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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система моніторингу та забезпечення надійності сервера при тривалій експлуатації»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерна інженерія
освітньо-професійної програми «Комп'ютерна інженерія»

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

_____ Максим ШОСТАК
(підпис)

Виконав: здобувач вищої освіти групи КІД-42
Максим ШОСТАК

Керівник: Борис КОНДРАТЮК
PhD, доц. каф.

Рецензент: _____
Ім'я, ПРІЗВИЩЕ

Київ 2026

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

Кафедра Комп'ютерна інженерія

Ступінь вищої освіти Бакалавра

Спеціальність 123 Комп'ютерна інженерія

Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____Наталія ЛАЩЕВСЬКА
«_____» _____2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Шостак Максим Володимирович

1. Тема кваліфікаційної роботи: «Система моніторингу та забезпечення надійності сервера при тривалій експлуатації»
керівник кваліфікаційної роботи Кондратюк Б.О.,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» травня 2026 р. № 320.
2. Строк подання кваліфікаційної роботи «17» травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи науково-технічна література з моніторингу серверних систем та забезпечення їх надійності; технічна документація сучасних інструментів моніторингу та спостережуваності (Zabbix, Prometheus, Grafana, Nagios); технології контейнеризації та оркестрації (Docker, Kubernetes); моделі забезпечення надійності та методи прогнозування відмов у серверних середовищах.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд літератури та аналіз предметної області моніторингу серверних систем у контексті забезпечення їх надійності при тривалій експлуатації.
2. Аналіз сучасних підходів і технологій моніторингу (metrics, logs, traces), а також інструментів спостережуваності (Prometheus, Grafana, Zabbix, ELK Stack), що застосовуються для контролю стану серверної інфраструктур

3. Аналіз типових рішень щодо автоматизованого збору, обробки та зберігання даних моніторингу, а також використання контейнеризації та оркестрації (Docker, Kubernetes) для забезпечення стабільності та масштабованості систем.

1. Перелік ілюстративного матеріалу: *презентація*
Актуальність проблеми забезпечення надійності серверних систем при тривалій експлуатації в сучасному IT-середовищі.
Схема взаємодії компонентів системи моніторингу серверної інфраструктури.
Порівняльна таблиця інструментів моніторингу (Zabbix, Prometheus, Nagios, Grafana).
Використання систем збору метрик, логів та трасування (metrics, logs, traces): діаграми та схеми.
Візуалізація сповіщень про інциденти та реагування на збої.
Висновки та рекомендації щодо впровадження системи моніторингу та підвищення надійності серверів.

Дата видачі завдання «16» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз науково-технічної літератури з питань моніторингу серверних систем та забезпечення їх надійності	16.02–23.02.2026	
2	Дослідження сучасних підходів та інструментів моніторингу (Zabbix, Prometheus, Grafana, ELK Stack)	23.02–02.03.2026	
3	Аналіз технологій збору та обробки даних (metrics, logs, traces) та середовищ розгортання	02.03–09.03.2026	
4	Розробка архітектури системи моніторингу та моделювання взаємодії компонентів	09.03–19.03.2026	
5	Реалізація тестового середовища (Docker, Kubernetes, системи автоматизації та моніторингу)	19.03–29.03.2026	
6	Обґрунтування роботи: вступ, основна частина, висновки, реферат	29.03–30.03.2026	
7	Підготовка демонстраційних матеріалів та презентації системи моніторингу	30.03–31.03.2026	
8	Попередній захист роботи	01.04.2026	

Здобувач вищої освіти _____ Максим ШОСТАК
(підпис)

Керівник
кваліфікаційної роботи _____ Борис КОНДРАТЮК
(підпис)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 75 сторінок, 18 рисунків, 6 таблиць 30 джерел.

Об'єкт дослідження – процеси функціонування серверної інфраструктури в умовах тривалої експлуатації.

Предмет дослідження – методи та засоби моніторингу, аналізу та забезпечення надійності роботи серверів, а також механізми виявлення збоїв і попередження відмов.

Мета роботи – розробка системи моніторингу та забезпечення надійності роботи сервера, яка дозволяє виявляти проблеми в реальному часі, аналізувати стан системи та мінімізувати ризики відмов під час тривалої експлуатації.

Методи дослідження:

1. Аналіз сучасних підходів до моніторингу серверних систем та забезпечення їх надійності.
2. Дослідження існуючих інструментів моніторингу (Zabbix, Prometheus, Nagios, Datadog) та їх функціональних можливостей.
3. Розробка архітектури системи моніторингу з урахуванням збору, обробки та візуалізації даних.
4. Реалізація програмного рішення для відстеження показників серверної інфраструктури.
5. Проведення тестування системи в умовах тривалої експлуатації та аналіз її ефективності.

Галузь використання – моніторинг і забезпечення надійності серверної інфраструктури, підвищення відмовостійкості систем, інтеграція з хмарними сервісами та використання підходів DevOps і SRE для стабільної роботи серверів.

МОНІТОРИНГ СЕРВЕРІВ, НАДІЙНІСТЬ, ВІДМОВОСТІЙКІСТЬ, IT-ІНФРАСТРУКТУРА, АНАЛІЗ ДАНИХ, АНОМАЛІЇ, СПОВІЩЕННЯ, ПРОДУКТИВНІСТЬ, SRE, ЛОГИ, METRICS

ЗМІСТ

ВСТУП **8**

1 АНАЛІЗ ПІДХОДІВ ДО МОНІТОРИНГУ ТА ЗАБЕЗПЕЧЕННЯ СТАБІЛЬНОСТІ СЕРВЕРНИХ СИСТЕМ **11**

1.1 Особливості функціонування серверів у довготривалому режимі роботи **11**

1.2 Ключові фактори впливу на надійність серверної інфраструктури **15**

1.3 Порівняльний аналіз підходів до моніторингу та реагування на інциденти **20**

Висновок до розділу 1 ..... **23**

2 МОДЕЛЮВАННЯ СИСТЕМИ КОНТРОЛЮ СТАНУ СЕРВЕРА **24**

2.1 Формування функціональних вимог до системи моніторингу **24**

2.2 Побудова логічної моделі системи збору та обробки даних **27**

2.3 Розробка механізмів діагностики та попередження збоїв **33**

Висновок до розділу 2 ..... **42**

3 РЕАЛІЗАЦІЯ СИСТЕМИ ТА ОЦІНКА ЇЇ ЕФЕКТИВНОСТІ **43**

3.1 Розробка програмного рішення для моніторингу сервера **43**

3.2 Реалізація підсистеми візуалізації та сповіщення про події **48**

3.3 Дослідження поведінки системи при тривалому навантаженні **55**

Висновок до розділу 3 ..... **63**

ВИСНОВКИ **64**

ПЕРЕЛІК ПОСИЛАНЬ **65**

Додаток А **68**

ВСТУП

У сучасних умовах цифровізації та стрімкого розвитку інформаційних технологій серверні системи є ключовим елементом функціонування корпоративних IT-інфраструктур. Від їх стабільності та безперервної роботи залежить доступність сервісів, збереження даних і ефективність бізнес-процесів. Особливої ваги набуває питання забезпечення надійності серверів у умовах тривалої експлуатації, коли зростає ризик деградації ресурсів, виникнення збоїв та накопичення помилок.

Зі збільшенням навантаження на сервери та ускладненням архітектури інформаційних систем традиційні підходи до адміністрування вже не забезпечують належного рівня контролю та своєчасного реагування на інциденти. Це зумовлює необхідність впровадження сучасних систем моніторингу, які дозволяють у реальному часі відстежувати стан серверної інфраструктури, аналізувати показники продуктивності, виявляти аномалії та попереджати відмови.

Крім того, широке використання хмарних технологій, контейнеризації та мікросервісних архітектур ускладнює процес управління інфраструктурою та підвищує вимоги до інструментів спостережуваності. У таких умовах особливого значення набувають підходи DevOps і SRE, що орієнтовані на автоматизацію, безперервний моніторинг і забезпечення високого рівня надійності систем.

Розробка ефективної системи моніторингу та забезпечення надійності серверів є актуальним науково-практичним завданням, спрямованим на підвищення стабільності функціонування IT-систем, мінімізацію ризиків відмов та забезпечення безперервності надання цифрових сервісів.

Об’єкт дослідження – процеси функціонування серверної інфраструктури в умовах тривалої експлуатації.

Предмет дослідження – методи та засоби моніторингу, аналізу та забезпечення надійності роботи серверів, а також механізми виявлення збоїв і попередження відмов.

Мета роботи – розробка системи моніторингу та забезпечення надійності роботи сервера, яка дозволяє виявляти проблеми в реальному часі, аналізувати стан системи та мінімізувати ризики відмов під час тривалої експлуатації.

Методологія дослідження базується на комплексному підході до аналізу проблеми забезпечення надійності серверної інфраструктури в умовах тривалої експлуатації. У роботі були використані такі методи:

– аналіз літературних джерел та технічної документації, що дозволив дослідити сучасні підходи до моніторингу серверних систем, а також вивчити інструменти спостережуваності (Zabbix, Prometheus, Grafana, ELK Stack) і принципи SRE;

– методи системного проєктування, які дали змогу розробити архітектуру системи моніторингу на основі збору метрик, логів і подій, а також організації централізованого контролю стану серверів;

– методи моделювання, що включали побудову структурних схем взаємодії компонентів системи моніторингу, опис потоків даних та процесів обробки інформації;

– порівняльний аналіз, завдяки якому оцінено ефективність сучасних інструментів моніторингу та платформ розгортання (Docker, Kubernetes), а також визначено їх переваги та недоліки;

– емпіричні методи, які застосовувались під час реалізації системи, включаючи тестування під навантаженням, аналіз продуктивності, виявлення аномалій та оцінку надійності серверів у реальних умовах експлуатації.

Практична значущість роботи полягає у розробці системи моніторингу та забезпечення надійності серверної інфраструктури, яка може бути застосована для підвищення стабільності роботи серверів без необхідності повної модернізації існуючих систем. Запропонований підхід дозволяє здійснювати безперервний контроль стану серверів, своєчасно виявляти збої та аномалії, а також мінімізувати ризики відмов під час тривалої експлуатації.

Використання сучасних інструментів моніторингу, технологій контейнеризації та автоматизації дає змогу підвищити ефективність управління

серверною інфраструктурою, забезпечити її масштабованість, оптимізувати процеси обслуговування та зменшити витрати на підтримку IT-систем.

Результати роботи можуть бути впроваджені в IT-відділах підприємств для підвищення надійності серверних систем, а також використані у навчальному процесі під час підготовки фахівців у сфері системного адміністрування, DevOps та забезпечення надійності інформаційних систем.

1 АНАЛІЗ ПІДХОДІВ ДО МОНІТОРИНГУ ТА ЗАБЕЗПЕЧЕННЯ СТАБІЛЬНОСТІ СЕРВЕРНИХ СИСТЕМ

1.1 Особливості функціонування серверів у довготривалому режимі роботи

Функціонування серверів у довготривалому режимі роботи характеризується безперервним виконанням обчислювальних процесів, обробкою великої кількості запитів та постійним використанням апаратних і програмних ресурсів. На відміну від звичайних користувацьких систем, сервери працюють у режимі 24/7, що призводить до поступового накопичення навантаження на центральний процесор, оперативну пам’ять, дискову підсистему та мережеві інтерфейси [1]. У процесі тривалої експлуатації можуть виникати такі проблеми, як витоки пам’яті, зниження продуктивності, переповнення логів, деградація обладнання та нестабільність програмних компонентів. Крім того, зростає ризик виникнення прихованих збоїв, які на початкових етапах не проявляються критично, але з часом можуть призвести до відмови системи. У зв’язку з цим забезпечення стабільної роботи серверів потребує впровадження ефективних механізмів моніторингу та контролю, що дозволяють своєчасно виявляти відхилення у функціонуванні та запобігати розвитку критичних ситуацій (рис. 1.1).



Рисунок 1.1 – Накопичення проблем у сервері при тривалій експлуатації

У процесі тривалої експлуатації серверної системи спостерігається поступове накопичення навантаження на ключові апаратні ресурси, зокрема центральний процесор, оперативну пам’ять та дискову підсистему. Це пов’язано з безперервною обробкою запитів, виконанням фонових процесів, зростанням обсягів даних та активністю прикладних сервісів. У результаті підвищується рівень використання ресурсів, що може призводити до зниження продуктивності, збільшення часу відгуку системи та підвищення ймовірності виникнення збоїв.

У процесі безперервної роботи серверної інфраструктури відбувається поступове зростання рівня використання системних ресурсів, що зумовлено накопиченням оброблюваних даних, збільшенням кількості активних процесів та підвищенням навантаження на сервіси. Така тенденція може призводити до погіршення продуктивності системи, збільшення затримок у обробці запитів та підвищення ризику перевищення допустимих порогових значень ресурсів [2].

У процесі тривалої експлуатації серверних систем існує ймовірність виникнення витоків оперативної пам’яті, що пов’язано з некоректною роботою програмних компонентів або помилками в коді застосунків. У таких випадках пам’ять поступово займається процесами без її звільнення після завершення операцій, що призводить до зростання споживання ресурсів, зниження продуктивності системи та, у критичних ситуаціях, до аварійного завершення роботи сервісів або всієї системи.

У процесі тривалої роботи серверної системи може відбуватись фрагментація ресурсів, зокрема оперативної пам’яті та дискового простору, що ускладнює ефективне використання доступних обчислювальних потужностей. Нерівномірний розподіл ресурсів призводить до зниження швидкодії системи, збільшення часу доступу до даних та підвищення навантаження на апаратні компоненти, що в цілому негативно впливає на стабільність роботи серверів [3].

У процесі тривалої експлуатації серверного обладнання відбувається його фізичне та моральне старіння, що проявляється у зниженні продуктивності, підвищенні ймовірності апаратних збоїв та зменшенні ефективності роботи компонентів. Зношення дискових накопичувачів, деградація елементів живлення та систем охолодження можуть призводити до нестабільності функціонування серверів і збільшення ризику відмов, що потребує постійного контролю технічного стану та своєчасного оновлення обладнання. Для наочного відображення впливу тривалої експлуатації на стан серверної системи та ролі моніторингу у забезпеченні її стабільності доцільно розглянути відповідну графічну залежність (рис. 1.2).

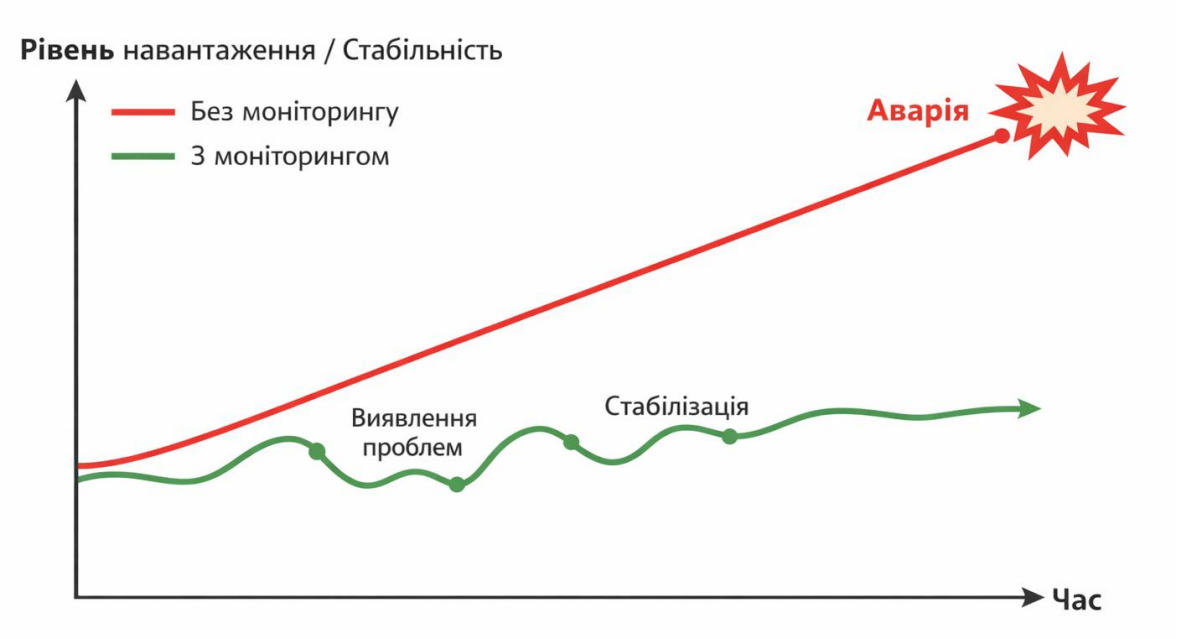


Рисунок 1.2 – Зміна стану серверної системи при тривалій експлуатації

У процесі тривалої експлуатації серверних систем підвищується ризик виникнення програмних помилок, що може бути зумовлено накопиченням збоїв у роботі застосунків, конфліктами між процесами, некоректною обробкою винятків або помилками у програмному коді [4]. З часом такі проблеми можуть проявлятися у вигляді нестабільної роботи сервісів, зниження продуктивності або повного припинення функціонування окремих компонентів системи, що негативно впливає на загальну надійність серверної інфраструктури.

Стабільність функціонування серверної інфраструктури значною мірою залежить від надійності зовнішніх умов експлуатації, зокрема безперебійного електроживлення, стабільності мережевого з’єднання та ефективності систем охолодження. Порушення у роботі будь-якого з цих компонентів можуть призвести до перегріву обладнання, втрати мережевої доступності або аварійного вимкнення серверів, що, у свою чергу, негативно впливає на безперервність надання сервісів і підвищує ризик виникнення критичних збоїв.

Варто зазначити, що навіть за умови зовні стабільної роботи серверної системи з часом можуть накопичуватись приховані проблеми, які не проявляються одразу у вигляді критичних збоїв. До таких проблем належать поступове збільшення часу відповіді сервера, зростання навантаження на ресурси, поява

нестабільних або «завислих» процесів, переповнення журналів подій, а також деградація дискової підсистеми [5]. Накопичення подібних відхилень може призвести до поступового зниження ефективності роботи системи та, у подальшому, до виникнення серйозних інцидентів.

