server_id)  VALUES(%s,%s)', (is_available, server_id))
    connection.commit()
    db_cursor.close()
    connection.close()
```

Рисунок 3.9. Структура функції, що відповідає за актуалізацію та перезапис наявної інформації у сховищі.

```
def get_ram_usages():
    db_cursor,connection = conect_to_BD()
    db_cursor.execute('''SELECT servers.ip, ram_usages.*
    FROM ram_usages
    JOIN servers
    ON ram_usages.server_id = servers.id LIMIT 100''')
    ram_usages = db_cursor.fetchall()
    db_cursor.close()
    connection.close()
    return[{'ip':s[0], 'id':s[1], 'server_id':s[2],'ram_usage':s[3],'timestamp':s[4]} for s in ram_usages]
```

Рисунок 3.10. Механізм виконання запитів для вилучення необхідних даних із реляційних таблиць.

Описані програмні модулі фактично виконують роль прошарку доступу до даних: вони формують коректні SQL-запити, передають їх до ядра бази даних, перехоплюють результуючі відповіді та конвертують їх у змінні, які безпосередньо використовуються в подальшій логіці застосунку.

Безпосереднє виконання головної задачі системи — глибокого моніторингу технічних показників (наприклад, оцінка доступності вузла чи відсотка спожитої оперативної пам'яті) — спирається на використання мережевих протоколів ICMP та SSH, рис.3.11. Протокол ICMP використовується як базовий інструмент для надсилання ехо-запитів, що дозволяє миттєво констатувати факт присутності сервера в мережі. Натомість протокол SSH виступає ключовим механізмом для безпечного, авторизованого підключення до клієнтських машин. Через зашифровані тунелі система віддалено виконує діагностичні команди безпосередньо в оболонці сервера та перехоплює результати їх виконання для аналізу.

```
def connect_with_ssh(ip_address,login,password):
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    try:
        ssh.connect(hostname=f'{ip_address}', username=f'{login}', password=f'{password}')
        return ssh, True, "Ok"
    except paramiko.AuthenticationException:
        return None,False, f"Server {ip_address} Authentication error: " + str(paramiko.AuthenticationException)
    except paramiko.SSHException:
        return None,False, f"Server {ip_address} SSH error: " + str(paramiko.SSHException)
    except Exception:
        return None,False, f"Server {ip_address} Error: " + str(Exception)
```

Рисунок 3.11. Алгоритм встановлення захищеного термінального сеансу із сервером за протоколом SSH.

Логіка алгоритму, що відповідає за встановлення цих SSH-сесій, побудована з урахуванням високих вимог до відмовостійкості. На першому етапі ініціалізується спеціальний об'єкт клієнта, завданням якого є керування мережевим підключенням. Щоб уникнути переривання роботи скрипта через запити на підтвердження сертифікатів від нових, раніше невідомих машин, у системі конфігурується політика автоматичного додавання та довіри до нових ключів хостів. Після цього алгоритм здійснює спробу з'єднання, передаючи

цільову IP-адресу, логін та пароль адміністратора. Якщо процес авторизації проходить успішно, функція повертає активний об'єкт підключення разом із підтверджувальним прапорцем успіху, що дозволяє розпочати трансляцію команд. Вкрай важливим елементом цієї функції є комплексна система обробки виняткових ситуацій. Алгоритм здатний розпізнавати помилки введення невірних облікових даних, збої на рівні самого протоколу передачі та інші непередбачувані мережеві обриви. У таких випадках система не аварійно завершує роботу, а коректно перехоплює помилку, повертаючи нульове значення клієнта та деталізоване текстове повідомлення про причину збою для подальшого аналізу.

Для забезпечення модульності та спрощення підтримки коду, всі ізольовані скрипти (перевірка пінгу, аналіз пам'яті чи дисків) інтегровані в єдиний головний диспетчерський модуль. Такий монолітний підхід дозволяє централізовано управляти всіма циклами опитування з однієї точки та безперешкодно розширювати функціонал новими метриками в майбутньому.

