МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проєкту (роботи)
магістра
(ступінь вищої освіти)

на тему **КОМПЛЕКСНА СИСТЕМА МОНІТОРИНГУ КОРПОРАТИВНОЇ ІНФРАСТРУКТУРИ НА БАЗІ ZABBIX**

Виконав: студент 2 курсу, групи КНТ-514м
спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні системи та мережі

ГУРЄЄВ К.О.
(ПРІЗВИЩЕ та ініціали)
Керівник СКРУПСЬКИЙ С.Ю.
(ПРІЗВИЩЕ та ініціали)
Рецензент ЩЕРБАКОВ В.І.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти магістерський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)

Освітня програма (спеціалізація) Комп'ютерні системи та мережі
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Зав. кафедри Кудерметов Р.К.
_____
"15" жовтня 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ГУРЕЄВА Костянтина Олеговича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Комплексна система моніторингу корпоративної інфраструктури на базі zabbix
керівник проєкту (роботи) кан. техн. наук, доцент, Скрупський Степан Юрійович
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом вищого навчального закладу від "30" жовтня 2025 року № 491
2. Строк подання студентом проєкту (роботи) 23 грудня 2025 року
3. Вихідні дані до проєкту (роботи) топологія корпоративної мережі підприємства, перелік серверного та мережевого обладнання (близько 60 одиниць), вимоги до рівня доступності сервісів (SLA 99.95%), час реакції на інцидент не більше 5 хвилин, глибина зберігання архівних даних не менше 1 року.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз предметної області та вибір інструментарію;
2) Проєктування архітектури системи моніторингу;
3) Програмна реалізація та налаштування системи;
4) Дослідження ефективності системи.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	СКРУПСЬКИЙ С. Ю., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01 жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз наявних методів та алгоритмів пошуку	01.10.2025 р.	
	об’єктів, перегляд отриманих результатів, виділення		
	недоліків та переваг		
2	Розробка структури системи	10.11.2025 р.	
3	Розробка алгоритму роботи системи	15.11.2025 р.	
4	Реалізація системи уточненого розпізнавання	25.11.2025 р.	
5	Дослідження системи уточненого розпізнавання	30.11.2025 р.	
6	Оформлення отриманих результатів у ПЗ	05.12.2025 р.	
7	Оформлення графічного матеріалу	10.12.2025 р.	
8	Оформлення допоміжного матеріалу	15.12.2025 р.	

Студент _____ **Костянтин ГУРЕЄВ**
(підпис) (ім’я, ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ **Степан СКРУПСЬКИЙ**
(підпис) (ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 122 с., 9 рис., 4 табл., 24 джерел.

ALERTING, GRAFANA, SNMP, POSTGRESQL, ZABBIX, ЕФЕКТИВНІСТЬ, МОНІТОРИНГ, IT-ІНФРАСТРУКТУРА.

Об’єкт дослідження - процес управління працездатністю та продуктивністю компонентів корпоративної IT-інфраструктури. Предмет дослідження - механізми та інструментальні засоби побудови автоматизованих систем моніторингу на базі платформи Zabbix.

Мета роботи - підвищення ефективності бізнес-процесів шляхом впровадження системи моніторингу для проактивного виявлення збоїв та скорочення часу реакції.

Наукова новизна полягає в удосконаленні підходу до моніторингу розподілених мереж шляхом поєднання активних проксі-серверів з теговою кореляцією подій, що забезпечує цілісність даних та знижує рівень інформаційного шуму під час аварій на 99%.

Практична цінність. Реалізована система дозволяє автоматизувати виявлення та класифікацію інцидентів у гетерогенному середовищі, зменшити час відновлення сервісів (MTTR) та мінімізувати фінансові втрати, а також забезпечити масштабування на нові філії без зміни ядра архітектури.

Матеріали та методи: системний та порівняльний аналіз, натурний експеримент (Zabbix 6.0 LTS, PostgreSQL, Debian Linux, Grafana). Результати: спроєктовано та впроваджено відмовостійку архітектуру моніторингу, розроблено алгоритми фільтрації подій та інтеграцію з Telegram, внаслідок чого середній час виявлення інцидентів (MTTD) скорочено з 45 хвилин до 1 хвилини.

Галузь використання - IT-департаменти підприємств малого та середнього бізнесу.

ABSTRACT

Explanatory note to the master’s work: 122 p., 9 figures, 4 tables, 24 sources.

ALERTING, EFFICIENCY, MONITORING, IT INFRASTRUCTURE, GRAFANA, SNMP, POSTGRESQL, ZABBIX.

The object of research is the process of managing the performance and availability of corporate IT infrastructure components. The subject of research is mechanisms and tools for building automated monitoring systems based on the Zabbix platform.

The purpose of the work is to increase business process efficiency by implementing a monitoring system for proactive failure detection and reduced incident response time.

Scientific novelty lies in improving the distributed network monitoring approach by combining active proxy servers with tag-based event correlation, ensuring data integrity and reducing information noise during accidents by 99%.

Practical value. The implemented system allows to automate incident detection and classification in a heterogeneous environment, reduce Mean Time To Resolve (MTTR) and minimize financial losses, and ensure scalability to new branches without changing the core architecture.

Materials and methods: system and comparative analysis, field experiment (Zabbix 6.0 LTS, PostgreSQL, Debian Linux, Grafana). Results: a fault-tolerant monitoring architecture, event filtering algorithms, and Telegram integration were designed and implemented; the average incident detection time (MTTD) was reduced from 45 minutes to 1 minute.