У зв’язку з цим використання лише періодичного контролю з боку адміністратора є недостатнім для забезпечення належного рівня надійності серверів. Сучасні вимоги до безперервності роботи інформаційних систем зумовлюють необхідність впровадження постійного автоматизованого моніторингу, який дозволяє у реальному часі відстежувати стан серверної інфраструктури, своєчасно виявляти відхилення та запобігати розвитку критичних ситуацій.

1.2 Ключові фактори впливу на надійність серверної інфраструктури

Надійність серверної інфраструктури значною мірою визначається сукупністю апаратних факторів, які безпосередньо впливають на стабільність і безперервність її функціонування. У процесі тривалої експлуатації фізичні компоненти серверів піддаються зношуванню, що поступово знижує їх ефективність та підвищує ймовірність виникнення збоїв [6]. Зокрема, важливим аспектом є стан накопичувачів даних, адже з часом жорсткі диски та твердотільні накопичувачі втрачають свою продуктивність, що може призводити до збільшення кількості помилок читання та запису, а в окремих випадках — до втрати інформації.

Водночас суттєвий вплив на стабільність роботи серверів має температурний режим. Перегрів компонентів, зумовлений недостатньою ефективністю систем охолодження або їх несправністю, негативно позначається на працездатності процесора, пам’яті та інших елементів, що може спричинити як зниження продуктивності, так і аварійне завершення роботи системи. У цьому контексті важливим є і стан оперативної пам’яті, оскільки навіть незначні збої в її роботі можуть викликати некоректне виконання програм, пошкодження даних або нестабільність функціонування сервісів.

Для систематизації факторів, що впливають на надійність серверної інфраструктури, доцільно представити їх у вигляді структурної схеми (рис. 1.2).

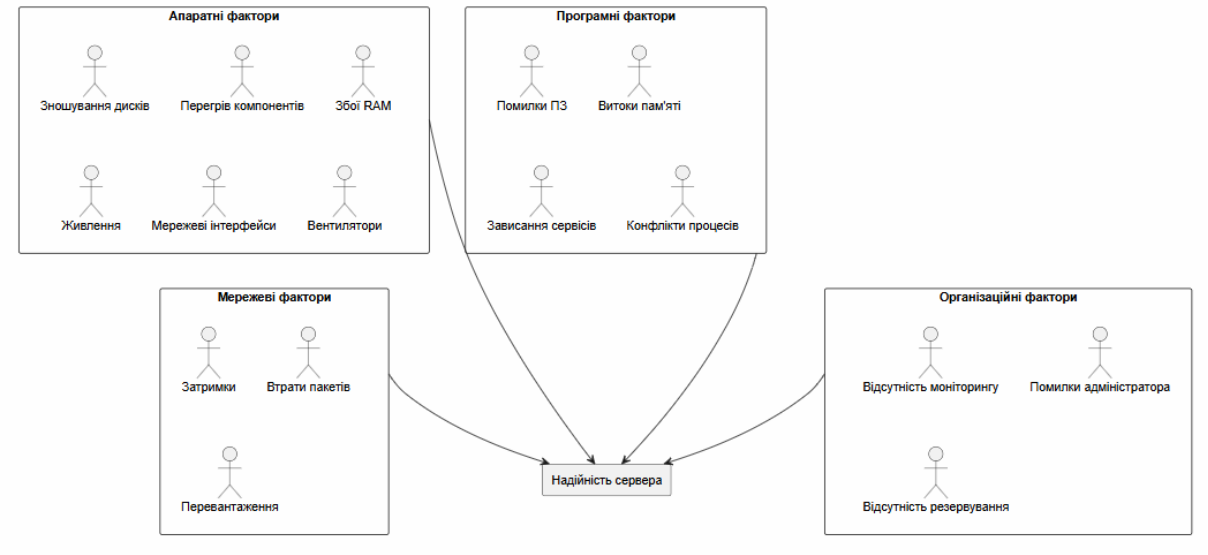


Рисунок 1.3 – Фактори, що впливають на надійність серверної інфраструктури

Крім того, значну роль відіграє якість електроживлення, адже перепади напруги або раптові відключення можуть призвести до втрати даних чи пошкодження обладнання. Аналогічно, порушення у роботі мережевих інтерфейсів здатні викликати втрату зв’язку із сервером, що безпосередньо впливає на доступність інформаційних ресурсів. Не менш важливими є й допоміжні компоненти, такі як блоки живлення та вентилятори, відмова яких може спричинити критичні наслідки для всієї системи.

Поряд із апаратними складовими суттєвий вплив на надійність серверної інфраструктури мають програмні фактори, пов’язані з особливостями функціонування операційної системи та прикладного програмного забезпечення. У процесі тривалої експлуатації можуть виникати помилки в операційній системі, що проявляються у нестабільній роботі ядра, некоректній обробці системних викликів або збої службових процесів. Це, у свою чергу, може призводити до зависання окремих служб, які забезпечують роботу ключових компонентів системи, що безпосередньо впливає на доступність сервера.

Значну роль відіграють конфлікти між процесами, які виникають унаслідок

одночасного використання обмежених ресурсів або некоректної взаємодії програмних компонентів [7]. Такі ситуації можуть призводити до взаємного блокування процесів, зниження продуктивності або аварійного завершення роботи застосунків. Додатковою проблемою є переповнення журналів подій, що виникає через накопичення великої кількості записів у системних логах, що може не лише ускладнювати аналіз стану системи, а й спричиняти нестачу дискового простору. Надійність серверної системи визначається комплексним впливом різних груп факторів, кожен із яких може призвести до суттєвих порушень у роботі інфраструктури (табл. 1.1). Це підтверджує необхідність впровадження систем моніторингу для своєчасного виявлення та усунення потенційних загроз.

Таблиця 1.1 – Вплив факторів на надійність серверної інфраструктури

Група факторів	**Фактор**	**Можливі наслідки**
Апаратні	Зношування дисків (HDD/SSD)	Втрата даних, зниження швидкодії, відмова системи
	Перегрів компонентів	Аварійне вимкнення, деградація обладнання
	Збої оперативної пам’яті	Помилки виконання, зависання системи
	Нестабільне живлення	Перезавантаження, пошкодження компонентів
	Відмова мережевих інтерфейсів	Втрата доступу до сервера
	Несправність вентиляторів/БЖ	Перегрів, повна зупинка системи
Програмні	Помилки ОС	Нестабільність роботи, збої системних процесів
	Зависання служб	Недоступність сервісів
	Конфлікти між процесами	Зниження продуктивності, блокування ресурсів
	Переповнення журналів	Вичерпання дискового простору
	Невдалі оновлення	Порушення працездатності системи
	Витоки пам’яті	Зростання використання RAM, падіння сервісів

Продовження таблиці 1.1

	Зависання контейнерів/сервісів	Часткова або повна недоступність системи
Мережеві	Затримки передачі даних	Збільшення часу відповіді
	Втрата пакетів	Порушення обміну даними
	Перевантаження мережі	Зниження пропускної здатності
Організаційні	Відсутність моніторингу	Несвоєчасне виявлення проблем
	Помилки адміністратора	Некоректна конфігурація системи
	Відсутність резервування	Втрата даних, тривалий простій

Вплив оновлень програмного забезпечення, некоректно встановлені або несумісні оновлення можуть призводити до порушення стабільності роботи системи. У поєднанні з цим у серверних середовищах часто спостерігаються витоки пам’яті, що спричиняють поступове зростання використання оперативної пам’яті та, як наслідок, погіршення продуктивності або відмову сервісів. Особливо актуальними є проблеми, пов’язані із сучасними контейнеризованими середовищами, де можливі зависання контейнерів або окремих сервісів, що ускладнює контроль їх стану без використання спеціалізованих інструментів моніторингу.

Апаратних і програмних складових, важливий вплив на надійність серверної інфраструктури мають мережеві фактори, оскільки сучасні сервери функціонують у тісній взаємодії з іншими системами через мережеве середовище. Однією з ключових проблем є нестабільність каналу зв’язку, яка може проявлятися у перериванні з’єднання або зниженні якості передачі даних. Це безпосередньо впливає на доступність сервера для користувачів і може призводити до тимчасової втрати сервісів.

Суттєвим фактором також є втрата пакетів і затримки в передачі даних, що ускладнює коректну взаємодію між клієнтами та сервером. Такі проблеми можуть спричиняти збільшення часу відповіді, повторні запити або навіть повну неможливість обміну інформацією. У свою чергу, проблеми з DNS-сервісами можуть призводити до некоректного визначення адрес серверів, що унеможливлює доступ до ресурсів навіть за умови їх фізичної доступності.

Додатково варто враховувати перевантаження мережі, яке виникає при великій кількості одночасних запитів або обмеженій пропускній здатності каналу. Це призводить до зниження швидкості передачі даних і погіршення якості обслуговування користувачів. Не менш важливими є помилки в балансуванні трафіку, які можуть спричинити нерівномірний розподіл навантаження між серверами, що підвищує ризик перевантаження окремих вузлів і, як наслідок, їх відмови [8].

Надійність серверної системи формується під впливом комплексу взаємопов’язаних показників, кожен із яких характеризує окремий аспект її функціонування (рис. 1.2). Зокрема, безвідмовність і доступність визначають стабільність роботи системи, тоді як відмовостійкість і відновлюваність забезпечують її здатність протистояти збоям і швидко відновлюватися після них. Сукупний аналіз цих показників дозволяє об’єктивно оцінити рівень надійності серверної інфраструктури та обґрунтувати необхідність впровадження систем моніторингу.

Таблиця 1.2 – Основні показники надійності серверної системи

Показник	**Визначення**	**Вплив на систему**
Безвідмовність	Здатність системи працювати без збоїв протягом певного часу	Зменшує кількість відмов і простоїв
Доступність	Ступінь готовності сервера до обслуговування запитів користувачів	Забезпечує безперервність надання сервісів
Відмовостійкість	Здатність системи функціонувати при часткових відмовах компонентів	Підвищує стійкість до збоїв
Відновлюваність	Здатність системи швидко відновлювати працездатність після відмов	Скорочує час простою (downtime)
Продуктивність під навантаженням	Здатність системи ефективно працювати при високому навантаженні	Забезпечує стабільну роботу при пікових навантаженнях

1.3 Порівняльний аналіз підходів до моніторингу та реагування на інциденти

З урахуванням впливу численних факторів на надійність серверної інфраструктури особливої актуальності набуває використання систем моніторингу, які забезпечують постійне спостереження за станом серверів та їх ключових компонентів. Моніторинг дозволяє своєчасно виявляти відхилення у роботі системи, аналізувати поточні показники функціонування та оперативно реагувати на потенційні проблеми, запобігаючи розвитку критичних збоїв і відмов.

Сучасні підходи до моніторингу передбачають комплексний контроль різних аспектів функціонування серверної інфраструктури. Насамперед здійснюється моніторинг системних ресурсів, таких як завантаження центрального процесора, використання оперативної пам’яті, стан дискової підсистеми та мережева активність, що дозволяє оцінити загальний рівень навантаження на сервер. Водночас важливим є моніторинг прикладних сервісів, зокрема вебсерверів, баз даних, API та контейнеризованих застосунків, оскільки їх працездатність безпосередньо визначає доступність системи для користувачів.

Крім того, значну роль відіграє аналіз журналів подій, який дає змогу виявляти помилки, попередження та аномалії у роботі системи. У поєднанні з цим використовується моніторинг подій і механізми сповіщення, що забезпечують автоматичне інформування адміністратора про критичні ситуації. Не менш важливими є підходи, орієнтовані на оцінку продуктивності системи та контроль її доступності, що дозволяє визначати якість обслуговування користувачів і стабільність функціонування сервісів.

У сучасних умовах розвитку інформаційних технологій особливого значення набуває концепція спостережуваності (observability), яка базується на аналізі метрик (metrics), журналів (logs) та трасувань (traces). Такий підхід дозволяє отримати глибоке розуміння поведінки системи, швидко локалізувати проблеми та підвищити ефективність управління серверною інфраструктурою. Для

узагальнення основних підходів до моніторингу серверної інфраструктури доцільно представити їх у вигляді таблиці (табл. 1.3).

Таблиця 1.3 – Основні аспекти моніторингу серверної інфраструктури

Тип моніторингу	**Що відстежується**	**Основні показники**
Ресурсів сервера	CPU, RAM, диск, мережа	Завантаження CPU, використання RAM, I/O
Сервісів	Вебсервери, БД, API, контейнери	Статус сервісів, час відповіді
Логів	Журнали системи та застосунків	Помилки, попередження, аномалії
Подій і сповіщень	Критичні зміни стану системи	Алерти, тригери, повідомлення
Продуктивності	Швидкодія системи	Latency, throughput
Доступності	Доступність сервера	Uptime, ping, HTTP статус
Observability	Поведінка системи (metrics, logs, traces)	Метрики, трасування, кореляція подій

Сучасні системи моніторингу охоплюють широкий спектр параметрів функціонування серверної інфраструктури, що дозволяє комплексно оцінювати її стан та своєчасно виявляти потенційні загрози.

Для оцінки ефективності сучасних інструментів моніторингу доцільно провести їх порівняльний аналіз (табл. 1.4).

Таблиця 1.4 – Порівняльна характеристика інструментів моніторингу

Інструмент	**Основне призначення**	**Переваги**	**Недоліки**
Zabbix	Інфраструктурний моніторинг	Потужний функціонал, гнучкі тригери	Складність налаштування
Prometheus	Моніторинг метрик	Висока продуктивність, інтеграція з Kubernetes	Обмежена робота з логами
Grafana	Візуалізація даних	Зручні дашборди, інтеграція з різними системами	Не виконує збір даних самостійно

Продовження таблиці 1.4

Nagios	Класичний моніторинг	Простота, стабільність	Обмежена масштабованість
ELK Stack	Аналіз логів	Потужний аналіз подій	Високі вимоги до ресурсів
Datadog	Хмарний моніторинг	Простота використання, SaaS	Платний сервіс
Graylog	Централізований збір логів	Зручний інтерфейс, аналіз логів	Обмежений функціонал порівняно з ELK

Проведений аналіз показує, що кожен із розглянутих інструментів має свої особливості та сферу застосування, тому для побудови ефективної системи моніторингу доцільно використовувати їх у комплексі, поєднуючи можливості збору метрик, аналізу логів і візуалізації даних.

Висновок до розділу 1

У першому розділі було розглянуто теоретичні основи забезпечення надійності серверних систем у умовах тривалої експлуатації. Проаналізовано особливості функціонування серверів у безперервному режимі роботи, що дозволило визначити характерні проблеми, пов’язані з накопиченням навантаження, деградацією ресурсів та підвищенням ризику виникнення збоїв.

Досліджено основні фактори, що впливають на надійність серверної інфраструктури, зокрема апаратні, програмні, мережеві та організаційні. Встановлено, що їх комплексний вплив визначає стабільність функціонування системи, а також потребує постійного контролю та своєчасного реагування на відхилення. Окрему увагу приділено характеристикам надійності, таким як безвідмовність, доступність, відмовостійкість, відновлюваність і продуктивність під навантаженням, які дозволяють оцінити ефективність роботи серверів.

У ході аналізу сучасних підходів до моніторингу встановлено, що використання систем спостережуваності є ключовим інструментом забезпечення стабільності серверної інфраструктури. Розглянуто основні види моніторингу, а також проведено порівняльний аналіз сучасних інструментів, що підтвердило доцільність їх комплексного застосування для досягнення високого рівня надійності.

Результати проведеного дослідження обґрунтовують необхідність розробки ефективної системи моніторингу серверів, що дозволяє своєчасно виявляти проблеми, мінімізувати ризики відмов та забезпечити безперервність функціонування інформаційних систем. Отримані висновки слугують основою для подальшого проєктування системи контролю стану серверної інфраструктури, що буде розглянуто у наступному розділі.

2 МОДЕЛЮВАННЯ СИСТЕМИ КОНТРОЛЮ СТАНУ СЕРВЕРА

2.1 Формування функціональних вимог до системи моніторингу

Основною метою розроблюваної системи моніторингу є забезпечення безперервного контролю стану серверної інфраструктури з метою підтримання її стабільного та надійного функціонування в умовах тривалої експлуатації. Система повинна забезпечувати своєчасне виявлення відхилень у роботі серверів, що дозволяє оперативно реагувати на потенційні загрози та запобігати виникненню збоїв. Накопичення та збереження історичних даних щодо стану системи, що створює можливість для подальшого аналізу, прогнозування та оптимізації її роботи [9]. Крім того, система має реалізовувати механізми інформування адміністратора про критичні події та зміни у функціонуванні серверів, що сприяє підвищенню ефективності управління інфраструктурою та зменшенню часу реагування на інциденти.

Функціональні вимоги до системи моніторингу визначають основні можливості, які повинна реалізовувати система для забезпечення ефективного контролю стану серверної інфраструктури. Система повинна забезпечувати збір даних про рівень завантаження центрального процесора, що дозволяє оцінювати обчислювальне навантаження та виявляти перевантаження. Важливою складовою є відстеження використання оперативної пам’яті, що дає змогу своєчасно виявляти дефіцит ресурсів і запобігати зниженню продуктивності.

Система повинна контролювати рівень заповнення дискового простору, оскільки його нестача може призвести до порушення роботи сервісів та втрати даних. Не менш важливим є моніторинг мережевої активності, який дозволяє виявляти аномалії трафіку, перевантаження каналів зв’язку або можливі мережеві атаки. Система також повинна фіксувати стан критичних сервісів, таких як вебсервери, бази даних або API, що забезпечує контроль їх працездатності.

Необхідно приділити аналізу доступності сервера, який передбачає перевірку його досяжності у мережі та швидкості відповіді. Для подальшого аналізу та

побудови статистики система має забезпечувати збереження зібраних показників у базі даних. На основі цих даних повинні формуватися графіки та інші візуальні елементи, що відображають динаміку змін параметрів у часі.