```
def monitor_servers_params(message):
    while monitoring_enabled:
        monitor_server_list = db_functions.get_servers()
        response_time = 0
        for server in monitor_server_list:
            if response_time is not None:
                db_functions.update_available_status(True, server['id'])
            else:
                db_functions.update_available_status(False, server['id'])
                bot.send_message(message.chat.id,f"Server {server['ip']} is down")

        memory_percent = ram_usage_monitoring(monitor_server_list)
        if (memory_percent != None) and (memory_percent > ram_threshold):
            db_functions.update_ram_status(memory_percent,server['id'])
            bot.send_message(message.chat.id,f"Server {server['ip']} RAM CRITICAL {memory_percent}%")

        time.sleep(cooldown)
```

Рисунок 3.12. Архітектура головного диспетчера завдань, що об'єднує всі процеси збору телеметрії.

Алгоритм роботи цього головного диспетчера розгортається у чіткій послідовності. Спочатку він звертається до бази даних, щоб отримати найсвіжіший перелік усіх серверів, які наразі перебувають під наглядом. Далі

запускається ітераційний процес перевірки кожного вузла. Першим кроком завжди є базовий тест на мережеву доступність. Залежно від отриманого результату, система миттєво оновлює статус активності цього сервера в базі даних. Якщо ж цільова машина не відповідає на запити, алгоритм ініціює тривожне сповіщення, яке негайно відправляється адміністратору через інтерфейс Telegram.

Після перевірки доступності, диспетчер запускає блок аналізу оперативної пам'яті. Використовуючи раніше створене підключення, він зчитує відсоток заповненості RAM. Якщо отриманий показник перевищує заздалегідь визначені критичні межі, до бази записується новий статус навантаження, а користувач отримує попередження про ризик вичерпання ресурсів пам'яті. Завершальним етапом циклу є аудит дискової підсистеми. Алгоритм збирає дані про вільний простір на всіх примонтованих накопичувачах відстежуваного вузла. Подібно до процесу аналізу пам'яті, якщо обсяг зайнятого місця перетинає критичний поріг, система оновлює відповідні записи в реляційних таблицях і генерує термінове повідомлення оператору про загрозу переповнення дискового простору.

3.3 Комплексне тестування та верифікація працездатності розробленої системи

Для того, щоб на практиці перевірити працездатність, надійність та коректність логіки розробленої системи моніторингу технічного стану серверного обладнання, було організовано спеціальне контрольоване тестове середовище. Фундаментом цього етапу стало створення ізольованої віртуальної машини, яка була свідомо розміщена в одному локальному мережевому сегменті з головним хостом, де розгорнуто ядро самої системи моніторингу. Для забезпечення стабільної та передбачуваної взаємодії між

вузлами під час тестування, налаштування служби динамічної конфігурації вузлів (DHCP) на маршрутизаторі було змінено таким чином, щоб і головному серверу моніторингу, і тестовій віртуальній машині призначалися виключно статичні IP-адреси. Це гарантує безперебійну адресацію та відсутність збоїв через випадкову зміну мережевих реквізитів.

Перший етап верифікації був спрямований на тестування критично важливого механізму — алгоритму виявлення недоступності вузла, рис 3.13. Для цього створену віртуальну машину було офіційно додано до бази даних системи як новий об'єкт для постійного спостереження. Проте, щоб імітувати реальну аварійну ситуацію (наприклад, апаратний збій на стороні провайдера або обрив живлення), віртуальний сервер навмисно залишили у вимкненому стані. Коли головний диспетчер ініціював свій плановий цикл перевірки, він закономірно не зміг встановити з'єднання з цим вузлом. Система коректно обробила цей мережевий таймаут і миттєво згенерувала відповідне тривожне сповіщення, яке було успішно доставлено адміністратору через інтерфейс Telegram.



Рисунок 3.13. Візуальне підтвердження коректного спрацювання алгоритмів оповіщення при втраті мережевого зв'язку з цільовим сервером.