Field of use - IT departments of small and medium-sized enterprises.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API	–	Application Programming Interface (Інтерфейс прикладного програмування)
CPU	–	Central Processing Unit (Центральний процесор)
HA	–	High Availability (Висока доступність)
HTTP	–	HyperText Transfer Protocol
ICMP	–	Internet Control Message Protocol
JSON	–	JavaScript Object Notation
LLD	–	Low-Level Discovery (Низькорівневе виявлення)
MTTD	–	Mean Time To Detect (Середній час виявлення)
MTTR	–	Mean Time To Resolve (Середній час вирішення)
NVPS	–	New Values Per Second (Нових значень за секунду)
OID	–	Object Identifier (Ідентифікатор об'єкта)
SLA	–	Service Level Agreement (Угода про рівень послуг)
SNMP	–	Simple Network Management Protocol
SPOF	–	Single Point of Failure (Єдина точка відмови)
SQL	–	Structured Query Language
VLAN	–	Virtual Local Area Network
VPN	–	Virtual Private Network

ВСТУП

Актуальність теми. На сучасному етапі розвитку інформаційних технологій стабільність та безперервність роботи IT-інфраструктури виступають фундаментальними факторами, що забезпечують ефективне функціонування будь-якого підприємства. Стрімка цифровізація бізнес-процесів, перехід до хмарних технологій та зростання обсягів даних призвели до того, що інформаційні системи стали критично важливою складовою виробничого циклу. За таких умов навіть короткочасні збої в роботі серверного обладнання, мережевих вузлів або систем управління базами даних здатні спричинити значні фінансові втрати, паралізувати роботу персоналу та завдати шкоди репутації компанії.

В умовах жорсткої ринкової конкуренції бізнес висуває суворі вимоги до доступності сервісів, часто вимагаючи режиму роботи 24/7. Однак у багатьох корпоративних мережах, особливо в сегменті малого та середнього бізнесу, досі зберігаються застарілі підходи до адміністрування. Традиційна модель експлуатації часто базується на реактивному реагуванні, коли технічні спеціалісти починають діяти лише після виникнення аварії та отримання скарг від користувачів. Такий підхід робить системних адміністраторів "сліпими" до стану мережі, унеможливлює прогнозування навантажень та перетворює процес підтримки на постійну боротьбу з наслідками інцидентів, а не з їхніми причинами.

Вирішення цієї проблеми полягає в переході до проактивної моделі управління, що реалізується шляхом впровадження комплексних систем моніторингу. Такі системи дозволяють у реальному часі збирати, обробляти та візуалізувати тисячі метрик стану апаратного та програмного забезпечення. Серед широкого спектра існуючих інструментів особливої уваги заслуговує платформа Zabbix. Вона поєднує в собі можливості корпоративного рівня, відкритий вихідний код, високу масштабованість та гнучкість налаштувань. Впровадження

Zabbix дозволяє автоматизувати процес виявлення аномалій, значно скоротити час реакції на інциденти та забезпечити прозорість роботи інфраструктури для керівництва та технічного персоналу.

Таким чином, тема дипломної роботи, присвячена розробці та дослідженню комплексної системи моніторингу, є вкрай актуальною. Вона спрямована на вирішення прикладного інженерного завдання - забезпечення надійності бізнесу через технологічну прозорість та керованість IT-активів.

Мета і завдання дослідження. Метою роботи є підвищення єфективності, продуктивності та відмовостійкості корпоративної IT-інфраструктури шляхом розробки, налаштування та впровадження автоматизованої системи моніторингу на базі платформи Zabbix, що забезпечить мінімізацію часу простою сервісів та проактивне виявлення несправностей.

Для досягнення поставленої мети в роботі необхідно вирішити низку взаємопов'язаних завдань. Першочерговим завданням є проведення глибокого аналізу предметної області, що включає дослідження ролі моніторингу в сучасних мережах, визначення критичних об'єктів інфраструктури та порівняльний аналіз існуючих інструментальних засобів, таких як Nagios, Prometheus та Zabbix. На основі проведеного аналізу наступним кроком є проєктування архітектури системи моніторингу, яке передбачає розробку логічної схеми корпоративної мережі, визначення схеми взаємодії компонентів Zabbix, розрахунок вимог до апаратного забезпечення та проєктування заходів із забезпечення високої доступності та інформаційної безпеки.

Важливим етапом роботи є безпосередня програмна реалізація та налаштування системи. Це завдання включає інсталяцію Zabbix-сервера на обрану платформу, налаштування бази даних, розгортання агентів моніторингу на серверах під управлінням операційних систем Linux та Windows, а також організацію збору метрик з мережевого обладнання. Окрему увагу в роботі приділено реалізації інтелектуальної системи сповіщень. Це завдання передбачає розробку та впровадження логіки фільтрації повідомлень з використанням системи тегів, що дозволяє автоматизувати маршрутизацію інцидентів до

відповідних груп технічних спеціалістів залежно від типу проблеми.

Завершальним завданням є дослідження ефективності розробленої системи. Для цього необхідно провести експериментальне моделювання типових аварійних ситуацій та виконати порівняльний аналіз часових показників реагування на інциденти до та після впровадження системи, що дозволить кількісно оцінити отримані результати.

Об’єкт, предмет та методи дослідження. Об’єктом дослідження є процес управління працездатністю, доступністю та продуктивністю компонентів гетерогенної корпоративної IT-інфраструктури.

Предметом дослідження є механізми, інструменти, алгоритми та архітектурні рішення для побудови автоматизованої системи моніторингу стану комп'ютерних мереж та серверного обладнання з використанням платформи Zabbix.

Механізми дослідження, використані в роботі, включають механізми системного аналізу для формування вимог до системи, механізми порівняльного аналізу для обґрунтування вибору програмного забезпечення, механізми моделювання для проєктування архітектури та сценаріїв збоїв, а також експериментальний алгоритм для перевірки працездатності системи та оцінки її ефективності на реальних кейсах.