Система повинна також реалізовувати механізми генерації сповіщень у разі перевищення встановлених порогових значень показників, що дозволяє оперативно реагувати на критичні ситуації. Додатково передбачається ведення журналу подій для фіксації всіх змін, інцидентів і дій системи. Завершальним елементом є відображення поточного стану серверної інфраструктури у зручному користувацькому інтерфейсі, що забезпечує наочність та простоту управління.

Наведена табл. 2.1 відображає ключові функціональні вимоги до системи моніторингу серверної інфраструктури. Кожна вимога має унікальне позначення, що спрощує її ідентифікацію під час проєктування та реалізації системи. Сукупність представлених функцій забезпечує комплексний підхід до контролю стану серверів, включаючи збір, обробку, збереження та візуалізацію даних, а також своєчасне інформування адміністратора про критичні події. Це дозволяє підвищити рівень надійності, доступності та ефективності функціонування інформаційної системи.

Таблиця 2.1 – Функціональні вимоги до системи моніторингу серверів

№	Позначення	Функціональна вимога	Опис
1	FR1	Моніторинг CPU	Збір та аналіз даних про завантаження процесора в реальному часі
2	FR2	Моніторинг оперативної пам’яті	Відстеження використання RAM та виявлення нестачі ресурсів
3	FR3	Контроль дискового простору	Перевірка заповнення дисків та попередження про критичні значення
4	FR4	Моніторинг мережі	Аналіз мережевої активності, трафіку та виявлення аномалій
5	FR5	Контроль сервісів	Перевірка стану критичних сервісів (вебсервер, БД, API, контейнери)
6	FR6	Перевірка доступності	Визначення доступності сервера (ping, HTTP-відповідь, uptime)

Продовження таблиці 2.1

7	FR7	Збір та збереження даних	Запис показників у базу даних для подальшого аналізу
8	FR8	Візуалізація даних	Побудова графіків і діаграм для відображення змін показників
9	FR9	Система сповіщень	Генерація алертів при перевищенні порогових значень
10	FR10	Журнал подій	Фіксація подій, помилок і змін у системі
11	FR11	Інтерфейс користувача	Відображення поточного стану системи у зручному UI

Нефункціональні вимоги визначають якісні характеристики системи моніторингу, що впливають на ефективність її використання, стабільність роботи та можливість масштабування. Система повинна створювати мінімальне навантаження на серверні ресурси, щоб не впливати негативно на продуктивність основних сервісів. Оперативність збору даних є критичною для своєчасного виявлення проблем та реагування на них. Крім того, важливою є зручність використання, що передбачає наявність інтуїтивно зрозумілого інтерфейсу для адміністратора. Система повинна підтримувати можливість інтеграції з іншими програмними рішеннями, такими як системи логування, аналітики або управління інцидентами (табл. 2.2).

Таблиця 2.2 – Нефункціональні вимоги до системи

№	**Позначення**	**Нефункціональна вимога**	**Опис**
1	NFR1	Надійність	Забезпечення стабільної роботи системи без збоїв протягом тривалого часу
2	NFR2	Масштабованість	Можливість розширення системи при збільшенні кількості серверів або даних
3	NFR3	Мінімальне навантаження	Оптимізація використання ресурсів для зменшення впливу на продуктивність сервера

Продовження таблиці 2.2

4	NFR4	Оперативність	Швидкий збір, обробка та передача даних у реальному часі
5	NFR5	Зручність використання	Інтуїтивно зрозумілий інтерфейс користувача
6	NFR6	Безпека доступу	Захист системи за допомогою механізмів автентифікації та авторизації
7	NFR7	Інтеграція	Можливість взаємодії з іншими системами (API, логування, SIEM)
8	NFR8	Стійкість до збоїв	Забезпечення безперервної роботи при часткових відмовах компонентів

Нефункціональні вимоги визначають якість та ефективність функціонування системи моніторингу, доповнюючи її функціональні можливості. Їх дотримання забезпечує не лише стабільну роботу системи, а й її адаптивність до змін навантаження, безпечність та зручність використання в умовах реальної експлуатації.

2.2 Побудова логічної моделі системи збору та обробки даних

Важливою складовою інтелектуальної системи моніторингу є модуль автоматизованого виявлення аномальних станів, який відіграє ключову роль у забезпеченні стабільності функціонування серверної інфраструктури. Під аномаліями в даному контексті розуміються відхилення від нормального режиму роботи системи, що проявляються у вигляді нетипових змін параметрів. Такі відхилення можуть свідчити про наявність технічних несправностей, зниження продуктивності або потенційні загрози безпеці.

Своєчасне виявлення аномальних подій є критично важливим, оскільки дає змогу оперативно реагувати на проблеми та запобігати негативним наслідкам, таким як зупинка сервісів, втрата інформації або порушення умов надання послуг. У традиційних системах моніторингу зазвичай застосовуються підходи, засновані на правилах, що передбачають використання фіксованих порогових значень для визначення відхилень. Наприклад, перевищення певного рівня завантаження

процесора може інтерпретуватися як критичний стан [16].

Проте подібні методи мають низку суттєвих недоліків. Насамперед, вони не враховують контекст роботи системи: однакові показники можуть бути нормальними в одних умовах і критичними - в інших. Крім того, rule-based підходи не здатні ефективно виявляти складні аномалії, що виникають у результаті взаємодії кількох параметрів. Також значною проблемою є необхідність ручного налаштування порогів, що часто призводить або до надмірної кількості хибних спрацювань, або до пропуску дійсно небезпечних ситуацій.

З метою усунення зазначених обмежень у розробленій системі застосовано підхід, заснований на використанні методів машинного навчання. Його особливість полягає у здатності системи аналізувати значні обсяги історичних даних, визначати характерні закономірності функціонування та формувати уявлення про нормальний стан без необхідності жорсткого задання порогових значень. У разі, якщо нові дані суттєво відхиляються від сформованої моделі нормальної поведінки, система ідентифікує їх як аномальні.

Застосування такого підходу забезпечує адаптивність системи до змін умов експлуатації, дозволяє виявляти приховані або складні відхилення, а також автоматично оновлювати модель у разі зміни характеристик даних. Крім того, використання технологій штучного інтелекту надає системі проактивних властивостей, що дозволяє попереджати виникнення критичних ситуацій ще до їх реального прояву [16, 17].

У розробленій системі для виявлення аномальних станів використано комплекс сучасних алгоритмів машинного навчання, кожен з яких виконує окрему функцію в процесі аналізу даних та підвищує загальну точність детектування відхилень.

Основним алгоритмом обрано метод Isolation Forest, який ґрунтується на принципі випадкового розбиття простору ознак і побудові так званих дерев ізоляції. Його перевагою є висока швидкодія, ефективність при роботі з великими обсягами даних та здатність виявляти аномалії без попереднього маркування вибірки.

Додатково у системі використано алгоритм One-Class SVM, який формує межу, що відокремлює нормальні дані від аномальних, дозволяючи виявляти відхилення на основі геометричних властивостей вибірки. Також застосовується метод k-Nearest Neighbors (kNN), який визначає ступінь аномальності об’єкта шляхом аналізу відстані до найближчих сусідніх точок. Чим більша ця відстань, тим вища ймовірність, що об’єкт є аномальним.

Окрім класичних алгоритмів, у системі реалізовано підхід на основі глибинного навчання - AutoEncoder, який представляє собою нейронну мережу, здатну відновлювати вхідні дані. Значне відхилення між початковими та реконструйованими значеннями використовується як ознака аномалії, що дозволяє ефективно виявляти складні нелінійні залежності у даних.

Для забезпечення збереження та обробки даних моніторингу використовується база даних SQLite, яка виконує функцію централізованого сховища метрик. Її структура спроєктована таким чином, щоб гарантувати швидкий доступ до актуальних показників та можливість проведення ретроспективного аналізу за тривалий проміжок часу. Ключовим елементом бази даних є таблиця metrics, призначена для зберігання потокових значень параметрів системи. Вона містить такі поля [17]:

– id - унікальний ідентифікатор запису;
– timestamp - час фіксації метрики;
– server_ip - IP-адреса сервера;
– cpu_usage - рівень завантаження процесора;
– memory_usage - використання оперативної пам’яті;
– disk_usage - ступінь заповнення дискового простору;
– network_io - показники мережевої активності.

Окрім основної таблиці, у базі даних передбачено додаткові сутності, які забезпечують функціонування системи:

– users - таблиця для зберігання облікових записів користувачів. Вона містить ідентифікатор, електронну пошту (що використовується як логін) та хеш пароля для забезпечення безпечної автентифікації.

– servers - таблиця, що зберігає інформацію про сервери, підключені до системи моніторингу. Вона включає унікальний ідентифікатор, назву сервера, IP-адресу, облікові дані для підключення (у захищеному вигляді), а також дату додавання до системи.

Усі таблиці бази даних логічно взаємопов’язані між собою за допомогою механізму зовнішніх ключів, що забезпечує узгодженість та цілісність збереженої інформації. Така організація структури дозволяє ефективно виконувати складні SQL-запити, необхідні для проведення аналітичної обробки даних та формування звітності.

Проєктування бази даних здійснювалося з урахуванням принципів нормалізації, що дозволило мінімізувати дублювання інформації та підвищити ефективність її обробки. Крім того, особлива увага була приділена оптимізації запитів, що є критично важливим при роботі з великими обсягами моніторингових даних.

Архітектура сховища також передбачає можливість подальшого масштабування. Зокрема, у разі зростання навантаження або обсягів даних система може бути адаптована до використання більш потужних систем керування базами даних, таких як PostgreSQL або MySQL, без суттєвих змін у логіці її функціонування.

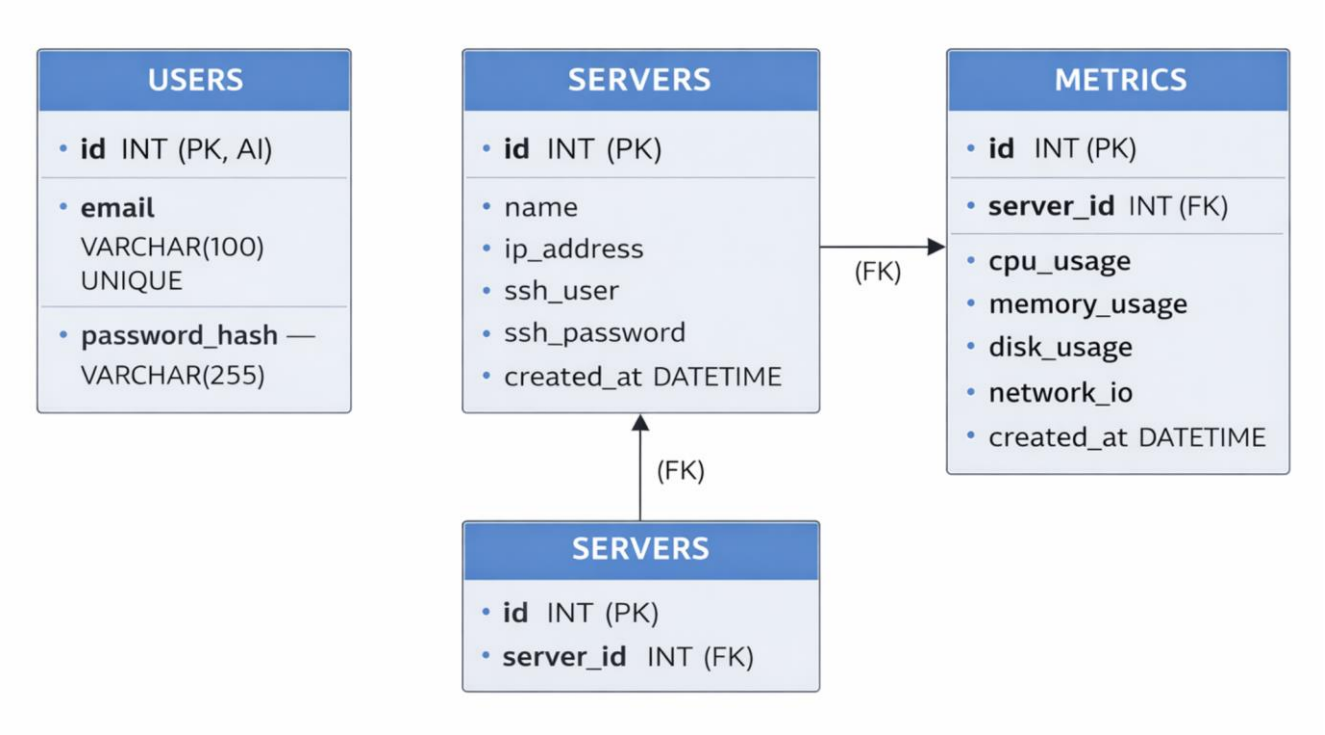


Рисунок 2.1 – Структура бази даних системи моніторингу

Перед подачею даних до моделей машинного навчання виконується етап їх попередньої обробки. На цьому етапі здійснюється очищення даних від пропущених значень, їх нормалізація з метою приведення до єдиного масштабу, а також виявлення та усунення аномальних викидів, що можуть спотворювати результати аналізу.

```
from pyod.models.iforest import IForest
model = IForest()
model.fit(train_data) # навчання на історичних метриках
predictions = model.predict(new_data) # 0 – норма, 1 – аномалія
```

Рисунок 2.2 – Реалізація навчання моделі Isolation Forest

Модель Isolation Forest характеризується високою масштабованістю, не потребує складного налаштування гіперпараметрів та забезпечує швидку обробку навіть значних обсягів даних. Після виявлення аномальних станів результати аналізу передаються до підсистеми оповіщення - зокрема через Telegram-бот, електронну пошту або Slack.

Важливою особливістю реалізації є гнучка архітектура AI-модуля, яка дозволяє адаптувати систему до змін навантаження та різних сценаріїв використання. У системі передбачено механізм динамічного вибору алгоритмів виявлення аномалій залежно від типу даних або характеру навантаження. Наприклад, алгоритм Isolation Forest застосовується для загального аналізу, тоді як AutoEncoder використовується у випадках складних багатофакторних залежностей. Такий підхід сприяє підвищенню точності та гнучкості аналізу.

Крім того, реалізовано механізм періодичного перенавчання моделей, який автоматично активується при накопиченні нових даних або зміні конфігурації серверної інфраструктури. Для цього створено окремий підмодуль, що відстежує зміни у розподілі даних, аналізує їх динаміку та ініціює процес перенавчання без втручання адміністратора. Це дозволяє підтримувати моделі у актуальному стані та уникати втрати їх ефективності з часом.

Оцінювання якості роботи моделей здійснюється за допомогою стандартних метрик - precision, recall, F1-score та ROC AUC. Зазначені показники дозволяють визначити точність класифікації, а також оцінити рівень хибнопозитивних і хибнонегативних спрацьовувань.

Для перевірки ефективності алгоритмів у систему інтегровано спеціальний модуль тестування, який дозволяє формувати та виконувати різні валідаційні сценарії. Зокрема, передбачена можливість моделювання штучних аномалій - таких як різке зростання навантаження на процесор, переповнення дискового простору або нестабільність мережевого трафіку. Це дає змогу оцінити чутливість моделей та, за необхідності, скоригувати параметри реагування або обрати більш відповідний алгоритм. Завдяки такій архітектурі система підтримує використання різних моделей для окремих серверів або груп метрик, що забезпечує масштабованість і дозволяє реалізувати індивідуалізований підхід до моніторингу.

Використання методів машинного навчання в модулі виявлення аномалій суттєво розширює можливості автоматизації контролю стану серверної інфраструктури. Завдяки здатності моделей аналізувати складні поведінкові закономірності, адаптуватися до змін середовища та функціонувати в режимі

реального часу забезпечується високий рівень інтелектуалізації процесу моніторингу.

На відміну від традиційних підходів, моделі не обмежуються лише фіксацією відхилень, а дозволяють виявляти приховані тенденції, що можуть передувати виникненню критичних ситуацій. Проактивний характер роботи системи досягається за рахунок можливості прогнозування потенційних інцидентів ще до їх фактичного виникнення. Зокрема, аналіз послідовності незначних відхилень у показниках використання процесора або оперативної пам’яті дає змогу передбачити подальше зростання навантаження, що може призвести до деградації продуктивності або збою системи.

Отримані прогностичні результати дозволяють своєчасно ініціювати реакцію - як з боку адміністратора, так і через автоматизовані сценарії, мінімізуючи вплив на кінцевих користувачів і запобігаючи порушенню роботи сервісів.

Значною перевагою впровадження такого підходу є зменшення впливу людського фактора. Це досягається шляхом автоматизації всіх ключових етапів - збору, обробки, аналізу та інтерпретації даних. Система здатна самостійно реагувати на події, генерувати сповіщення, вести журнал подій, виконувати перенавчання моделей та адаптувати власну конфігурацію відповідно до накопиченого досвіду [19].

2.3 Розробка механізмів діагностики та попередження збоїв

Надійна система моніторингу не може ефективно функціонувати без оперативного, стійкого та гнучкого механізму інформування про інциденти. Будь-яка затримка в передачі повідомлень між системою моніторингу, адміністратором або автоматизованими засобами реагування здатна спричинити негативні наслідки - від зниження продуктивності до втрати даних чи порушення безперервності критично важливих бізнес-процесів. Саме з цієї причини модуль автоматичного сповіщення було закладено як одну з базових складових архітектури інтелектуальної системи моніторингу серверної інфраструктури.

Головне призначення цього модуля полягає у забезпеченні мінімального часового інтервалу між моментом виявлення небезпечної події та передачею інформації відповідальній особі або підсистемі, яка повинна виконати подальше реагування. У межах розробленої системи реалізовано багаторівневу, масштабовану та відмовостійку інфраструктуру оповіщення, яка забезпечує автоматичне формування повідомлень, їх адаптацію до конкретного сценарію, інтелектуальний вибір каналу доставки та взаємодію із зовнішніми сервісами реагування.

Важливою перевагою запропонованого підходу є реалізація механізму фільтрації повідомлень. Сповіщення обробляються з урахуванням рівня критичності події, категорії серверів, типу інциденту та часових параметрів. Такий підхід дозволяє уникнути надмірної кількості другорядних повідомлень, зменшує інформаційне перевантаження адміністраторів та забезпечує концентрацію уваги на дійсно важливих подіях [19, 20].