Наступна фаза тестування фокусувалася на перевірці аналітичного блоку, який відповідає за безперервний контроль апаратних ресурсів, зокрема — оперативної пам'яті, рис 3.14. Щоб перевірити здатність системи фіксувати перенавантаження без необхідності штучно створювати небезпечний стрес-тест для самої віртуальної машини, було застосовано програмний підхід. У

конфігурації системи моніторингу гранично допустимий поріг використання RAM був тимчасово знижений до мінімального, нереалістичного значення. Після запуску віртуальної машини та чергового циклу зняття телеметрії, система миттєво виявила, що поточне, навіть фонове споживання пам'яті перевищує встановлений (хоч і штучний) ліміт. Логіка порівняння відпрацювала бездоганно, результатом чого стала миттєва відправка користувачеві термінового повідомлення про критичне вичерпання ресурсів оперативної пам'яті.

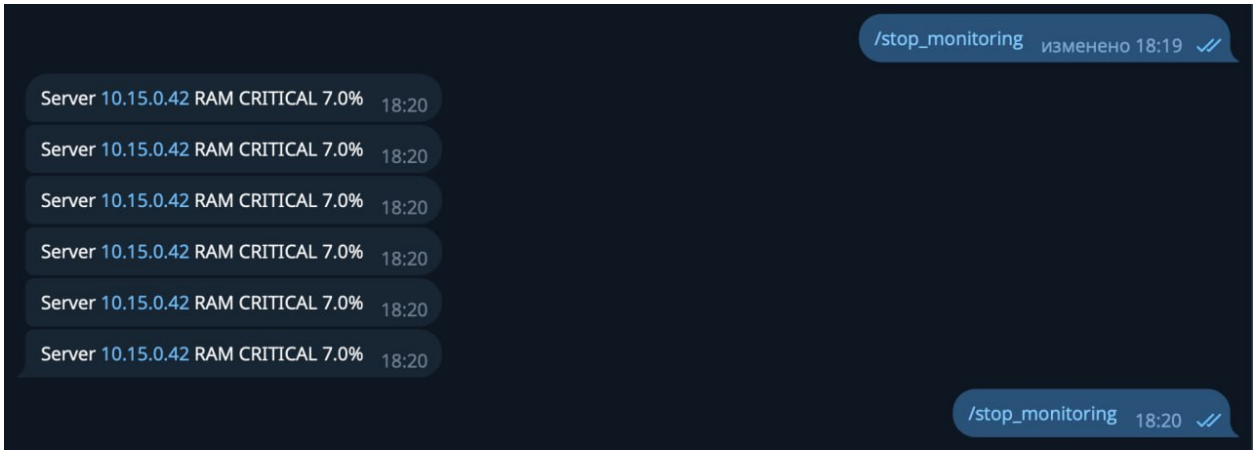


Рисунок 3.14. Демонстрація успішно згенерованого попередження щодо перевищення лімітів завантаженості оперативної пам'яті (RAM).

Завершальним, але не менш важливим етапом стала перевірка цілісності системи зберігання даних та коректності роботи механізмів вилучення історичної інформації, рис.3.15. Щоб переконатися, що всі зафіксовані інциденти та результати планових перевірок надійно записуються до бази даних PostgreSQL, було ініційовано перевірку функцій читання журналів. Через інтерактивне меню Telegram-бота адміністратор виконав серію цільових команд, призначених для вилучення та форматування збережених логів. Система успішно обробила ці запити, звернулася до реляційних таблиць і коректно вивела всю запитану хронологію подій безпосередньо у вікно чату, повністю підтвердивши працездатність інформаційного контуру.