Практичне значення та прикладна цінність роботи. Практичне значення одержаних результатів полягає у створенні та впровадженні повністю функціонального інструментарію для контролю стану корпоративної мережі, який дозволяє комплексно вирішити проблему відсутності оперативного контролю за IT-активами. Результати роботи можуть бути використані для модернізації процесів технічної підтримки на підприємствах різного масштабу.

Прикладна цінність роботи визначається кількома ключовими аспектами. По-перше, розроблено уніфікований інженерний механізм розгортання системи моніторингу, яка враховує сучасні вимоги до відмовостійкості та безпеки, що дозволяє масштабувати рішення без значних капітальних витрат. По-друге, впроваджено оптимізований механізм реагування на інциденти через систему

тегування подій. Такий підхід автоматизує розподіл заявок між лініями підтримки, гарантуючи, що проблеми фізичного рівня потрапляють до лінійних інженерів, а логічні помилки - до системних адміністраторів, що суттєво зменшує час простою.

По-третє, робота демонструє прямий економічний ефект від впровадження проактивного моніторингу. Здатність системи виявляти потенційні проблеми, такі як вичерпання ресурсів або деградація продуктивності дисків, дозволяє запобігати аваріям ще до моменту зупинки бізнес-процесів, зберігаючи кошти компанії. Крім того, створена система забезпечує централізацію даних, об'єднуючи в єдиному інтерфейсі показники з різнорідних джерел, що надає керівництву об'єктивну картину стану інфраструктури для прийняття управлінських рішень. Доказова база ефективності запропонованих рішень підтверджується проведеним експериментальним дослідженням та порівнянням метрик швидкодії реагування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ІНСТРУМЕНТАРІЮ

1.1 Роль моніторингу в управлінні IT-інфраструктурою підприємства

Сучасне підприємство функціонує в умовах повної залежності від інформаційних технологій. Інфраструктура більше не є допоміжним елементом, а виступає фундаментом для надання послуг, логістики, фінансових операцій та комунікації. У такому середовищі поняття "якість сервісу" нерозривно пов'язане з поняттям "контрольованість інфраструктури". Система моніторингу в цьому контексті виступає ключовим інструментом, що забезпечує зворотний зв'язок між апаратним/програмним забезпеченням та персоналом, який ним управляє. Відсутність ефективного моніторингу призводить до ситуації невизначеності, коли управлінські рішення приймаються на основі здогадок, а не фактів [1].

Аналіз стандартів ITIL/ITSM у контексті моніторингу. Для побудови системи управління IT-послугами світова практика спирається на бібліотеку передового досвіду ITIL (Information Technology Infrastructure Library) [2]. У рамках методології ITSM (IT Service Management), моніторинг не є ізольованим технічним завданням, а інтегрований у ключові процеси експлуатації послуг (Service Operation). Фундаментальними процесами, що безпосередньо залежать від системи моніторингу, є управління подіями (Event Management) та управління інцидентами (Incident Management) [3].

Процес управління подіями (Event Management) є основою для будь-якої автоматизації. Подія в термінології ITIL визначається як будь-яка зміна стану, що має значення для управління конфігураційною одиницею або IT-послугою. Система моніторингу виступає головним джерелом генерації цих подій. Без автоматизованого засобу виявлення подій адміністратори змушені покладатися на ручні перевірки, що є неефективним та ненадійним підходом [4].

Класифікація подій згідно з ITIL дозволяє налаштувати логіку системи моніторингу для фільтрації інформаційного шуму [5]:

– інформаційні події, які фіксують нормальне виконання операцій та не

вимагають негайної реакції, але важливі для аналітики;

– попереджувальні події, що сигналізують про наближення показників до порогових значень, наприклад, заповнення дискового простору на 80 %;

– події-виключення, які вказують на те, що сервіс або пристрій працює некоректно, наприклад, відмова жорсткого диска в RAID-масиві [3].

Ефективно налаштована система моніторингу повинна автоматично класифікувати отримані метрики та трансформувати їх у відповідні події. Це дозволяє реалізувати перехід від простого спостереження до управління. Якщо система фіксує подію-виключення, вона автоматично ініціює процес управління інцидентами (Incident Management) [3].

Метою процесу управління інцидентами є якнайшвидше відновлення нормальної роботи послуги та мінімізація впливу на бізнес-операції. Тут роль моніторингу полягає у зменшенні часу виявлення інциденту (MTTD - Mean Time To Detect). У класичній схемі без моніторингу ланцюжок виглядає так - виникнення збою → виявлення користувачем → дзвінок у підтримку → реєстрація інциденту → діагностика. Впровадження системи моніторингу змінює цю парадигму, виключаючи користувача з ланцюжка виявлення. Система автоматично генерує інцидент у момент збою, надаючи інженерам точну інформацію про місце та причину проблеми ще до того, як це відчує бізнес [6].

Класифікація видів моніторингу.

Комплексний підхід до контролю IT-інфраструктури вимагає багаторівневої моделі моніторингу. Недостатньо контролювати лише фізичну доступність серверів, оскільки "зелений" індикатор утиліти ping не гарантує, що веб-сайт відкривається, а база даних обробляє транзакції. Сучасні системи класифікують моніторинг за трьома основними рівнями - інфраструктурний, прикладний та бізнес-моніторинг. Інфраструктурний моніторинг (Infrastructure Monitoring) є базовим рівнем, що відповідає за контроль стану апаратного забезпечення та середовища віртуалізації. На цьому рівні збираються низькорівневі метрики, які характеризують "здоров'я" компонентів. До основних об'єктів інфраструктурного моніторингу належать [4]:

– серверне обладнання, де контролюються показники температури процесорів, швидкості обертання вентиляторів та статуси блоків живлення;

– ресурси операційних систем, включаючи завантаження центрального процесора (CPU Load), використання оперативної пам'яті (RAM usage) та черги дискових операцій (Disk I/O latency);

– мережеве обладнання, для якого критичними є метрики утилізації каналів зв'язку, кількості помилок на портах та стан протоколів маршрутизації.