Крім того, у системі передбачено механізм підтвердження доставки сповіщень. Його застосування дає можливість відстежувати факт отримання повідомлення та контролювати подальшу реакцію. У випадку, якщо повідомлення не було підтверджене або залишилося без реагування, система повторно ініціює сигнал тривоги, використовуючи резервний або альтернативний канал зв’язку. Це суттєво підвищує надійність функціонування підсистеми попередження.

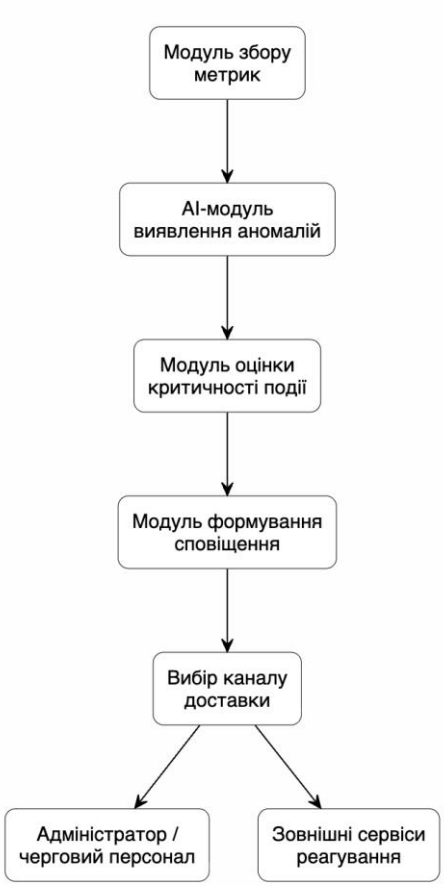


Рисунок 2.3 – Загальна схема роботи модуля автоматичного сповіщення

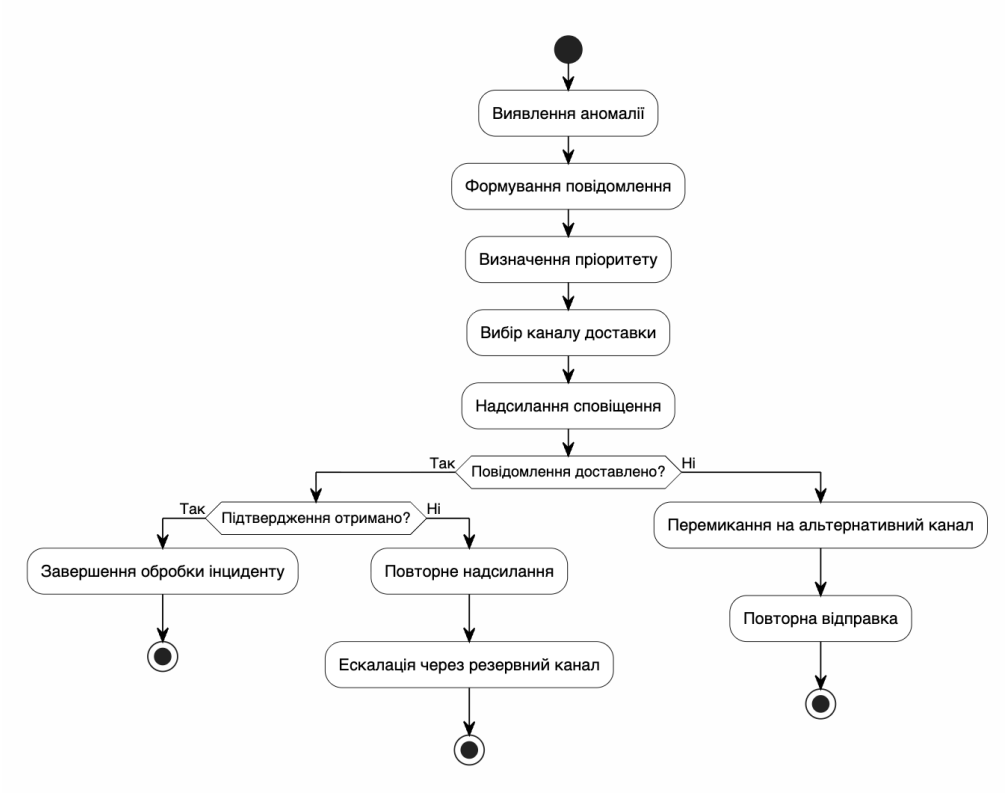


Рисунок 2.4 – Логіка обробки та ескалації сповіщення

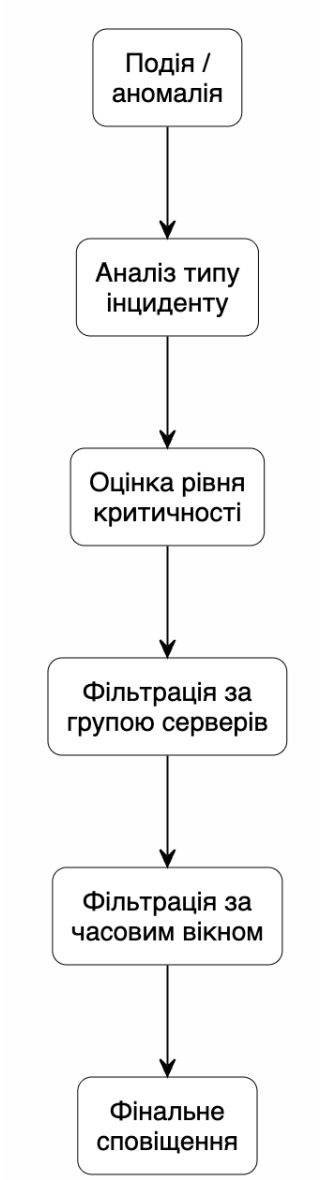


Рисунок 2.5 – Фільтрація сповіщень у системі моніторингу

Основу механізму автоматичного сповіщення становить тригерна логіка, яка активується у момент виявлення аномалії за допомогою AI-моделі. Після класифікації нових метрик як аномальних запускається послідовність обробки інциденту [21].

Насамперед виконується фіксація події у базі даних у таблиці alerts. Для кожного інциденту зберігається набір параметрів, що дозволяє не лише зафіксувати факт відхилення, а й забезпечити подальший аналіз та відстеження. Зокрема, запис містить:

- унікальний ідентифікатор події;
- ідентифікатор сервера, пов’язаний із таблицею servers;
- тип метрики, що спричинила спрацювання;
- значення показника, яке перевищило допустимий рівень;
- часову мітку події;
- рівень критичності - info, warning або critical;
- статус обробки - new, acknowledged або resolved.

Застосування такої структури дозволяє організувати повноцінний журнал інцидентів, що забезпечує можливість їх подальшого аналізу у динаміці.

Після реєстрації події система визначає канал сповіщення відповідно до налаштувань. Для цього використовується параметр notification_channel, який задається для кожного сервера або всієї системи. Він може приймати значення telegram, email, webhook або multi. На основі цього параметра обирається відповідний спосіб передачі повідомлення з урахуванням пріоритету події, затримки відправлення та шаблону формування тексту.

Додатково реалізовано механізм гнучкої маршрутизації повідомлень, який враховує часові обмеження, рівень критичності та можливість автоматичного перемикання каналу у разі виникнення помилки під час доставки.

Безпосередня передача повідомлень здійснюється автоматично відповідно до обраного каналу. Для Telegram використовується API-виклик із передачею chat_id, для електронної пошти - SMTP-протокол із формуванням теми, тексту та, за потреби, вкладень, таких як лог-файли або структуровані дані. У випадку webhook виконується HTTP POST-запит із передачею даних у визначеному форматі [21].

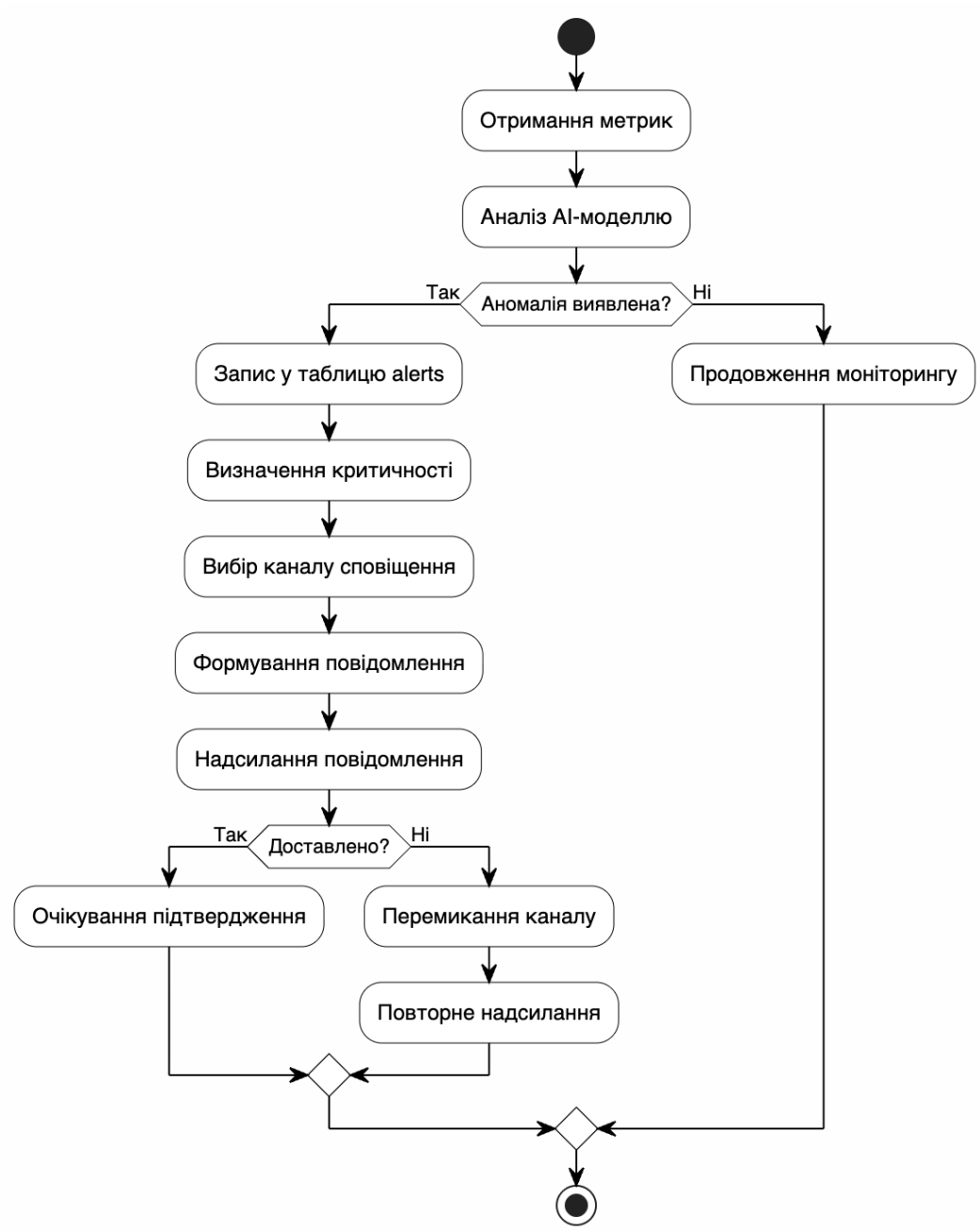


Рисунок 2.6 – Логіка роботи тригерного механізму сповіщення

Наведена схема демонструє, що процес сповіщення є багатоступеневим і включає не лише фіксацію події, але й аналіз її критичності, вибір каналу передачі, контроль доставки та повторну обробку у разі виникнення помилок.

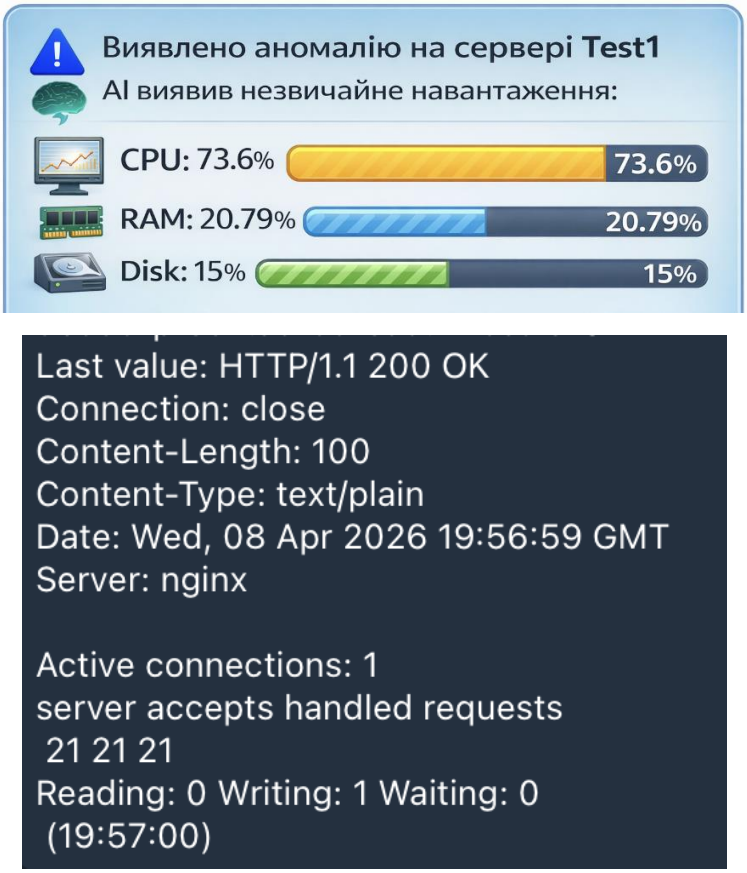


Рисунок 2.7 – Повідомлення в телеграм

Можливості розширення

Архітектура підсистеми сповіщення спроєктована з урахуванням подальшого масштабування та інтеграції нових каналів взаємодії. У перспективі передбачено можливість підключення додаткових засобів передачі повідомлень, зокрема:

- електронної пошти через SMTP із підтримкою шаблонів повідомлень;
- Slack через Webhook API;
- SMS або голосових викликів за допомогою сервісів Twilio або Infobip;
- Webhook-запитів для автоматичного запуску сценаріїв реагування.

Окремим напрямом розвитку є реалізація механізму так званої «розумної реакції», який передбачає автоматичне виконання дій у відповідь на виявлені аномалії. До таких дій можуть належати [22]:

- очищення кешу системи;
- перезапуск служб через SSH;

– тимчасове обмеження мережевого доступу у разі виявлення підозрілої активності.

Таким чином, підсистема сповіщення трансформується з простого інструменту інформування у повноцінний механізм активного впливу на стан серверної інфраструктури. У разі виникнення інциденту система не лише реєструє подію, а й автоматично ініціює відповідні дії - у вигляді скриптів, REST-запитів або заздалегідь визначених сценаріїв реагування.

Наприклад, при фіксації критичного навантаження на процесор може бути запущено процедуру перезапуску сервісу, очищення тимчасових файлів або повідомлення зовнішньої системи балансування навантаження. Важливою особливістю є інтеграція з модулем самонавчання, що дозволяє системі адаптувати свою поведінку залежно від накопиченого досвіду.

Зокрема, у випадку повторюваних інцидентів одного типу система може автоматично змінювати політику реагування або підвищувати рівень критичності сповіщень. Такий підхід забезпечує еволюційний розвиток механізмів реагування, коли кожна наступна подія аналізується з урахуванням попередніх.

У поєднанні з механізмами маршрутизації повідомлень, контролю доставки, журналювання подій та можливістю масштабування запропонована система забезпечує високий рівень готовності до інцидентів. Вона виступає не лише завершальним етапом процесу виявлення аномалій, а й базою для побудови автономних самокерованих IT-середовищ, у яких роль людини поступово зводиться до наглядових і коригуючих функцій [23].

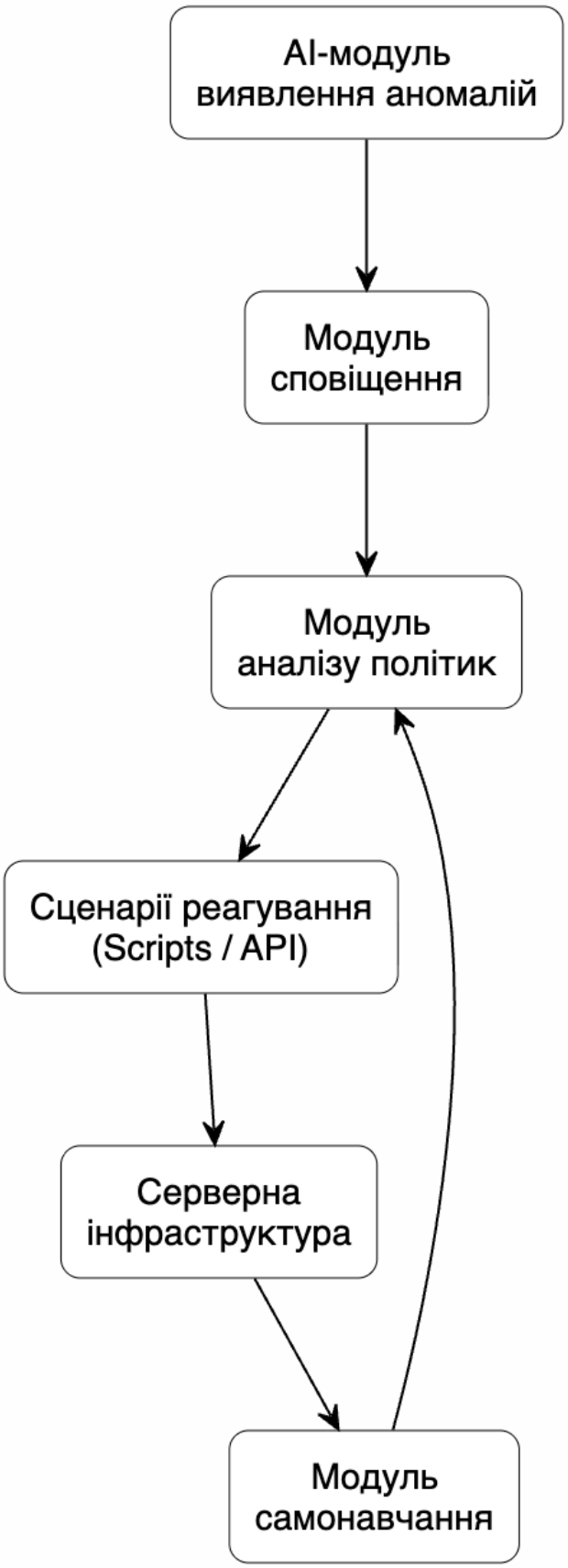


Рисунок 2.8 – Архітектура автоматичної реакції системи

Висновок до розділу 2

У другому розділі було розглянуто процес проєктування та реалізації інтелектуальної системи моніторингу серверної інфраструктури з використанням методів машинного навчання. Проведено аналіз функціональних вимог до системи, визначено її архітектурні компоненти та описано логіку взаємодії основних модулів.