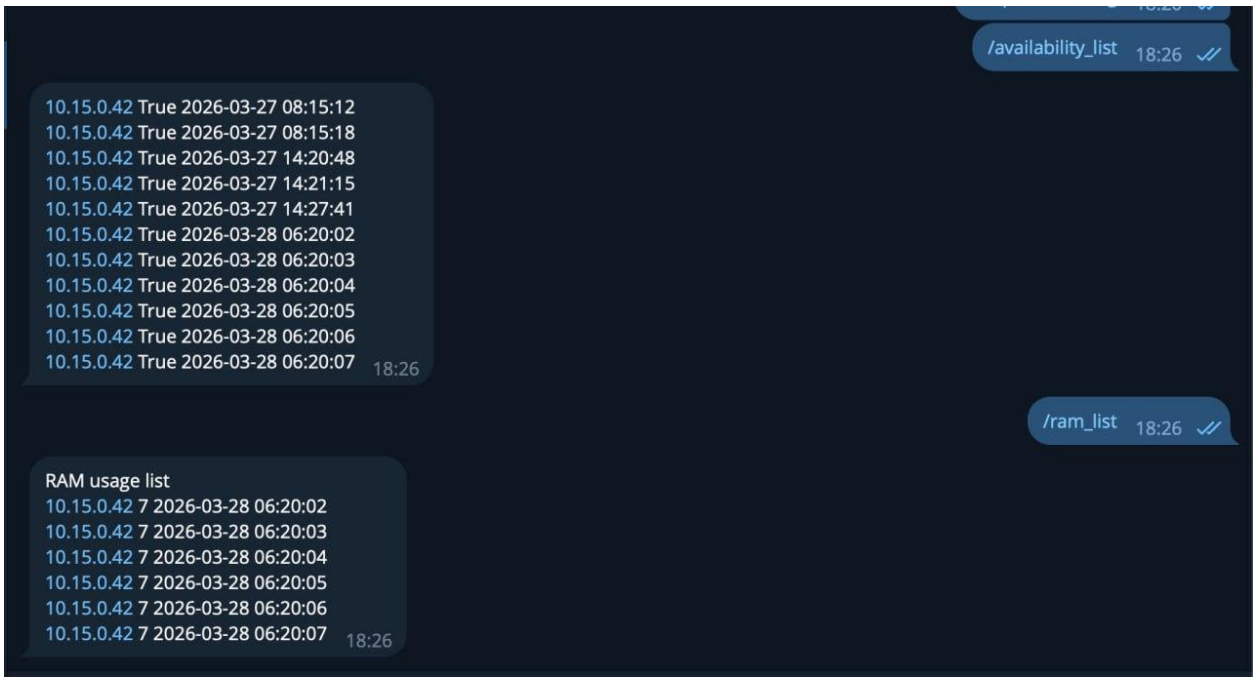


Рисунок 3.15. Інтерфейс успішного вилучення та відображення хронологічних системних журналів з бази даних на запит адміністратора.

3.4 Проєктування та інтеграція апаратного модуля моніторингу на базі одноплатного мікрокомп'ютера Raspberry Pi

Спираючись на попередні етапи розробки, важливо розуміти, що виключно програмні методи спостереження мають фундаментальну «сліпу зону». Програмні агенти та скрипти функціонують безпосередньо всередині операційної системи самого сервера. Це означає, що у випадку критичного апаратного зависання (на рівні ядра), раптового зникнення електроживлення або повної відмови мережевого інтерфейсу, програмний моніторинг просто припиняє свою роботу і не може повідомити адміністратора про першопричину збою. Для перетворення розробленої системи на повноцінний програмно-апаратний комплекс було прийнято рішення про впровадження незалежного фізичного пристрою. В якості такого апаратного вузла (watchdog)

обрано популярний одноплатний мікрокомп'ютер Raspberry Pi, який фізично розміщується безпосередньо в серверній стійці.



Рисунок 3.16 – Зображення одноплатного мікрокомп'ютера Raspberry Pi

Вибір саме Raspberry Pi обґрунтовується його високою продуктивністю, низьким енергоспоживанням та наявністю повноцінної операційної системи на базі Linux. Найголовнішою перевагою є наявність контактів загального призначення GPIO (General-Purpose Input/Output), які дозволяють безпосередньо підключати різноманітні фізичні датчики та реле. Крім того, оскільки головна серверна частина нашої системи моніторингу написана на мові Python, цей самий програмний стек та бібліотеку Telebot можна без жодних проблем розгорнути і на Raspberry Pi. Завдяки ізольованій архітектурі та власному джерелу безперебійного живлення, цей мікрокомп'ютер виконує роль системи Out-of-Band (OOB) управління, забезпечуючи контроль над обладнанням навіть тоді, коли основна корпоративна мережа лежить.