Прикладний моніторинг (Application Performance Monitoring - APM) фокусується на роботі програмного забезпечення та сервісів. Цей рівень дозволяє зрозуміти не те, чи працює сервер, а те, наскільки якісно працює додаток на ньому. Часто виникають ситуації, коли сервер має достатньо вільних ресурсів, але додаток працює повільно через блокування в базі даних або помилки в коді. Завданнями прикладного рівня є [7]:

– контроль доступності мережевих сервісів, таких як HTTP, SMTP, DNS, шляхом перевірки відповіді на синтетичні запити;

– аналіз продуктивності баз даних, що включає моніторинг кількості повільних запитів, використання кешу та стану реплікації;

– моніторинг роботи веб-серверів та серверів застосунків, зокрема відстеження кількості активних сесій, кодів відповідей веб-сервера (5xx, 4xx) та часу генерації сторінки.

Бізнес-моніторинг (Business Activity Monitoring) є вищим рівнем абстракції, який перекладає технічні метрики на мову бізнес-показників. Для керівництва компанії технічні деталі про завантаження процесора часто не несуть корисної інформації. Їм важливо знати, чи відбуваються продажі та чи працює логістика. На цьому рівні відстежуються такі агреговані показники:

– кількість успішних транзакцій або замовлень за одиницю часу, що дозволяє миттєво побачити проблеми в бізнес-логіці;

– кількість активних користувачів у системі, зниження якої може свідчити про проблеми з доступом у певному регіоні;

– швидкість проходження бізнес-процесу, наприклад, час від моменту

оформлення заявки до її передачі на склад.

Інтеграція всіх трьох рівнів у єдину систему дозволяє реалізувати принцип наскрізної видимості (Observability). Коли система моніторингу показує кореляцію між зростанням затримки дискової підсистеми (інфраструктура), збільшенням часу виконання SQL-запитів (додаток) та падінням кількості замовлень (бізнес), адміністратор може миттєво локалізувати проблему [6].

Вплив простоїв на бізнес та вартість відмови. Відсутність надійної системи моніторингу неминуче призводить до збільшення часу простою (Downtime). У сучасній цифровій економіці час - це прямий фінансовий еквівалент. Вплив інцидентів на бізнес можна розділити на прямі та непрямі втрати. Прямі втрати легко піддаються розрахунку - це недоотриманий прибуток за час простою сервісу та вартість оплати праці співробітників, які не можуть виконувати свої обов'язки. Непрямі втрати включають репутаційні збитки, втрату лояльності клієнтів та можливі штрафні санкції за порушення умов надання послуг (SLA) [8].

Для формалізації вимог до надійності використовується угода про рівень послуг (Service Level Agreement – SLA), яка визначає гарантований відсоток доступності сервісу. Наприклад, доступність 99,9 % дозволяє сервісу бути недоступним не більше 43 хвилин на місяць. Підвищення рівня доступності до 99,99 % ("чотири дев'ятки") скорочує допустимий час простою до 4 хвилин на місяць, що вимагає миттєвої реакції на збої, неможливої без автоматизованого моніторингу.

Важливими метриками, що описують вплив простоїв, є RTO (Recovery Time Objective) та RPO (Recovery Point Objective). У контексті моніторингу ключову роль відіграє скорочення часових інтервалів реагування [3]:

– час виявлення (Time to Detect), який без автоматизації може складати години, а з системою моніторингу - секунди;

– час діагностики (Time to Diagnose), який скорочується завдяки наявності історичних даних та логів у системі моніторингу;

– час відновлення (Time to Repair), що залежить від точності локалізації проблеми [4].

Статистичні дані дослідницьких агентств (Gartner, IDC) свідчать, що середня вартість години простою для підприємств середнього бізнесу може сягати десятків тисяч доларів. Графік залежності фінансових втрат від часу реакції має нелінійний характер - перші хвилини простою можуть бути некритичними, але зі збільшенням часу втрати зростають експоненціально через каскадний ефект зупинки суміжних бізнес-процесів, рис 1.1.

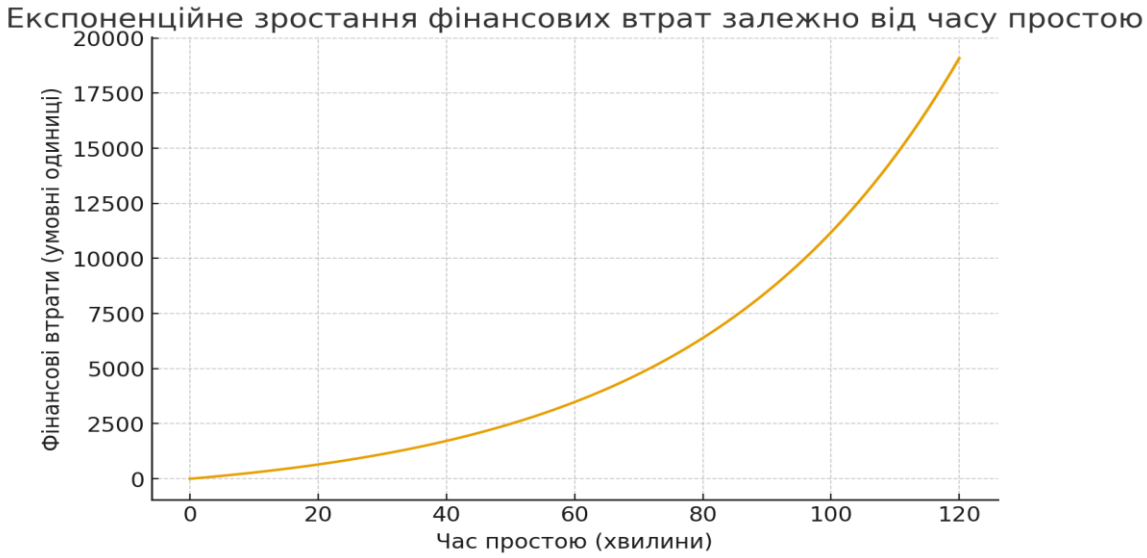

Рисунок 1.1 - Графік зростання вартості години простою залежно від масштабу підприємства