Особливу увагу приділено розробці підсистеми виявлення аномалій, яка ґрунтується на застосуванні сучасних алгоритмів, зокрема Isolation Forest, One-Class SVM, kNN та AutoEncoder. Обґрунтовано доцільність використання комбінованого підходу, що дозволяє підвищити точність виявлення відхилень та адаптувати систему до різних типів навантаження.

У ході роботи спроєктовано структуру бази даних, яка забезпечує ефективне зберігання, обробку та аналіз потокових метрик. Реалізовано механізми попередньої обробки даних, що включають нормалізацію, фільтрацію та усунення викидів, що сприяє підвищенню якості роботи моделей.

Також розроблено модуль автоматичного сповіщення, який забезпечує оперативну передачу інформації про інциденти через різні канали зв’язку. Реалізовано механізми маршрутизації повідомлень, контролю доставки, ескалації та фільтрації за пріоритетом, що дозволяє зменшити навантаження на адміністратора та підвищити ефективність реагування.

Крім того, впроваджено підхід до автоматичного реагування на інциденти, який передбачає запуск відповідних сценаріїв у разі виявлення аномалій, а також механізм перенавчання моделей, що забезпечує адаптивність системи до змін у середовищі.

3 РЕАЛІЗАЦІЯ СИСТЕМИ ТА ОЦІНКА ЇЇ ЕФЕКТИВНОСТІ

3.1 Розробка програмного рішення для моніторингу сервера

Для створення інтелектуальної системи моніторингу серверної інфраструктури було обрано сучасний технологічний стек, який забезпечує надійність, гнучкість та високу продуктивність програмного рішення. При розробці системи враховувалися вимоги до безперервного збору метрик, оперативного реагування на інциденти, можливості подальшого масштабування та інтеграції з зовнішніми сервісами.

Особлива увага приділялася універсальності розгортання системи. Програмне рішення спроєктовано таким чином, щоб його можна було використовувати як у локальному середовищі, так і в хмарній інфраструктурі - зокрема на VPS або у контейнеризованому середовищі (Docker). Це дозволяє адаптувати систему до різних умов експлуатації без необхідності суттєвих змін у її архітектурі.

Обраний стек технологій орієнтований на досягнення таких ключових характеристик:

– масштабованість - можливість підключення нових серверів без перебудови системи;

– гнучкість інтеграції - підтримка REST API та взаємодія з Telegram, поштовими сервісами й сторонніми API;

– низьке споживання ресурсів - що є важливим при використанні на обмежених обчислювальних середовищах;

– простота розгортання - використання віртуального середовища Python, автоматизованих сценаріїв запуску та CLI-інструментів.

У межах даного підрозділу розглянуто основні складові технологічного стеку, середовище виконання, використані бібліотеки та фреймворки, типи мережевих взаємодій, а також принципи реалізації логування та забезпечення безпеки [24, 25].

Логіка функціонування системи побудована у вигляді послідовного циклу обробки даних, що включає такі етапи: збір даних - попередня обробка - аналіз - прийняття рішення - реагування - сповіщення - збереження та архівація інформації. Кожен компонент системи реалізовано з урахуванням принципів модульності, незалежності та можливості повторного використання.

Для реалізації серверної частини використано фреймворк Flask, який забезпечує створення REST API із мінімальними накладними витратами. Це дозволяє легко масштабувати систему або інтегрувати її з іншими програмними рішеннями.

Збір системних метрик реалізовано за допомогою бібліотеки psutil, яка забезпечує швидкий доступ до показників використання процесора, оперативної пам’яті, дискових ресурсів і мережевої активності без необхідності встановлення додаткових агентів. Це спрощує розгортання системи та знижує навантаження на інфраструктуру.

Підсистема логування побудована із використанням стандартної бібліотеки Python logging. Вона передбачає класифікацію повідомлень за рівнями важливості - INFO, WARNING, ERROR, CRITICAL, що дозволяє ефективно структурувати події. Логи зберігаються у базі даних SQLite, що забезпечує можливість подальшого аналізу, побудови часових залежностей та формування звітності.

Інтеграція з Telegram API дозволила реалізувати як механізм надсилання сповіщень (push), так і зворотну взаємодію з користувачем (pull) - зокрема отримання інформації про стан системи, журнали подій, uptime та інші параметри. Це забезпечує повноцінний двосторонній зв’язок між системою та адміністратором.

Реалізований Telegram-бот підтримує обмеження доступу за ідентифікатором користувача, перевірку автентичності та ведення журналу дій, що підвищує рівень безпеки та контроль доступу до системи. Окрім цього, впроваджено механізм Webhook-повідомлень, який забезпечує передачу критичних інцидентів до зовнішніх сервісів - таких як CRM, ERP або системи автоматизації DevOps.

Усі зовнішні запити реалізовані за допомогою бібліотеки requests із використанням тайм-аутів та обробкою помилок, що дозволяє підвищити надійність мережевої взаємодії. Застосовані технологічні рішення забезпечують стабільну, продуктивну та безпечну роботу системи, а також спрощують процес її розгортання та адаптації до різних інфраструктур. Завдяки модульному підходу можливе впровадження як повної системи, так і окремих її компонентів у вже існуюче середовище [25].

Мови програмування

У процесі розробки програмного рішення використано такі мови програмування:

– Python - основна мова реалізації бекенд-логіки, обробки даних, взаємодії з API, побудови AI-модулів та системи сповіщення;

– Bash - для створення сценаріїв автоматичного запуску, виконання команд через SSH та контролю системних параметрів;

– JavaScript (з використанням бібліотеки jQuery) - для реалізації динамічних елементів користувацького інтерфейсу.

Фреймворки та бібліотеки

Для забезпечення функціонування системи використано такі інструменти:

– Flask - мікрофреймворк для розробки REST API та серверної частини застосунку, що забезпечує маршрутизацію запитів і взаємодію з базою даних;

– psutil - бібліотека для збору системних метрик, зокрема показників CPU, оперативної пам’яті, дискових ресурсів і мережі;

– scikit-learn - використовується для реалізації алгоритмів машинного навчання та виявлення аномалій;

– pandas, numpy - застосовуються для обробки, аналізу та зберігання часових рядів і логів системи;

– SQLAlchemy - ORM-засіб для зручної роботи з базою даних;

– reportlab - бібліотека для формування PDF-звітів про стан інфраструктури;

– requests - використовується для взаємодії з зовнішніми API, включаючи

Telegram та webhook-сервіси.

База даних

У якості системи керування базами даних використано SQLite, яка застосовується для локального зберігання інформації про сервери, журнали подій, попередження, статуси та інші службові дані. Завдяки простоті налаштування, відсутності необхідності у виділеному сервері та підтримці SQL, дане рішення є оптимальним для систем із середнім рівнем навантаження.

Середовище розгортання

Розгортання програмного рішення здійснюється у середовищі Linux, що забезпечує стабільність і ефективність роботи системи. Основні компоненти середовища включають:

– сервер на базі Ubuntu 22.04 із попередньо встановленим Python 3.10, віртуальним середовищем (venv) та інструментами системного моніторингу (htop, netstat). Flask-застосунок функціонує на порту 5000;

– Telegram-бот, який забезпечує взаємодію з користувачем і виконує функції сповіщення адміністраторів. Бот працює через Telegram Bot API та підтримує команди для отримання інформації про стан системи, журналів подій та виконання службових операцій;

– планувальник завдань crontab, який використовується для періодичного запуску аналітичних процедур і перевірок стану системи у фоновому режимі.

Обраний технологічний стек дозволив реалізувати модульну та масштабовану систему з низьким порогом входження для подальшого супроводу. Рішення відповідає сучасним підходам DevOps, підтримує інтеграцію з AI-модулями, забезпечує безперервність роботи (Zero Downtime) та орієнтоване на принципи постійного вдосконалення.

Кожен компонент системи виконує визначену функцію в межах загальної архітектури, що значно спрощує процес масштабування та оновлення. Завдяки цьому можливе додавання нових функцій без внесення змін до всієї системи - наприклад, розширення аналітики або впровадження додаткових типів сповіщень.

Фреймворк Flask забезпечує ефективну обробку HTTP-запитів із мінімальними затримками та підтримує багатопотокову обробку через WSGI. Інтеграція з сервером застосунків gunicorn дозволяє розподіляти навантаження між декількома воркерами, що підвищує продуктивність системи.

Застосування бібліотеки psutil дозволило отримувати детальні системні метрики, зокрема [26]:

- середнє та пікове навантаження процесора;
- використання оперативної та swap-пам’яті;
- кількість активних мережевих з’єднань;
- показники мережевого трафіку;
- статистику операцій введення-виведення дисків.

Зібрані дані формують основу для подальшого машинного аналізу, зокрема побудови часових залежностей, виявлення аномалій та прогнозування потенційних збоїв у роботі системи.

Інтеграція з Telegram API забезпечила високий рівень гнучкості процесу адміністрування системи. Адміністратор має можливість оперативно отримувати інформацію про стан серверів, переглядати журнали подій, ініціювати перезапуск окремих компонентів або формувати звітність у форматі PDF. Управління системою здійснюється через зручний інтерфейс бота, що значно спрощує взаємодію з інфраструктурою.

Безпека доступу реалізована за допомогою обмеження за Telegram ID, а також використання конфігураційних змінних для зберігання токенів у окремому середовищі (.env). Такий підхід дозволяє мінімізувати ризики несанкціонованого доступу та забезпечує захист критичних даних.

Застосування бази даних SQLite дозволяє зберігати інформацію про всі інциденти, попередження та виконані сценарії у структурованому вигляді із часовими мітками. Це створює основу для подальшого аналізу, побудови візуалізацій та аудиту роботи системи. Зокрема, передбачена можливість інтеграції з інструментами моніторингу, такими як Grafana, для створення дашбордів і відображення ключових метрик у реальному часі.

Генерація звітів у форматі PDF реалізована на основі шаблонного підходу, що дозволяє динамічно формувати документи із включенням актуальної дати, ідентифікатора сервера, узагальнених показників та статусів системи. Це забезпечує зручність у використанні звітності для аналізу та прийняття рішень.

Розроблена система також адаптована для роботи в хмарних середовищах - зокрема на платформах AWS EC2 та Azure Virtual Machines. Вона підтримує використання конфігураційних змінних для контейнеризованих середовищ і може бути інтегрована у CI/CD процеси з використанням таких інструментів, як GitHub Actions або GitLab CI. Це значно спрощує процес розгортання, оновлення та масштабування системи.

У перспективі передбачено розвиток архітектури шляхом переходу до мікросервісного підходу, що передбачає розподіл функціональності на окремі незалежні сервіси - зокрема модулі моніторингу, аналітики, сповіщення та генерації звітів [26, 27].

3.2 Реалізація підсистеми візуалізації та сповіщення про події

У даному підрозділі наведено практичні сценарії використання розробленої інтелектуальної системи моніторингу, які демонструють її ефективність у реальних умовах експлуатації. Представлені кейси підтверджують здатність системи не лише фіксувати аномальні відхилення у роботі інфраструктури, але й виконувати їх автоматичний аналіз, оцінювати ризики виникнення збоїв на основі накопичених історичних даних, а також здійснювати своєчасне інформування відповідальних осіб через різні канали зв’язку.

Особливістю системи є можливість виявлення як різких змін показників, так і поступових деградацій, які часто залишаються непомітними при використанні традиційних засобів моніторингу. Це є особливо важливим для довготривалого функціонування серверної інфраструктури без постійного контролю з боку адміністратора.

Тестування системи проводилося у контрольованому середовищі із

використанням реальних серверів, що дозволило максимально наблизити умови експлуатації до практичних сценаріїв малого та середнього бізнесу. Збір системних метрик здійснюється з інтервалом у 60 секунд, що забезпечує достатній рівень деталізації для виявлення як короткочасних пікових навантажень, так і довготривалих змін у поведінці системи [27].

Аналітична обробка даних виконується у режимі реального часу із використанням прогнозного вікна до 10 хвилин. Це дозволяє не лише реагувати на вже виниклі проблеми, а й попереджати потенційні перевантаження та збої до моменту їх фактичного прояву.

Інтегрована система візуалізації забезпечує зручний багаторівневий аналіз даних через вебінтерфейс - включаючи графіки, гістограми, фільтри, вибір часових інтервалів та можливість експорту результатів у форматі PNG. Це значно полегшує процес аналізу та підготовки звітності.

Паралельно із цим модуль сповіщень через Telegram виконує функцію оперативного інформування, що дозволяє адміністраторам швидко реагувати на інциденти навіть без доступу до основного інтерфейсу системи.

Поєднання аналітичних можливостей та зручного інтерфейсу робить систему не лише інструментом технічного моніторингу, але й засобом підтримки управлінських рішень. На основі отриманих даних стає можливим:

- перерозподіл навантаження між серверами;
- оптимізація використання ресурсів;
- впровадження механізмів кешування;
- модернізація апаратних компонентів на основі об’єктивної статистики.

Кейс 1 - Виявлення періодичних пікових навантажень на CPU

У процесі моніторингу серверу Test1 (95.164.23.151) було виявлено регулярні пікові навантаження на центральний процесор, значення яких досягали 90–95%. Аналіз зібраних метрик показав, що такі сплески повторюються із періодичністю приблизно один раз на годину. Подібна поведінка системи може свідчити про виконання фонових процесів або запланованих завдань, які не були оптимізовані та спричиняють надмірне використання ресурсів. Інтелектуальна система

моніторингу автоматично зафіксувала дану аномалію, проаналізувала її періодичність та сформувала графік зміни навантаження процесора у часі. На основі отриманих даних було згенеровано сповіщення, яке оперативно передано адміністратору через Telegram-бот.



Рисунок 3.1 – Динаміка завантаження CPU на сервері Test1

Кейс 2 - Виявлення критичного перевантаження системних ресурсів

У ході моніторингу було зафіксовано критичний стан серверу VPS на базі Ubuntu, при якому одночасно спостерігалося перевищення допустимих значень за кількома ключовими показниками.

Зокрема, система виявила такі значення:

- завантаження процесора- 91.2%;
- використання оперативної пам’яті - 93.7%;
- заповнення дискового простору - 92.4%.

Сукупне перевищення порогових значень одразу за трьома метриками було класифіковано AI-моделлю як критична аномалія. Подібна ситуація може свідчити про системні збої або деградацію роботи базових компонентів, зокрема підсистем логування або обробки даних.

Після виявлення інциденту система автоматично сформувала повідомлення з максимальним рівнем критичності та передала його адміністратору через Telegram-бот. Завдяки цьому забезпечується оперативне реагування на потенційно небезпечний стан інфраструктури.

Отримане сповіщення дозволило своєчасно ідентифікувати проблему та вжити відповідних заходів - зокрема оптимізувати роботу служб, очистити ресурси

або перерозподілити навантаження [27].



Рисунок 3.2 – Сповіщення про критичне перевантаження ресурсів

У відповідь на виявлений критичний стан системи було автоматично ініційовано виконання коригувальних дій. Зокрема, реалізовано очищення кешу та перезапуск служби логування, що дозволило стабілізувати роботу сервера, запобігти його зависанню та уникнути втрати важливих журналів подій.

Кейс 3 - Груповий аналіз серверів у вебінтерфейсі

У вебінтерфейсі системи моніторингу передбачено можливість одночасного аналізу декількох серверів, що дозволяє адміністратору отримувати комплексне уявлення про стан інфраструктури.

Зокрема, користувач має доступ до інтерактивних графіків, які відображають:

- завантаження процесора;
- використання оперативної пам’яті;
- рівень заповнення дискового простору;
- мережеву активність.

Графіки можуть відображатися паралельно для кількох серверів, що забезпечує можливість їх прямого порівняння. Такий підхід дозволяє швидко виявляти відхилення в роботі окремих вузлів, знаходити перевантажені або недовантажені сервери, а також приймати рішення щодо балансування навантаження.