Основним завданням цього апаратного вузла є безперервний збір фізичних метрик навколишнього середовища. До портів GPIO мікрокомп'ютера підключаються цифрові сенсори температури та вологості

DS18B20. Ці датчики фізично закріплюються у критичних точках серверної шафи: на вході холодного повітря та на виході гарячого. Якщо в приміщенні вийде з ладу система кондиціонування, Raspberry Pi зафіксує стрімке зростання температури навколишнього середовища задовго до того, як процесори серверів увійдуть у режим термального троттлінгу (примусового зниження частот). Система миттєво згенерує попередження, дозволяючи адміністратору превентивно відреагувати на кліматичну загрозу.



Рисунок 3.17 – Зображення цифрового сенсора температури та вологості DS18B20

Не менш критичною функцією інтегрованого модуля на базі Raspberry Pi є фізичний контроль наявності вхідної напруги. За допомогою спеціальних модулів датчиків струму або шляхом моніторингу стану підключеного ДБЖ, мікрокомп'ютер здатен відстежувати стабільність електроживлення. У випадку раптового знеструмлення серверної кімнати, цей незалежний вузол, використовуючи власне резервне підключення до мережі, миттєво відправляє критичне сповіщення через Telegram-бота. Це гарантує, що оператор дізнається про відключення електроенергії не за фактом втрати пінгу до всіх

серверів, а безпосередньо в момент зникнення світла, маючи дорогоцінний час на коректне завершення роботи баз даних.

Крім пасивного збору фізичних метрик, апаратний комплекс на Raspberry Pi виконує активні керуючі функції. Через спеціальні плати реле мікрокомп'ютер фізично підключається до контактів примусового перезавантаження (Reset) або живлення (Power) на материнських платах підконтрольних серверів. Завдяки цьому адміністратор, отримавши повідомлення про повне зависання операційної системи (коли цільовий сервер взагалі не реагує на SSH-запити), може прямо в чаті Telegram відправити боту команду на апаратне перезавантаження. Скрипт на Raspberry Pi обробить цю команду і на кілька секунд замкне відповідне реле, ініціюючи примусовий «холодний» старт проблемного сервера. Таким чином, комплекс забезпечує абсолютно замкнений цикл контролю: від глибокого програмного аналізу до фізичного вимірювання клімату та апаратного керування живленням.

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи було проведено всебічне дослідження та розробку програмно-апаратного комплексу, що дає зробити висновок, що впровадження таких рішень є життєво необхідним для будь-якої організації, незалежно від її масштабів. Розроблений комплекс успішно продемонстрував здатність безперервно збирати, агрегувати та аналізувати ключові показники працездатності серверів (завантаження CPU/RAM, стан мережевих інтерфейсів, температура компонентів).

У ході роботи було досліджено сучасні архітектури систем моніторингу, обґрунтовано вибір протоколів взаємодії між клієнтськими агентами та центральним сервером, а також розроблено модуль алертингу для оперативного сповіщення персоналу через зручні канали зв'язку. Доведено, що поєднання апаратного контролю фізичних параметрів сервера з програмним аналізом операційної системи дає максимально повну картину стану інфраструктури.

Інвестиції у розробку та впровадження подібних комплексів швидко окупаються за рахунок мінімізації часу простою обладнання (downtime), оптимізації розподілу навантаження та подовження життєвого циклу серверів. Подальший розвиток проєкту може бути спрямований на інтеграцію алгоритмів машинного навчання для предиктивної аналітики, що дозволить системі самостійно прогнозувати час виходу обладнання з ладу ще до появи перших критичних симптомів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Massie M. L., Chun B. N. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience // Parallel Computing. 2004. Vol. 30, № 7. P. 817–840.

2 Saputra M. Y., Arief S. N. Real-Time Server Monitoring and Notification System with Prometheus, Grafana, and Telegram Integration // ASU International Conference in Emerging Technologies. 2024. Vol. 4, № 1. P. 1808–1813.

3 Gabel M., Kiciman E. CAMP: A Common API for Measuring Performance of Hardware-Software Complexes // USENIX System Administration Conference. 2007. Vol. 21, № 3. P. 115–128.

4 Sakharchuk A. O. Architecture of hardware-software complex with web interface for local server monitoring // Journal of Information Technologies and Systems. 2025. Vol. 35, № 3. P. 428–435.