Без впровадження системи моніторингу компанія перебуває в зоні високого ризику. Адміністратори працюють у режимі "гасіння пожеж", що призводить до професійного вигорання та збільшення кількості помилок через людський фактор. Впровадження інструментарію на кшталт Zabbix дозволяє перейти від парадигми "Availability Monitoring" (чи працює сервер?) до "Performance Monitoring" (як працює сервер?) та "Business Service Monitoring" (як почувається бізнес?), що є необхідною умовою для забезпечення конкурентоспроможності підприємства [6].

1.2 Аналіз поточного стану інфраструктури об'єкта дослідження

Ефективність впровадження будь-якої інформаційної системи залежить від чіткого розуміння вихідних умов. Для проєктування комплексної системи моніторингу необхідно детально проаналізувати поточну архітектуру корпоративної мережі, наявні апаратні та програмні засоби, а також процеси експлуатації, що склалися на підприємстві. Об'єктом дослідження у цій роботі виступає типова корпоративна мережа середнього підприємства, яка характеризується гетерогенністю використовуваних технологій та динамічним зростанням кількості сервісів [3, 16].

Опис моделі "як є" - архітектура та процеси адміністрування. На даний момент IT-інфраструктура підприємства є складною розподіленою системою, яка забезпечує роботу близько 150 робочих станцій, низки критично важливих бізнес-додатків, IP-телефонії та зовнішніх веб-сервісів. Мережева топологія побудована за ієрархічним принципом і включає рівень ядра, рівень агрегації та рівень доступу. Активне мережеве обладнання представлене маршрутизаторами та комутаторами різних виробників (Cisco, MikroTik, HP), що ускладнює уніфікацію управління [7].

Серверний сегмент інфраструктури включає як фізичні сервери, так і віртуальні машини, що функціонують під управлінням гіпервізорів VMware ESXi та Proxmox. Операційні системи, що використовуються на серверах, також різнорідні: більшість веб-сервісів та баз даних працюють на базі дистрибутивів Linux (Debian, Ubuntu, CentOS), тоді як інфраструктурні сервіси, такі як Active Directory, DNS, DHCP та термінальні сервери для бухгалтерського обліку, розгорнуті на базі Windows Server.

Поточна модель управління інфраструктурою ("As-Is") характеризується високим рівнем децентралізації та переважанням ручних механізмів контролю. На підприємстві відсутня єдина консоль ("Single Pane of Glass"), яка б відображала стан здоров'я всієї мережі в реальному часі. Адміністрування здійснюється реактивно, а діагностика проблем базується на ситуативному використанні розрізнених утиліт [4].

Процес моніторингу в поточному стані можна охарактеризувати наступними рисами:

– фрагментарність контролю, коли кожен системний адміністратор відповідає за свій сегмент мережі та використовує власні інструменти для перевірки працездатності, що унеможливлює отримання цілісної картини стану інфраструктури керівництвом відділу;

– відсутність історичних даних, оскільки вбудовані засоби операційних систем (Task Manager у Windows або top/htop у Linux) показують лише миттєві значення навантаження, не зберігаючи історію змін метрик у часі, що робить неможливим аналіз трендів та розслідування інцидентів постфактум;

– ручна перевірка логів, що передбачає необхідність для адміністратора заходити на кожен сервер окремо через SSH або RDP для перегляду журналів подій, що є трудомістким процесом і часто виконується лише після виникнення збою;

– залежність від користувачів, які фактично виконують роль "системи моніторингу", повідомляючи технічну підтримку про недоступність сервісів, що є неприпустимим для сучасного бізнесу, оскільки технічний відділ дізнається про проблему останнім.

Така організація роботи створює значні ризики для стабільності бізнес-процесів. Адміністратори не бачать передумов виникнення аварій, таких як поступове заповнення дискового простору логами, деградація продуктивності дискового масиву або збільшення часу відповіді бази даних. Вони змушені реагувати вже на факт зупинки сервісу, коли бізнес починає втрачати кошти [8].

Аналіз проблемних ситуацій та кейсів відмов. Для обґрунтування необхідності впровадження автоматизованої системи моніторингу було проаналізовано журнал інцидентів за останній квартал. Аналіз виявив низку типових проблем, вирішення яких зайняло невиправдано багато часу через відсутність інструментів раннього попередження та діагностики.

Одним із найбільш показових прикладів є інцидент із зупинкою роботи корпоративної бази даних. Ситуація розвивалася наступним чином:

– сервіс обробки замовлень почав працювати повільно, користувачі скаржилися на "зависання" інтерфейсу, але оскільки повного розриву з'єднання не було, адміністратори не отримали критичних сигналів;

– через дві години система повністю зупинилася, паралізувавши роботу відділу продажів;

– у ході діагностики, яка зайняла ще близько години, було виявлено, що причиною збою стала нестача вільного місця в табличному просторі бази даних через некоректно налаштовану процедуру архівації логів транзакцій;

– наслідком інциденту став тригодинний простій критичного сервісу в розпал робочого дня.