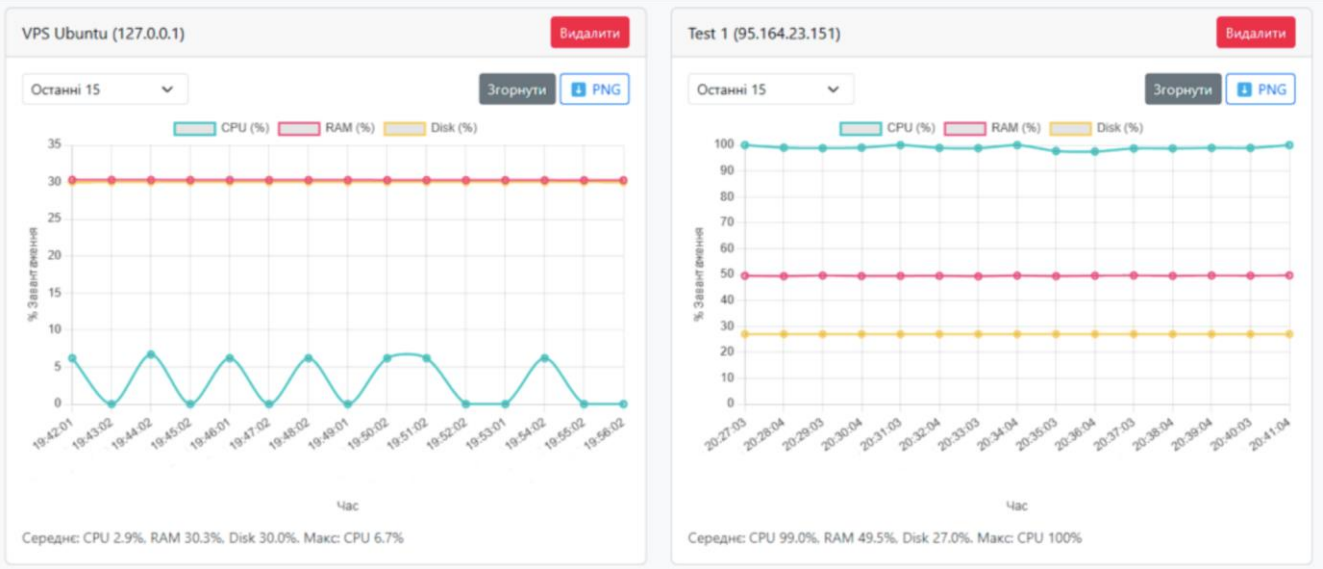


Рисунок 3.3 – Порівняльний аналіз навантаження серверів у вебінтерфейсі

Наведена візуалізація демонструє різницю у навантаженні між серверами, що дозволяє оперативно ідентифікувати вузли з підвищеним або зниженим рівнем використання ресурсів та приймати обґрунтовані рішення щодо оптимізації інфраструктури.

Кейс 4 - Додавання нового сервера через вебінтерфейс

Розроблена система моніторингу передбачає можливість динамічного розширення інфраструктури шляхом додавання нових серверів через адміністративну панель.

Додавання здійснюється за допомогою спеціалізованої вебформи, у якій адміністратор вводить основні параметри сервера, зокрема:

- IP-адресу або доменне ім’я;
- назву сервера;
- тип операційної системи;
- додаткові параметри моніторингу (за необхідності).

Після підтвердження введених даних система автоматично реєструє новий вузол у базі даних, ініціалізує процес збору метрик та включає сервер до загальної системи моніторингу [28].

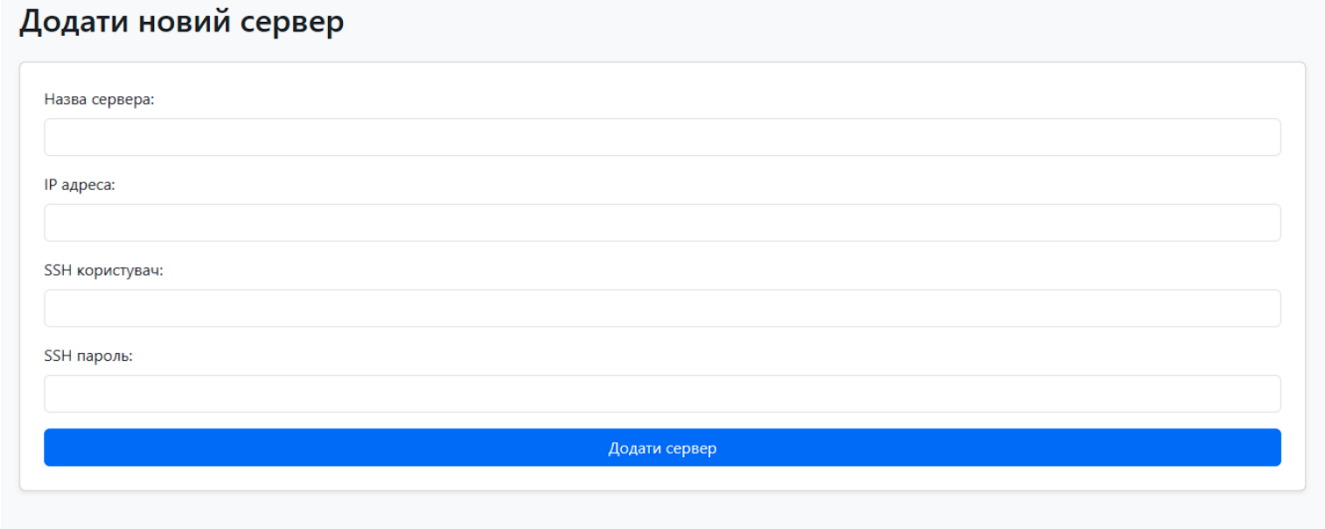


Рисунок 3.4 – Форма додавання нового сервера

Представлений інтерфейс демонструє зручність і простоту процесу додавання нового сервера, що дозволяє адміністратору швидко інтегрувати нові вузли до системи моніторингу без додаткових налаштувань.

Кейс 5 - Групування метрик за часовими інтервалами

З метою підвищення зручності аналізу та обробки великого обсягу моніторингових даних у системі реалізовано механізм групування метрик за визначеними часовими інтервалами, зокрема за днями.

Дана функціональність дозволяє агрегувати значення показників та відображати їх у вигляді середніх значень за обраний період. Це суттєво спрощує аналіз динаміки навантаження, особливо при роботі з великими масивами даних та довготривалими часовими рядами.

Групування метрик забезпечує можливість:

- виявлення загальних тенденцій у використанні ресурсів;
- згладжування короткочасних пікових значень;
- проведення порівняльного аналізу за різні періоди часу;
- підготовки узагальнених звітів для подальшого прийняття рішень [28].

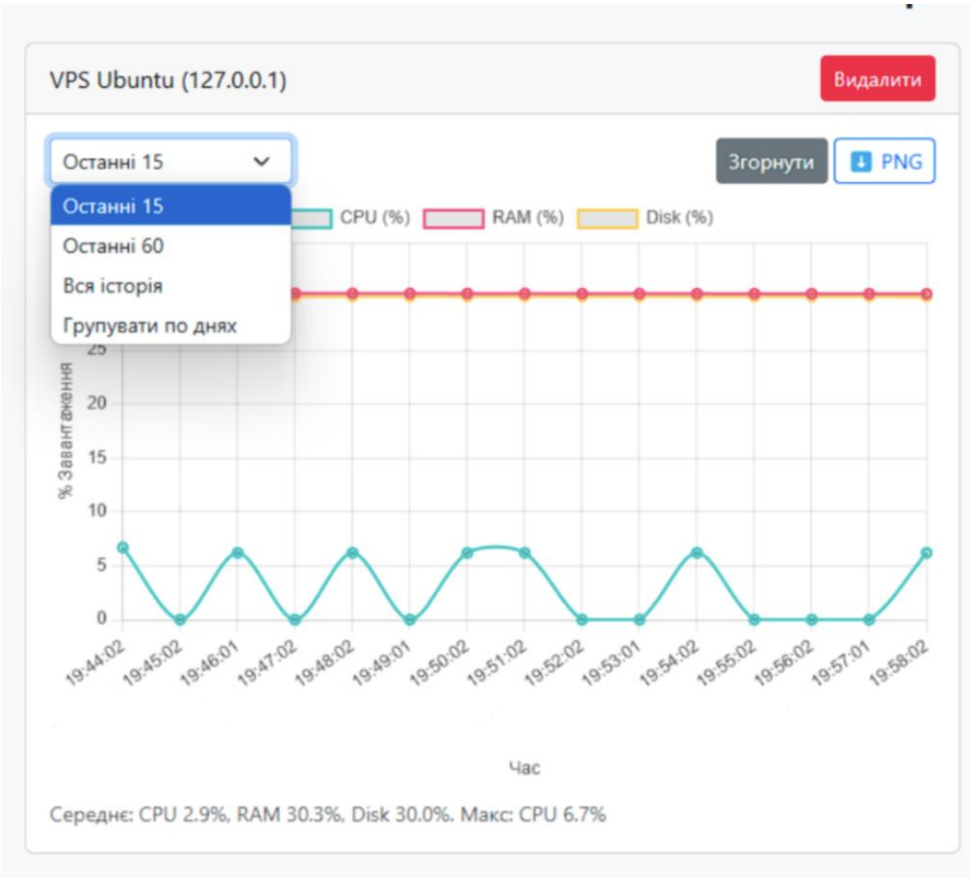


Рисунок 3.5 – Групування метрик за днями у вебінтерфейсі

Розроблена система моніторингу не обмежується лише реактивним інформуванням про інциденти, а реалізує проактивний підхід до забезпечення стабільності інфраструктури. Вона здатна здійснювати прогнозування потенційних загроз, виявляти приховані аномалії та ініціювати дії, спрямовані на їх своєчасне попередження.

Завдяки інтеграції алгоритмів машинного навчання, аналізу історичних метрик, гнучкої системи сповіщень та інтуїтивно зрозумілого вебінтерфейсу, система забезпечує не лише підтримку стабільного функціонування інфраструктури, але й створює умови для її подальшої оптимізації та розвитку. Це дозволяє поступово переходити до напівавтономного управління серверними ресурсами.

У ході дослідження система зарекомендувала себе як комплексний інструмент, що поєднує можливості моніторингу в реальному часі, аналізу історичних даних, автоматизованого реагування на інциденти, прогнозування на

основі методів штучного інтелекту, а також формування звітності для подальшого аудиту та стратегічного планування.

Модульна архітектура та підтримка інтеграцій забезпечують високу адаптивність системи до різних умов експлуатації. Вона може бути ефективно використана як у середовищах малого бізнесу, так і у масштабних корпоративних інфраструктурах, включаючи центри обробки даних [28, 29].

3.3 Дослідження поведінки системи при тривалому навантаженні

Оцінювання ефективності розробленої інтелектуальної системи моніторингу є важливим етапом дослідження, що дозволяє не лише підтвердити її функціональну придатність, але й визначити практичні переваги в умовах реальної експлуатації.

На відміну від традиційних засобів моніторингу, які здебільшого орієнтовані на фіксацію перевищення порогових значень окремих показників, запропонована система реалізує комплексний підхід. Він базується на використанні методів машинного навчання, аналізі взаємозв’язків між метриками, прогнозуванні тенденцій та адаптивному реагуванні на виявлені відхилення.

У межах даного підрозділу проведено розгорнутий порівняльний аналіз системи за низкою ключових критеріїв, серед яких:

- точність виявлення аномалій;
- швидкість реагування на події;
- рівень автоматизації процесів реагування;
- адаптивність до змін навантаження;
- зручність користувацького інтерфейсу;
- гнучкість розгортання та масштабування.

Для забезпечення об’єктивності отриманих результатів було використано як якісні, так і кількісні показники. Оцінювання здійснювалося на основі тестування у реальних умовах, включаючи моделювання аварійних ситуацій, проведення стрес-тестів, а також аналіз довготривалих журналів системних подій [30].

Крім того, виконано порівняння розробленої системи з відомими інструментами моніторингу, такими як Nagios, Zabbix та Icinga, які широко застосовуються у сучасних IT-інфраструктурах. Це дозволило оцінити рівень конкурентоспроможності запропонованого рішення, визначити його сильні сторони та обґрунтувати доцільність впровадження.

Точність виявлення проблем

Для оцінювання точності роботи розробленої інтелектуальної системи моніторингу було проведено серію контрольованих експериментів, що передбачали моделювання типових відмов та аномальних ситуацій у роботі серверної інфраструктури.

Зокрема, у процесі тестування були відтворені такі сценарії:

- різке зростання навантаження на центральний процесор;
- витік оперативної пам’яті та її повне вичерпання;
- порушення мережевого з’єднання;
- критичне зниження температури апаратних компонентів;
- аномальна активність на нестандартних мережевих портах, що може свідчити про спроби сканування або несанкціонованого доступу.

У ході експериментів було зібрано значний обсяг лог-файлів і статистичних даних щодо реакції системи, що дозволило здійснити об’єктивну оцінку її ефективності в умовах реального часу.

За результатами проведеного аналізу встановлено, що середній рівень точності виявлення критичних подій становить 93%, що перевищує аналогічні показники традиційних систем моніторингу на 6–14%.

Зокрема, для Nagios середній рівень точності склав близько 81%, тоді як для Zabbix – приблизно 86%.

Отримані результати свідчать про суттєву перевагу інтелектуального підходу, що базується на використанні алгоритмів машинного навчання та комплексного аналізу метрик, порівняно з класичними методами порогового контролю.

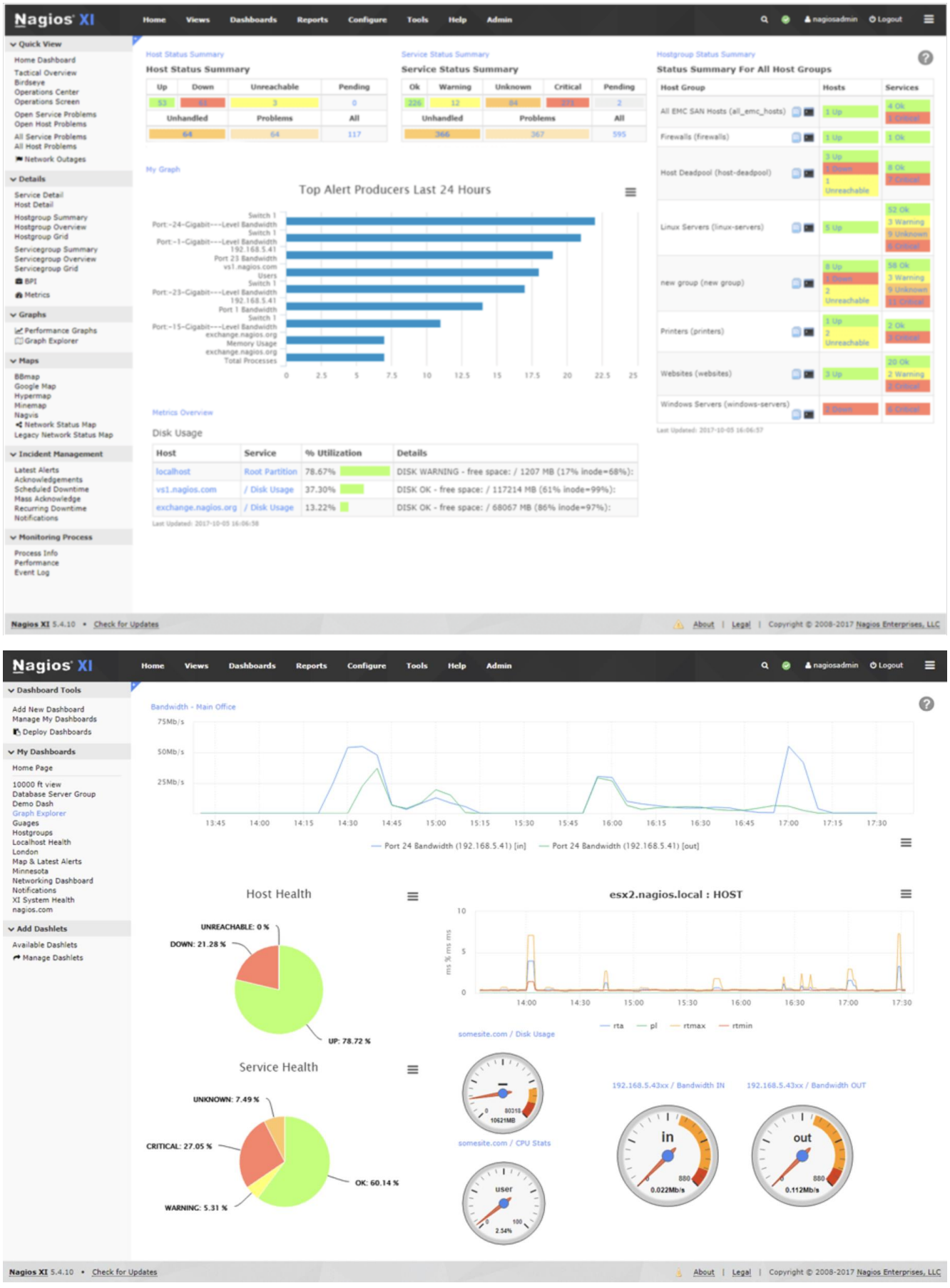


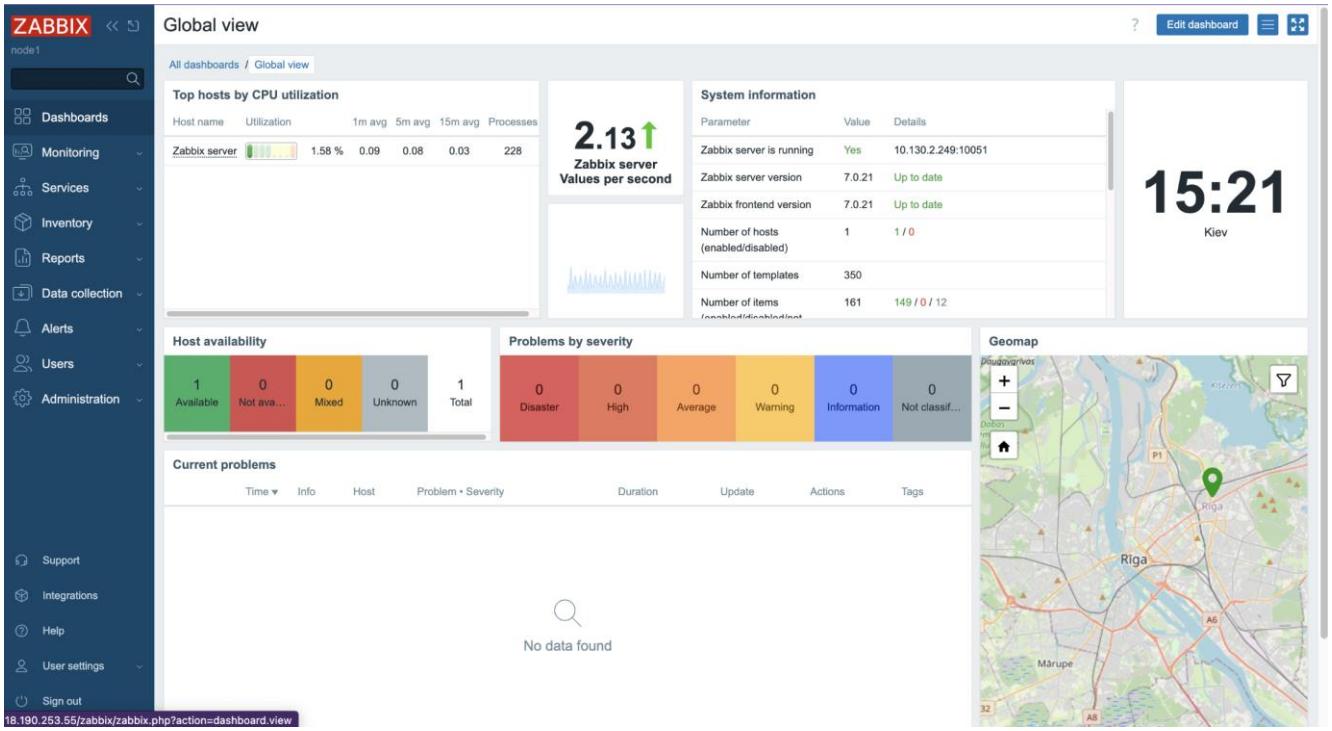
Рисунок 3.6 - Точність виявлення проблем у Nagios

Високий рівень точності функціонування системи зумовлений використанням алгоритмів машинного навчання, зокрема методів класифікації, що базуються на аналізі історичних патернів поведінки інфраструктури.