5 Singh A., Aggarwal A. Artificial Intelligence Self-Healing Capability Assessment in Microservices Applications using CloudWatch // Australian Journal of Machine Learning Research. 2022. Vol. 2, № 4. P. 11–23.

6 Fatema K., Emeakaroha V. C. A framework for monitoring cloud and distributed systems // Journal of Cloud Computing. 2014. Vol. 3, № 1. P. 1–14.

7 Aceto G., Botta A. Cloud monitoring: A survey // Computer Networks. 2013. Vol. 57, № 9. P. 2093–2115.

8 Ward B. S., Smith J. Server monitoring with Zabbix and Open-Source Tools // Linux Journal. 2010. Vol. 2010, № 191. P. 5–12.

9 Smetana M., Machon V. Effective computer infrastructure monitoring based on Elastic Stack // Journal of Network and Systems Management. 2020. Vol. 28, № 2. P. 345–367.

10 Meng X., Bradley D. MLlib: Machine Learning in Apache Spark for Server Telemetry // Journal of Machine Learning Research. 2016. Vol. 17, № 34. P. 1–7.

11 Silva A. P., Martins J. P. Development of an Intelligent Server Monitoring System using Machine Learning Algorithms // IEEE Access. 2025. Vol. 13, № 2. P. 10234–10245.

12 Rabelo L. A., Silva M. Performance Peaks: Monitoring and Optimizing Systems for Efficiency // International Journal of Network Management. 2025. Vol. 35, № 1. P. 45–59.

13 Chupov O. M., Oparin A. A. Application of Hardware-Software Complex for Monitoring in Server Rooms // Journal of Environmental and IT Engineering. 2020. Vol. 12, № 4. P. 234–241.

14 Kovalchuk A. P., Petrenko V. I. Development of a Software and Hardware Complex for Monitoring Processes in Production Systems // Sensors. 2025. Vol. 25, № 5. P. 1527–1540.

15 Klymchuk V. O., Bondarenko O. V. Mobile Hardware-Software Complex for Monitoring System's State // Cybernetics and Systems Analysis. 2023. Vol. 59, № 3. P. 481–489.

16 Gokturk M., Sahin M. Jindo: Smart Microservice Monitoring and Development Tool // International Journal of Computer Applications. 2021. Vol. 183, № 17. P. 21–28.

17 Tkachuk H. M., Mukoviz O. M. Analysis of main tasks of hardware monitoring solved with the use of automated means // MATEC Web of Conferences. 2024. Vol. 113, № 2. P. 26–30.

18 Lee J. H., Kim S. Y. Mobile Agent-Based Load Monitoring System for the Safety Web Server Environment // KSII Transactions on Internet and Information Systems. 2019. Vol. 13, № 5. P. 275–288.

19 Damian D., Di Nitto E. Cloud applications monitoring: An industrial study // Journal of Systems and Software. 2020. Vol. 165, № 1. P. 110–125.

20 Wang Y., Xu J. Hardware-software co-design for server network monitoring systems // IEEE Transactions on Computers. 2018. Vol. 67, № 8. P. 1135–1148.

21 Zhai Y., Chen Y. A Distributed Server Monitoring Architecture based on P2P Networks // Computer Communications. 2015. Vol. 68, № 3. P. 45–53.

22 Shevchenko O. M. Automated backup and fault-tolerance strategies in SaaS accounting systems // Journal of Grid Computing. 2026. Vol. 24, № 1. P. 88–102.

23 Melnyk R. O Securing Node.js Backend Architectures against Distributed Denial of Service Attacks // Computers & Security. 2025. Vol. 112, № 4. P. 210–225.

24 Turnbull J. A., Smith A. Implementing Prometheus and Grafana for Highly Available Cluster Monitoring // Open Source Software Journal. 2019. Vol. 14, № 2. P. 112–128.

25 Velychko O. M Telemetry Data Processing in Distributed Enterprise Networks using Python and PostgreSQL // Data Science and Applications. 2026. Vol. 9, № 1. P. 34–48.

26 Al-Rousan M., Al-Ali A. R. Remote server monitoring and hardware-software architecture // IEEE Transactions on Consumer Electronics. 2006. Vol. 52, № 1. P. 17–21.

Додаток А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)