Якби в інфраструктурі була налаштована система моніторингу, адміністратор отримав би попереджувальне сповіщення (Warning) при заповненні диска на 80% і критичне сповіщення (Critical) при 90%. Це дозволило б розширити дисковий простір або очистити старі логи в плановому режимі без зупинки сервісу. В даному випадку відсутність моніторингу перетворила рутинну задачу обслуговування на аварію.

Інший характерний кейс пов'язаний із проблемами мережевої взаємодії. Періодично користувачі в одному з офісних сегментів скаржилися на повільну роботу інтернету та розриви з'єднань з файловим сервером. Проблема мала "плаваючий" характер: вона з'являлася і зникала хаотично, що ускладнювало діагностику в момент прибуття спеціаліста. Без системи моніторингу, яка збирає статистику з мережевого обладнання, виявити причину було неможливо. Лише після тривалого ручного аналізу було встановлено, що один з комутаторів рівня доступу перегрівався через вихід з ладу системи охолодження, що призводило до троттлінгу процесора та втрати пакетів. Наявність моніторингу температури обладнання та лічильників помилок на портах дозволила б виявити несправний вентилятор задовго до того, як це почало впливати на користувачів.

Також суттєвою проблемою поточної моделі є складність планування ресурсів (Capacity Planning). Керівництво компанії періодично ставить питання про необхідність закупівлі нових серверів. За відсутності зібраної статистики

утилізації CPU та RAM за тривалий період (місяць, рік), адміністратори не можуть обґрунтовано відповісти, чи дійсно наявні потужності вичерпано, або ж проблема криється в неоптимізованому програмному коді. Рішення про модернізацію приймаються інтуїтивно, що часто призводить до неефективних витрат бюджету [6].

Формалізація технічних вимог до системи моніторингу. На основі аналізу поточного стану інфраструктури та виявлених проблем було сформовано перелік технічних вимог до нової системи моніторингу. Система повинна вирішити завдання централізації контролю, автоматизації виявлення збоїв та забезпечення аналітичних даних для прийняття рішень.

Вимоги до функціональності системи включають:

– підтримку мультиплатформенного моніторингу, що передбачає можливість збору даних з операційних систем Linux, Windows, а також мережевого обладнання через протоколи SNMP, IPMI та ICMP [10];

– наявність механізму візуалізації, що дозволяє будувати графіки, діаграми та комплексні дашборди для відображення стану системи в реальному часі;

– гнучку систему сповіщень, яка підтримує відправку повідомлень через сучасні канали зв'язку (месенджери Telegram, Slack, електронна пошта) та дозволяє налаштовувати ескалацію інцидентів [11, 12];

– можливість розширення, що дозволяє створювати власні сценарії перевірки (скрипти) для специфічного корпоративного програмного забезпечення.

Критично важливими є вимоги до рівня обслуговування (SLA - Service Level Agreement) та часових показників реакції. Цільовим показником доступності критичних сервісів встановлено рівень 99,9%, що допускає сумарний час простою не більше 8 годин 46 хвилин на рік. Для досягнення цього показника система моніторингу повинна забезпечувати наступні параметри швидкодії:

– інтервал опитування критичних метрик (доступність хоста, завантаження CPU) не повинен перевищувати 60 секунд;

– час обробки події та генерації сповіщення (Reaction Time) має складати менше 5 хвилин з моменту виникнення інциденту;

– система повинна гарантувати доставку сповіщення відповідальному адміністратору незалежно від стану корпоративної пошти (використання зовнішніх каналів зв'язку).

Вимоги до зберігання даних (Retention Policy) формуються виходячи з потреб аналізу інцидентів та планування потужностей. Система повинна забезпечувати дворівневу схему зберігання історії:

– детальну історію значень метрик (History) необхідно зберігати протягом 14 днів для можливості детального розбору недавніх інцидентів;

– усереднені тренди (Trends), що показують мінімальні, максимальні та середні значення за годину, повинні зберігатися не менше 365 днів (1 рік).

Це дозволить проводити річний аналіз зростання навантаження, виявляти сезонні піки активності та обґрунтовано планувати бюджет на модернізацію IT-інфраструктури.

Окремий блок вимог стосується безпеки самої системи моніторингу. Оскільки Zabbix матиме доступ до великої кількості серверів, він стає критичним елементом безпеки. Необхідно забезпечити шифрування трафіку між сервером моніторингу та агентами, розмежування прав доступу користувачів (Read-Only для операторів, Full Admin для старших інженерів) та аудит дій адміністраторів у самій системі [6, 15].

Впровадження системи, що відповідає зазначеним вимогам, дозволить перейти від моделі ручного управління до автоматизованого контролю якості надання IT-послуг.

1.3 Порівняльна характеристика сучасних систем моніторингу

Ринок програмного забезпечення для моніторингу IT-інфраструктури сьогодні пропонує десятки рішень, починаючи від простих утиліт для перевірки доступності хостів (ping-checkers) і закінчуючи складними платформами класу

AIOps (Artificial Intelligence for IT Operations). Вибір конкретного інструменту є стратегічним рішенням, оскільки зміна системи моніторингу в майбутньому вимагатиме значних витрат часу на міграцію даних, переналаштування агентів та навчання персоналу. Тому перед детальний оглядом конкретних продуктів необхідно визначити критерії, за якими здійснюється оцінювання [10].

Критерії вибору системи моніторингу. Для об'єктивного порівняння програмних продуктів доцільно виділити ряд ключових технічних та економічних показників, які є критичними для побудови корпоративної системи нагляду.

До основних критеріїв вибору належать:

– сукупна вартість володіння (TCO);

– архітектурна масштабованість;

– поріг входження та зручність використання;

– функціональна гнучкість та підтримка протоколів.