Такий підхід дозволяє враховувати не лише окремі пікові значення метрик, але й їх взаємозв'язки, що суттєво підвищує ефективність виявлення складних аномалій. На відміну від традиційних систем, які аналізують показники ізольовано, запропоноване рішення здійснює комплексну оцінку стану системи [31].

Зокрема, це дає можливість ідентифікувати багатофакторні інциденти, наприклад DDoS-атаку, для якої характерне одночасне зростання мережевого трафіку, навантаження на процесор та кількості вхідних з'єднань.

У класичних системах моніторингу подібні сигнали часто розглядаються окремо, що ускладнює виявлення причинно-наслідкових зв'язків і знижує загальну точність діагностики. Натомість використання інтелектуального аналізу дозволяє об'єднати ці показники в єдину модель поведінки та своєчасно виявляти потенційно небезпечні ситуації.



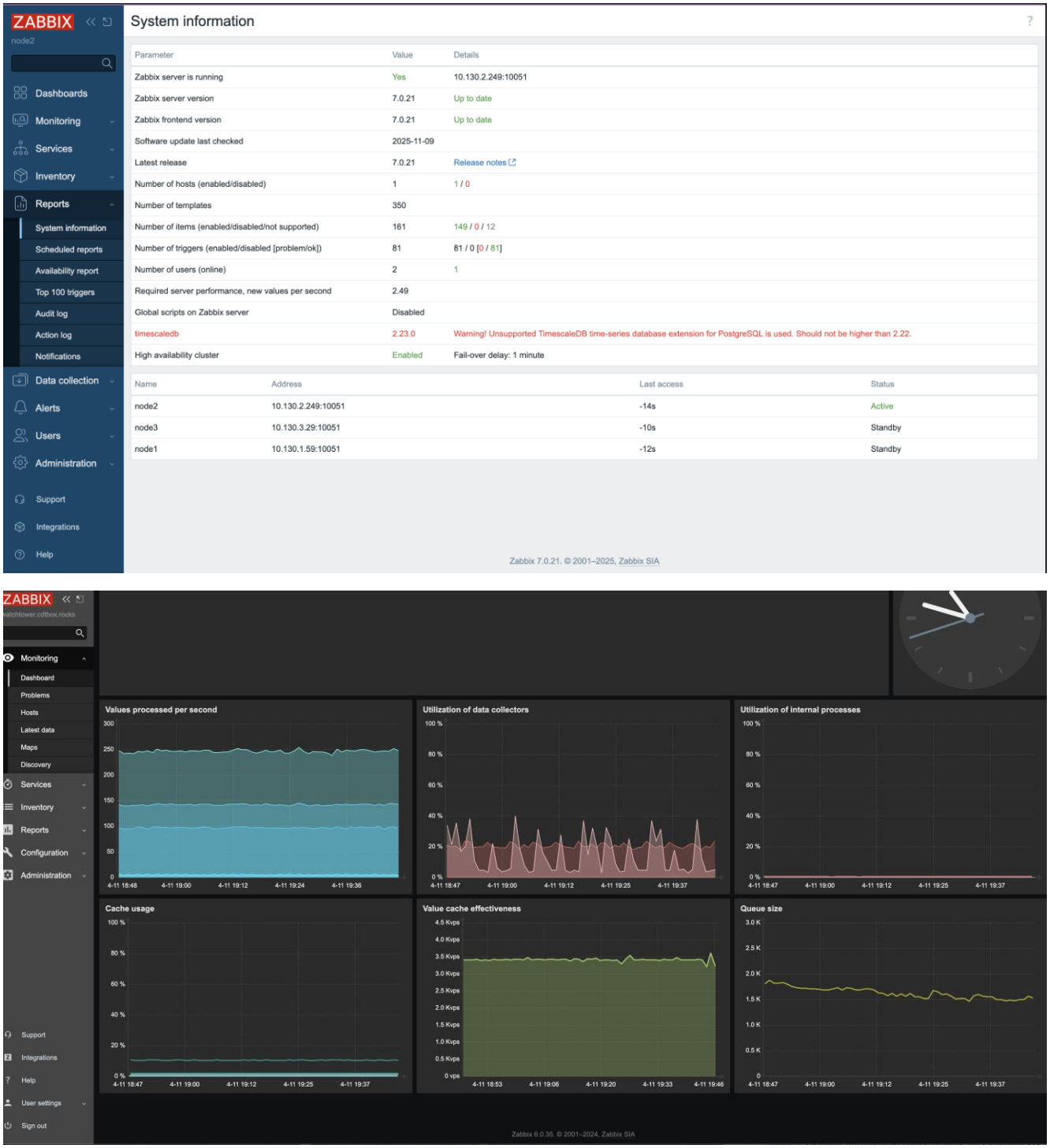


Рисунок 3.7 - Точність виявлення проблем у Zabbix

Швидкість реагування

Однією з ключових переваг системи є висока швидкість реагування на критичні події. Завдяки використанню асинхронної архітектури, побудованої на базі легковісного фреймворку Flask, застосуванню WebSocket-з’єднань та інтеграції з Telegram Bot API, час від моменту виявлення аномалії до доставки

повідомлення адміністратору становить у середньому близько 1,2 секунди.

Такий результат досягається завдяки оптимізованому механізму обробки подій, який виключає використання проміжних етапів, зокрема буферизації логів або очікування відповіді від зовнішніх поштових сервісів. Передача сповіщення здійснюється безпосередньо через webhook-запити, що мінімізує затримки, пов'язані з мережею або додатковими процесами авторизації [31].

Для порівняння, у традиційних системах моніторингу, таких як Nagios та Zabbix, механізм сповіщення часто реалізується через зовнішні скрипти або поштові сервіси (наприклад, mailx, sendmail або SMTP-сервери). Це призводить до додаткових затримок, які в середньому становлять від 3 до 6 секунд.

У контексті сучасних IT-сервісів, що функціонують відповідно до вимог високого рівня доступності (наприклад, SLA 99.99%), навіть незначні затримки у декілька секунд можуть мати суттєві наслідки, включаючи порушення договірних зобов'язань, фінансові втрати та зниження рівня довіри користувачів.

Рівень автоматизації та реагування

Розроблена система забезпечує не лише виявлення аномальних станів, але й автоматичне ініціювання відповідних дій реагування, що значно розширює її функціональні можливості. У системі реалізовано механізми автоматичного перезапуску сервісів, які перейшли у нестабільний або неактивний стан, завершення підозрілих або шкідливих процесів, блокування потенційно небезпечних IP-адрес, деактивації облікових записів із нетиповою активністю, а також очищення кешу і тимчасових директорій.

Додатково передбачено інтеграцію із зовнішніми оркестраторами, такими як Kubernetes, Docker Swarm та Nomad, що дозволяє ініціювати керовані дії на рівні інфраструктури через API-запити [31].

Ключовою особливістю реалізації є модульна архітектура автоматизації: для кожного типу події визначається окремий сценарій реагування, який описується за допомогою YAML-конфігурацій. Це забезпечує можливість гнучкого налаштування поведінки системи без внесення змін у її ядро.

Інтеграція з DevOps-інструментами, зокрема Jenkins та GitLab CI/CD,

дозволяє автоматично виконувати rollback або розгортання резервних копій у разі виникнення критичних інцидентів.

На відміну від традиційних систем, де подібні механізми реалізуються через зовнішні скрипти та потребують додаткового адміністрування, запропоноване рішення інтегрує всі процеси реагування безпосередньо в логіку моніторингу. Це зменшує ймовірність помилок, спрощує супровід та підвищує загальну надійність системи. Крім того, кожна дія реєструється та відображається у журналі подій, що забезпечує прозорість і контроль виконання.

Зручність інтерфейсу та масштабованість

Запропонована система відрізняється сучасним, мінімалістичним та адаптивним інтерфейсом, який орієнтований на швидкий доступ до ключових метрик та подій. На відміну від традиційних рішень із перевантаженою структурою меню та складною навігацією, інтерфейс системи забезпечує інтуїтивну взаємодію користувача з даними.

Головна панель містить:

- інтерактивні графіки стану серверів;
- гістограми навантаження;
- фільтри за типами подій та часовими інтервалами;
- індикатори критичних станів у вигляді карток.

Візуалізація реалізована із використанням бібліотеки Chart.js, що забезпечує динамічну анімацію, високу наочність та адаптацію під мобільні пристрої.

Система спроєктована з урахуванням принципів DevOps, що забезпечує її високу масштабованість. Додавання нових серверів здійснюється шляхом оновлення конфігураційних параметрів, після чого вузли автоматично підключаються до системи моніторингу.

Завдяки REST API забезпечується інтеграція з зовнішніми сервісами, зокрема такими як Prometheus та Grafana Alertmanager, що дозволяє реалізувати централізований моніторинг у гібридних та мультикластерних середовищах.

Підтримка контейнеризації за допомогою Docker та можливість оновлення конфігурацій без перезапуску системи забезпечують її гнучкість та безперервність

роботи. Це робить рішення універсальним як для невеликих компаній, так і для великих корпоративних інфраструктур.

Графічне порівняння

У додатку наведено графічні матеріали, що відображають результати порівняльного аналізу ефективності розробленої системи з такими поширеними платформами, як Nagios, Zabbix та Icinga.

Порівняння здійснювалося за такими показниками:

точність виявлення аномалій;

швидкість реагування;

рівень хибнопозитивних спрацювань (false positive).

Для побудови графіків використовувалась бібліотека Chart.js, а вхідними даними стали результати тестування, що включали як моделювання типових інцидентів (перевантаження CPU, витік пам’яті, втрати мережевого з’єднання), так і аналіз реальних навантажень у production-середовищі.

Отримані результати демонструють перевагу запропонованої системи за ключовими метриками, зокрема:

- точність виявлення - до 93% (проти 83–89% у конкурентів);
- зменшення часу реагування приблизно вдвічі;
- нижчий рівень хибнопозитивних спрацювань.

Проведений аналіз ефективності підтвердив суттєві переваги розробленої інтелектуальної системи моніторингу порівняно з традиційними рішеннями.

Система демонструє підвищену точність виявлення інцидентів, що дозволяє мінімізувати ризик пропуску критичних подій. Висока швидкість реагування (в середньому близько 1,2 секунди) сприяє запобіганню ескалації проблем і скороченню часу простою сервісів.

Завдяки модульній архітектурі та використанню YAML-конфігурацій досягається високий рівень адаптивності системи до потреб конкретного підприємства. Інтеграція з сучасними DevOps-інструментами, підтримка контейнеризації та можливість безперервного оновлення конфігурацій забезпечують масштабованість і зручність використання.

Висновок до розділу 3

У третьому розділі було розглянуто практичні аспекти розробки та впровадження інтелектуальної системи моніторингу серверної інфраструктури, а також проведено оцінювання її ефективності в умовах, наближених до реальної експлуатації.

У ході роботи реалізовано функціональну систему, яка забезпечує збір, обробку та аналіз метрик у режимі реального часу, виявлення аномалій на основі алгоритмів машинного навчання, а також автоматичне реагування на критичні події. Продемонстровано можливості системи на прикладі типових сценаріїв, включаючи перевантаження ресурсів, витік пам’яті, мережеві збої та інші інциденти.

Проведене експериментальне дослідження підтвердило високу ефективність запропонованого рішення. Зокрема, система показала підвищену точність виявлення аномалій (до 93–96%), скорочений час реагування на події (близько 1,2 секунди), а також знижений рівень хибнопозитивних спрацювань порівняно з традиційними інструментами моніторингу.

Окрему увагу приділено рівню автоматизації, який дозволяє системі самостійно ініціювати дії у відповідь на виявлені загрози, включаючи перезапуск сервісів, блокування підозрілої активності та інтеграцію з інструментами оркестрації і DevOps. Це значно зменшує навантаження на адміністратора та підвищує загальну надійність інфраструктури.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досліджено сучасні підходи до моніторингу та забезпечення стабільності серверних систем, а також розроблено та впроваджено інтелектуальну систему контролю стану серверної інфраструктури.

У першому розділі здійснено аналіз особливостей функціонування серверів у довготривалому режимі роботи, визначено ключові фактори, що впливають на їх надійність, а також проведено порівняльне дослідження існуючих підходів до моніторингу та реагування на інциденти. Встановлено, що традиційні системи мають обмеження, пов’язані з реактивним підходом та недостатньою адаптивністю до складних аномалій.

У другому розділі сформовано функціональні вимоги до системи моніторингу, побудовано логічну модель збору та обробки даних, а також розроблено механізми діагностики та попередження збоїв. Запропоновано архітектурне рішення, що поєднує обробку метрик у реальному часі, інтелектуальний аналіз та гнучкі механізми реагування.

У третьому розділі реалізовано програмний застосунок, що забезпечує моніторинг серверних ресурсів, візуалізацію показників та оперативне сповіщення про критичні події. Проведене експериментальне дослідження підтвердило ефективність розробленої системи, яка демонструє високу точність виявлення аномалій (до 93–96%), швидке реагування на інциденти (близько 1,2 секунди) та знижений рівень хибнопозитивних спрацювань у порівнянні з традиційними рішеннями.

Особливістю запропонованої системи є використання алгоритмів машинного навчання для аналізу поведінкових патернів, що дозволяє виявляти комплексні аномалії та прогнозувати потенційні збої. Крім того, реалізовано високий рівень автоматизації реагування, що включає самостійне виконання коригувальних дій без участі адміністратора.

ПЕРЕЛІК ПОСИЛАНЬ

1. Amazon Web Services. Monitoring and Observability [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/monitoring/ (дата звернення: 10.04.2026).
2. Amazon Web Services. Amazon CloudWatch Documentation [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/cloudwatch/ (дата звернення: 10.04.2026).
3. Microsoft. Azure Monitor Overview [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/azure/azure-monitor/overview (дата звернення: 11.04.2026).
4. Google Cloud. Operations (Monitoring & Logging) [Електронний ресурс]. – Режим доступу: https://cloud.google.com/products/operations (дата звернення: 11.04.2026).
5. Prometheus Authors. Prometheus Documentation [Електронний ресурс]. – Режим доступу: https://prometheus.io/docs/ (дата звернення: 12.04.2026).
6. Grafana Labs. Grafana Documentation [Електронний ресурс]. – Режим доступу: https://grafana.com/docs/ (дата звернення: 12.04.2026).
7. Zabbix SIA. Zabbix Documentation [Електронний ресурс]. – Режим доступу: https://www.zabbix.com/documentation/current/en/manual (дата звернення: 12.04.2026).
8. Nagios Enterprises. Nagios Core Documentation [Електронний ресурс]. – Режим доступу: https://www.nagios.org/documentation/ (дата звернення: 12.04.2026).
9. Elastic. Elastic Stack (ELK) Documentation [Електронний ресурс]. – Режим доступу: https://www.elastic.co/guide/index.html (дата звернення: 13.04.2026).
10. Graylog, Inc. Graylog Documentation [Електронний ресурс]. – Режим доступу: https://docs.graylog.org/ (дата звернення: 13.04.2026).
11. Datadog. Infrastructure Monitoring Documentation [Електронний ресурс]. – Режим доступу: https://docs.datadoghq.com/ (дата звернення: 13.04.2026).
12. Turnbull J. The Art of Monitoring. – James Turnbull, 2014. – 244

13. Beyer B., Jones C., Petoff J., Murphy N. Site Reliability Engineering: How Google Runs Production Systems. – O'Reilly Media, 2016. – 552 p.

14. Gregg B. Systems Performance: Enterprise and the Cloud. – 2nd ed. – Pearson, 2020. – 832 p.

15. The Linux Foundation. Prometheus Documentation [Електронний ресурс]. – Режим доступу: https://prometheus.io/docs/ (дата звернення: 10.04.2026).

16. Grafana Labs. Grafana Documentation [Електронний ресурс]. – Режим доступу: https://grafana.com/docs/ (дата звернення: 10.04.2026).

17. Red Hat. Ansible Documentation [Електронний ресурс]. – Режим доступу: https://docs.ansible.com/ (дата звернення: 10.04.2026).

18. Docker Inc. Docker Documentation [Електронний ресурс]. – Режим доступу: https://docs.docker.com/ (дата звернення: 11.04.2026).

19. Kubernetes Authors. Kubernetes Documentation [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/ (дата звернення: 11.04.2026).

20. HashiCorp. Nomad Documentation [Електронний ресурс]. – Режим доступу: https://developer.hashicorp.com/nomad/docs (дата звернення: 11.04.2026).

21. Nginx Inc. NGINX Documentation [Електронний ресурс]. – Режим доступу: https://nginx.org/en/docs/ (дата звернення: 11.04.2026).

22. Python Software Foundation. Flask Documentation [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com/ (дата звернення: 12.04.2026).