Сукупна вартість володіння є визначальним фактором для багатьох організацій. Вона складається не лише з ціни ліцензії на програмне забезпечення, але й з витрат на апаратне забезпечення, технічну підтримку та оплату праці спеціалістів. Комерційні продукти часто мають модель ліцензування, що залежить від кількості сенсорів або хостів, що може призвести до експоненціального зростання витрат при розширенні мережі. Open Source рішення, хоч і є безкоштовними, часто вимагають більш кваліфікованого персоналу для налаштування, що збільшує операційні витрати (OPEX).

Архітектурна масштабованість визначає здатність системи обробляти зростаючий обсяг даних без деградації продуктивності. Важливо оцінити, чи зможе система підтримувати моніторинг не лише поточної кількості вузлів (наприклад, 100 серверів), але й потенційного навантаження через 3–5 років (наприклад, 1000 серверів та 50 000 метрик). Системи, що не підтримують розподілену архітектуру або кластеризацію, стають вузьким місцем інфраструктури.

Поріг входження та зручність використання впливають на швидкість впровадження системи. Деякі інструменти орієнтовані на досвідчених DevOps-

інженерів і керуються виключно через конфігураційні файли та командний рядок (CLI). Інші мають зручний графічний інтерфейс (GUI) та майстри налаштування, що дозволяє працювати з ними спеціалістам середньої кваліфікації.

Функціональна гнучкість визначає здатність системи працювати з гетерогенним середовищем. Корпоративна мережа зазвичай містить суміш технологій: Windows та Linux сервери, бази даних різних вендорів, мережеве обладнання, джерела безперебійного живлення тощо. Система моніторингу повинна підтримувати стандартні протоколи (SNMP, IPMI, WMI, JMX) "з коробки" або мати механізми розширення через скрипти та плагіни.

Детальний огляд Nagios. Nagios Core тривалий час вважався промисловим стандартом у світі Open Source моніторингу. Це одна з найстаріших систем, архітектура якої вплинула на розвиток цілого класу програмного забезпечення (так звані "форки" Nagios, наприклад, Icinga) [13].

Архітектура Nagios базується на:

– ядрі системи (Core Scheduler);

– плагінах виконання перевірок;

– текстових конфігураційних файлах.

Основна ідея Nagios полягає в тому, що саме ядро не виконує перевірок. Воно лише запускає зовнішні скрипти (плагіни), які повертають код стану (0 - OK, 1 - Warning, 2 - Critical) та текстове повідомлення. Це забезпечує виняткову гнучкість: адміністратор може написати перевірку на будь-якій мові програмування (Bash, Perl, Python), і Nagios зможе її виконати [9, 14].

Однак, для сучасних динамічних середовищ Nagios має суттєві недоліки. Головним з них є складність конфігурації. Усі налаштування зберігаються у текстових файлах зі складним синтаксисом. Додавання нового хоста вимагає редагування файлів та перезавантаження сервісу. У великих інфраструктурах це призводить до помилок людського фактору ("configuration drift"). Крім того, базовий інтерфейс Nagios є морально застарілим і не надає сучасних засобів візуалізації графіків, що змушує використовувати сторонні надбудови.

Детальний огляд Prometheus. Prometheus - це система моніторингу нового

покоління, яка стала стандартом де-факто для хмарних середовищ (Cloud Native) та контейнеризації (Kubernetes). Вона принципово відрізняється від класичних систем підходом до збору даних [14].

Ключові особливості Prometheus:

– модель збору даних Pull (система сама забирає дані);

– база даних часових рядів (TSDB) [13];

– мова запитів PromQL.

На відміну від систем, де агенти надсилають дані на сервер (Push-модель), Prometheus періодично опитує кінцеві точки (exporters) через HTTP-запити. Це ідеально підходить для моніторингу мікросервісів, які динамічно створюються і знищуються. База даних Prometheus оптимізована для запису величезної кількості числових метрик, що робить її надзвичайно продуктивною [14].

Проте для класичної корпоративної інфраструктури ("legacy") Prometheus має обмеження. Він не заточений під моніторинг "заліза" через SNMP або WMI у традиційному розумінні. Для кожного типу обладнання потрібно встановлювати та налаштовувати окремий "експортер". Крім того, Prometheus фокусується на зберіганні оперативних даних (зазвичай до 14-30 днів) і не призначений для довгострокового зберігання історії за роки без додаткових компонентів (наприклад, Thanos або VictoriaMetrics). Відсутність вбудованої системи керування користувачами та дашбордів (зазвичай використовується в парі з Grafana) також ускладнює його використання як єдиної самостійної системи "з коробки" [21-22].

Детальний огляд PRTG Network Monitor. PRTG від компанії Paessler AG є прикладом комерційного продукту "все в одному". Це рішення орієнтоване на максимальну простоту розгортання та використання, що робить його популярним у сегменті малого та середнього бізнесу (SMB) [15].

Переваги PRTG включають:

– швидкий старт (автоматичне виявлення мережі);

– інтуїтивно зрозумілий веб-інтерфейс;

– широкий набір попередньо налаштованих сенсорів.

Архітектура PRTG базується на концепції "сенсорів". Сенсор - це один аспект моніторингу на одному пристрої (наприклад, завантаження CPU на сервері - це один сенсор, вільне місце на диску - інший). Ліцензування продукту також базується на кількості сенсорів [15].

Головним недоліком PRTG для великих проєктів є вартість та платформа. Серверна частина PRTG працює виключно на ОС Windows, що вимагає додаткових ліцензійних витрат на серверну ОС та ресурсів (Windows зазвичай споживає більше ресурсів, ніж Linux). Крім того, при масштабуванні понад 5-10 тисяч сенсорів вартість ліцензії стає значною, а продуктивність одного ядра системи може знижуватися, вимагаючи встановлення додаткових зондів (probes) та ускладнення архітектури [15].