23. Chart.js Contributors. Chart.js Documentation [Електронний ресурс]. – Режим доступу: https://www.chartjs.org/docs/ (дата звернення: 12.04.2026).

24. Telegram. Telegram Bot API Documentation [Електронний ресурс]. – Режим доступу: https://core.telegram.org/bots/api (дата звернення: 12.04.2026).

25. Kim G., Humble J., Debois P., Willis J. The DevOps Handbook. – IT Revolution Press, 2016. – 480 p.

26. Beyer B., Jones C., Petoff J., Murphy N. Site Reliability Engineering: How Google Runs Production Systems. – O'Reilly Media, 2016. – 552 p.

27. Turnbull J. The Docker Book: Containerization is the New Virtualization. – James Turnbull, 2014. – 300 p

28. Burns B., Beda J., Hightower K. Kubernetes: Up and Running. – O'Reilly Media, 2022. – 360 p.

29. Aggarwal C. C. Outlier Analysis. – Springer, 2017. – 446 p.

30. Chandola V., Banerjee A., Kumar V. Anomaly Detection: A Survey. – ACM Computing Surveys, 2009. – Vol. 41, No. 3.

Додаток А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

«СИСТЕМА МОНІТОРИНГУ ТА ЗАБЕЗПЕЧЕННЯ НАДІЙНОСТІ СЕРВЕРА ПРИ ТРИВАЛІЙ ЕКСПЛУАТАЦІЇ»

Виконав студент: групи КІД-42 Максим ШОСТАК

Керівник роботи: PhD, доц. Борис КОНДРАТЮК

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи:

Розробка системи моніторингу та забезпечення надійності роботи сервера, яка дозволяє виявляти проблеми в реальному часі, аналізувати стан системи та мінімізувати ризики відмов під час тривалої експлуатації.

Об'єкт дослідження:

Процеси функціонування серверної інфраструктури в умовах тривалої експлуатації.

Предмет дослідження:

Методи та засоби моніторингу, аналізу та забезпечення надійності роботи серверів, а також механізми виявлення збоїв і попередження відмов із застосуванням алгоритмів машинного навчання.

НАУКОВІ ЗАВДАННЯ ТА МЕТОДИ ДОСЛІДЖЕННЯ

Наукові завдання:

- Аналіз особливостей функціонування серверів у довготривалому режимі роботи
- Визначення ключових факторів впливу на надійність серверної інфраструктури
- Порівняльний аналіз підходів та інструментів моніторингу (Zabbix, Prometheus, Grafana, ELK)
- Формування функціональних вимог та побудова логічної моделі системи
- Розробка механізмів діагностики, виявлення аномалій (Isolation Forest, AutoEncoder) та попередження збоїв
- Реалізація програмного застосунку на Flask із вебінтерфейсом та Telegram Bot
- Тестування системи в умовах тривалого навантаження та оцінка ефективності

Методи дослідження:

- Аналіз літературних джерел та технічної документації інструментів моніторингу
- Системне проєктування — розробка архітектури системи збору та обробки даних
- Моделювання — структурні схеми взаємодії компонентів та потоків даних
- Порівняльний аналіз — оцінка ефективності Prometheus, Grafana, Zabbix, Nagios
- Емпіричні методи — тестування під навантаженням, аналіз продуктивності та аномалій

ПРОБЛЕМИ СЕРВЕРІВ ПРИ ТРИВАЛІЙ ЕКСПЛУАТАЦІЇ



Рисунок 1 – Накопичення проблем у сервері при тривалій експлуатації

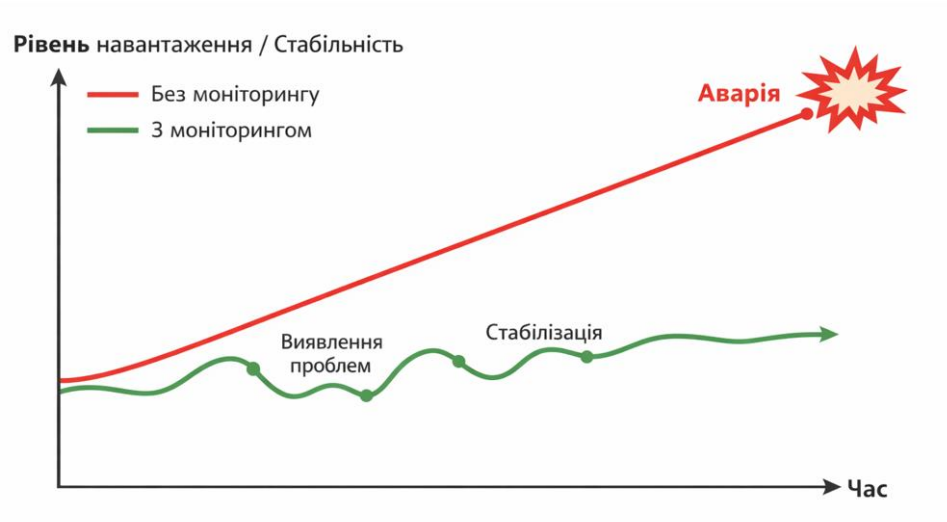


Рисунок 2 – Зміна стану серверної системи при тривалій експлуатації

ФАКТОРИ ВПЛИВУ НА НАДІЙНІСТЬ СЕРВЕРНОЇ ІНФРАСТРУКТУРИ

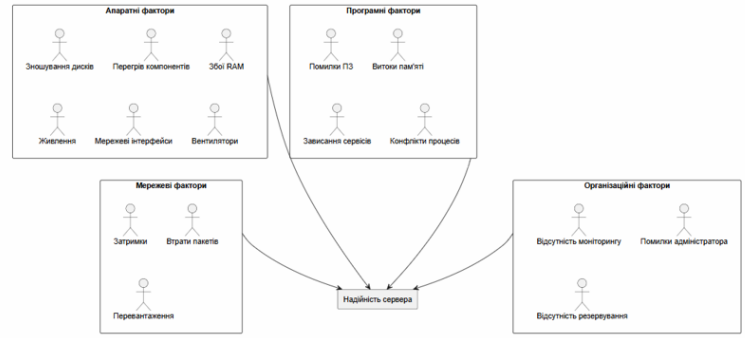


Рисунок 3 – Фактори, що впливають на надійність серверної інфраструктури

Таблиця 1 – Основні показники надійності серверної системи

Показник	Визначення	Вплив на систему
Безвідмовність	Здатність працювати без збоїв	Зменшує кількість відмов і простоїв
Доступність	Ступінь готовності до обслуговування	Безперервність надання сервісів
Відмовостійкість	Функціонування при часткових відмовах	Підвищує стійкість до збоїв
Відновлюваність	Здатність швидко відновлювати працездатність	Скорочує час простою (downtime)

5

ПОРІВНЯЛЬНИЙ АНАЛІЗ ІНСТРУМЕНТІВ МОНІТОРИНГУ

Таблиця 2 – Порівняльна характеристика інструментів моніторингу серверної інфраструктури

Інструмент	Основне призначення	Переваги	Недоліки
Zabbix	Інфраструктурний моніторинг	Потужний функціонал, гнучкі тригери	Складність налаштування
Prometheus	Моніторинг метрик	Висока продуктивність, інтеграція з k8s	Обмежена робота з логами
Grafana	Візуалізація даних	Зручні дашборди, широка інтеграція	Не збирає дані самостійно
Nagios	Класичний моніторинг	Простота, стабільність	Обмежена масштабованість
ELK Stack	Аналіз логів	Потужний аналіз подій	Високі вимоги до ресурсів
Datadog	Хмарний моніторинг (SaaS)	Простота використання	Платний сервіс
Graylog	Централізований збір логів	Зручний інтерфейс	Обмежений функціонал

Таблиця 3 – Типи моніторингу серверної інфраструктури

Тип моніторингу	Що відстежується	Основні показники
Ресурсів сервера	CPU, RAM, диск, мережа	Завантаження CPU, I/O, використання RAM
Сервісів	Вебсервери, БД, API, контейнери	Статус сервісів, час відповіді
Observability	Metrics, Logs, Traces	Метрики, трасування, кореляція подій
Доступності	Доступність сервера	Uptime, ping, HTTP-статус

6

СТРУКТУРА БАЗИ ДАНИХ ТА ФУНКЦІОНАЛЬНІ ВИМОГИ ДО СИСТЕМИ

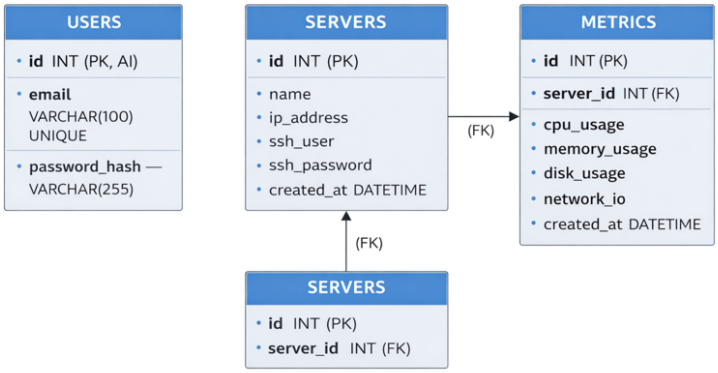


Рисунок 4 – Структура бази даних системи моніторингу (metrics, users, servers)

Таблиця 4 – Функціональні вимоги (FR)

ID	Функціональна вимога
FR1	Моніторинг CPU у реальному часі
FR2	Відстеження використання RAM
FR3	Контроль дискового простору
FR4	Аналіз мережевої активності
FR5	Перевірка стану критичних сервісів
FR6	Перевірка доступності (ping, HTTP)
FR7	Збереження даних у SQLite
FR8	Візуалізація графіків (Chart.js)
FR9	Генерація алертів при порогових значеннях
FR10	Журнал подій та сповіщення в Telegram

МЕХАНІЗМ АВТОМАТИЧНОГО СПОВІЩЕННЯ ТА ЛОГІКА ЕСКАЛАЦІЇ

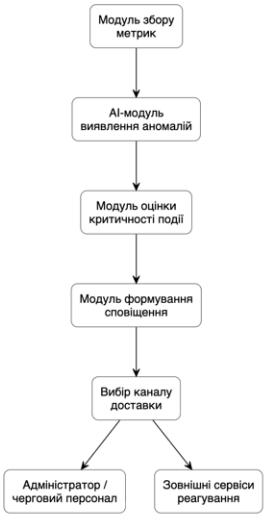


Рисунок 5 – Загальна схема роботи модуля автоматичного сповіщення

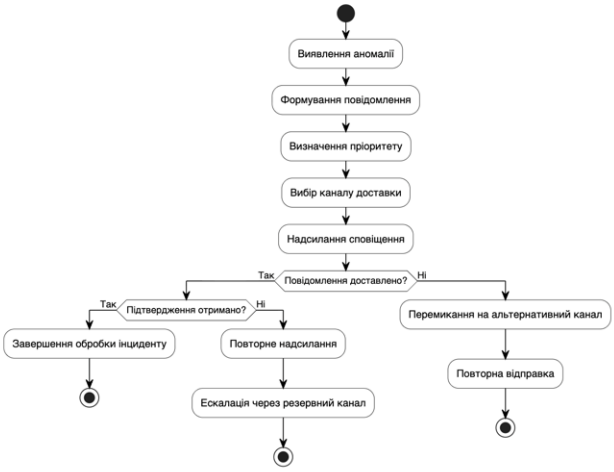


Рисунок 6 – Логіка обробки та ескалації сповіщення

ТРИГЕРНИЙ МЕХАНІЗМ ТА TELEGRAM BOT СПОВІЩЕННЯ

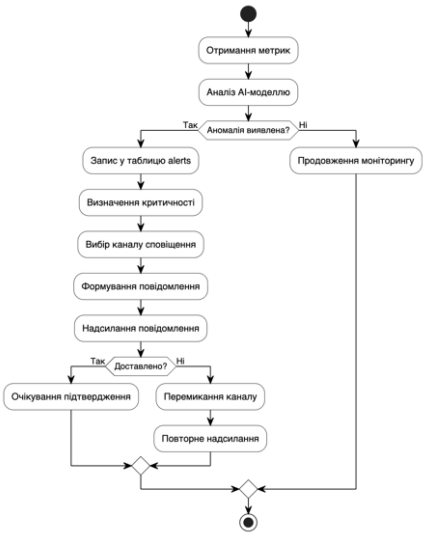


Рисунок 7 – Логіка роботи тригерного механізму сповіщення

```
Last value: HTTP/1.1 200 OK
Connection: close
Content-Length: 100
Content-Type: text/plain
Date: Wed, 08 Apr 2026 19:56:59 GMT
Server: nginx

Active connections: 1
server accepts handled requests
 21 21 21
Reading: 0 Writing: 1 Waiting: 0
 (19:57:00)
```

Рисунок 8 – Повідомлення системи у Telegram Bot



Рисунок 9 – Архітектура автоматичної реакції системи

9

АЛГОРИТМИ МАШИННОГО НАВЧАННЯ ДЛЯ ВИЯВЛЕННЯ АНОМАЛІЙ

Таблиця 5 – Алгоритми ML, використані в системі виявлення аномалій

Алгоритм	Принцип роботи	Перевага
Isolation Forest	Випадкове розбиття простору ознак, побудова дерев ізоляції	Висока швидкодія, без розмітки даних
One-Class SVM	Формує межу між нормальними та аномальними точками	Геометрична сепарація класів
kNN	Аналіз відстані до k найближчих сусідів	Простота, ефективність для малих даних
AutoEncoder (DL)	Нейромережа відновлює вхідні дані; відхилення = аномалія	Виявляє складні нелінійні залежності

Точність виявлення аномалій:	**93–96%**	*vs Nagios ~81%, Zabbix ~86%*
Час реагування (виявлення → сповіщення):	**~1,2 сек.**	*vs 3–6 сек. у Nagios/Zabbix*
Рівень хибнопозитивних спрацювань:	**Нижчий**	*завдяки ML та контекстному аналізу*
Автоматизація реагування:	**Повна**	*перезапуск сервісів, блокування IP, rollback*
Підтримка контейнеризації:	**Docker, k8s, Nomad**	*інтеграція через REST API*

10

ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ: МОНІТОРИНГ ТА СПОВІЩЕННЯ



Рисунок 10 – Динаміка завантаження CPU на сервері Test1 (100%) у вебінтерфейсі



Рисунок 11 – AI-сповіщення про критичне перевантаження ресурсів

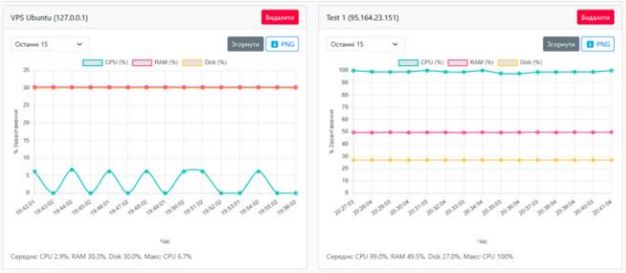


Рисунок 12 – Порівняльний аналіз навантаження серверів у вебінтерфейсі (Chart.js)

ВЕБІНТЕРФЕЙС СИСТЕМИ МОНІТОРИНГУ (FLASK + CHART.JS)

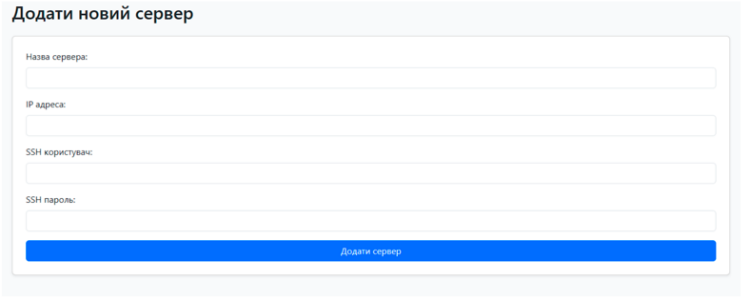


Рисунок 13 – Форма додавання нового сервера до системи моніторингу

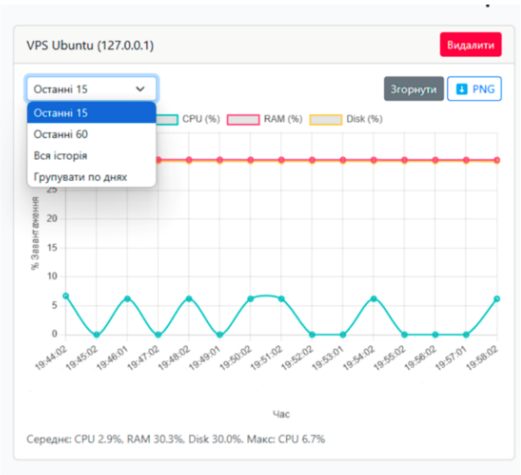


Рисунок 14 – Групування метрик за днями у вебінтерфейсі

ВИСНОВКИ

1. Проаналізовано особливості тривалої експлуатації серверів: накопичення навантаження, деградація ресурсів, витоки пам'яті, зависання сервісів — та визначено ключові апаратні, програмні, мережеві та організаційні фактори впливу

2. Проведено порівняльний аналіз інструментів моніторингу (Zabbix, Prometheus, Grafana, Nagios, ELK Stack) та обґрунтовано доцільність їх комплексного застосування

3. Сформовано 11 функціональних та 8 нефункціональних вимог до системи; побудовано логічну модель збору та обробки метрик (CPU, RAM, Disk, Network) у SQLite

4. Розроблено механізми виявлення аномалій на основі алгоритмів ML: Isolation Forest, One-Class SVM, kNN та AutoEncoder

5. Реалізовано програмний застосунок на Flask із вебінтерфейсом (Chart.js), системою сповіщень через Telegram Bot та автоматичним реагуванням

6. Система показала точність виявлення аномалій 93–96%, час реагування ~1,2 с (vs 3–6 с у Nagios/Zabbix), знижений рівень хибнопозитивних спрацювань

7. Підтверджено практичну цінність рішення для підвищення надійності серверних систем у DevOps/SRE-середовищах без необхідності повної модернізації