Детальний огляд Zabbix. Zabbix позиціонується як система моніторингу корпоративного класу з відкритим вихідним кодом. Це універсальне рішення, яке намагається поєднати гнучкість Nagios, продуктивність сучасних систем та функціональність комерційних продуктів.

Особливості архітектури Zabbix:

– централізоване зберігання конфігурації та даних у реляційній базі даних (MySQL, PostgreSQL) [10-11];

– підтримка як агентного (Agent), так і безагентного (SNMP, IPMI, SSH) моніторингу;

– вбудовані засоби візуалізації та картографії.

Zabbix використовує гібридну модель збору даних (Push та Pull), що дозволяє адаптуватися до різних мережевих топологій. Важливою перевагою є наявність Zabbix Proxy - легковагого компонента, який може збирати дані у віддалених філіях і буферизувати їх у разі втрати зв'язку з центральним сервером. Це робить Zabbix ідеальним для розподілених мереж.

Система має потужний механізм низькорівневого виявлення (LLD), який автоматично створює елементи даних та тригери для нових файлових систем, мережевих інтерфейсів або віртуальних машин. Це значно спрощує адміністрування великих парків обладнання.

Порівняльна таблиця систем моніторингу. Для наочного порівняння розглянутих систем складено зведену таблицю, що відображає їх відповідність визначеним критеріям (таблиця 1.1).

Таблиця 1.1 - Порівняльна характеристика систем моніторингу

Характеристика	Nagios Core	Prometheus	PRTG Network Monitor	Zabbix
Тип ліцензії	Open Source (GPL)	Open Source (Apache)	Комерційна (Proprietary)	Open Source (GPL)
Вартість	Безкоштовно	Безкоштовно	Залежить від сенсорів (висока)	Безкоштовно
ОС Сервера	Linux/Unix	Linux/Unix	Windows Server	Linux/Unix
Архітектура	Core + Plugins	Pull model, TSDB	Core + Probes	Server + Proxy + Agents
Зберігання даних	RRD (файли)	Локальна TSDB	Власна БД	SQL (MySQL, PostgreSQL)
Механізм налаштування	Текстові файли	YAML файли	Веб-інтерфейс (GUI)	Веб-інтерфейс (GUI)
Складність освоєння	Висока	Середня/Висока	Низька	Середня
Візуалізація	Слабка (потрібні плагіни)	Базова (потрібна Grafana)	Відмінна (вбудована)	Хороша (вбудована + Grafana)
Масштабованість	Обмежена	Висока (для метрик)	Обмежена ліцензією/ОС	Висока (завдяки Proxy)
Агент моніторингу	NRPE (складний)	Node Exporter	Немає (WMI/SNMP)	Zabbix Agent (легкий)
Основна ніша	Legacy системи	Microservices, Cloud	SMB, Windows-мережі	Enterprise, Heterogeneous

Аналіз переваг та недоліків рішень. На основі проведеного огляду та порівняльної таблиці можна зробити висновки щодо доцільності використання кожної з систем у контексті даної дипломної роботи.

Аналіз Nagios показує, що попри свою стабільність, система морально застаріла. Необхідність ручного редагування конфігураційних файлів для додавання кожного нового хоста є неприйнятною для динамічного бізнесу. Відсутність вбудованих засобів для роботи з історичними даними та складнощі з масштабуванням роблять Nagios нераціональним вибором для побудови нової системи з нуля, хоча підтримка існуючих інсталяцій може тривати.

Аналіз Prometheus демонструє його беззаперечну перевагу у світі контейнеризації та мікросервісів. Проте, для об'єкта дослідження, який є

класичною корпоративною мережею з фізичними серверами, комутаторами, принтерами та джерелами безперебійного живлення, Prometheus вимагатиме встановлення великої кількості додаткових компонентів (експортерів) та налаштування зовнішнього сховища для довготривалої історії. Це ускладнює архітектуру та підвищує поріг входження для адміністраторів, звиклих до традиційних інструментів.

Аналіз PRTG свідчить про те, що це продукт з найкращим User Experience (UX) та найшвидшим стартом. Однак, модель ліцензування стає блокуючим фактором. Для моніторингу мережі середнього розміру, де кожен порт 48-портового комутатора може вважатися сенсором, вартість ліцензії може перевищити бюджет IT-відділу. Крім того, прив'язка до платформи Windows суперечить тенденції використання вільних операційних систем (Linux) для інфраструктурних сервісів.

Аналіз Zabbix дозволяє виділити його як найбільш збалансоване рішення для поставленої задачі. До ключових аргументів на користь Zabbix належать:

– повна функціональність Enterprise-рівня без ліцензійних відрахувань, що дозволяє спрямувати бюджет на апаратне забезпечення;

– універсальність, що дозволяє однаково ефективно моніторити як сучасні хмарні сервіси, так і застаріле обладнання через протокол SNMP, що є критичним для гетерогенної мережі об'єкта дослідження [6];

– наявність потужної системи шаблонів та автовиявлення (Discovery), що значно прискорює розгортання системи на сотнях вузлів;

– розвинута система прав доступу та безпеки, що відповідає вимогам корпоративної політики [2].

Таким чином, Zabbix є оптимальним вибором, який поєднує масштабованість, економічну ефективність та необхідну функціональність. Саме ця платформа буде використана як основа для розробки комплексної системи моніторингу в наступних розділах роботи.

1.4 Обґрунтування вибору платформи Zabbix

Вибір програмного забезпечення для побудови корпоративної системи моніторингу не може ґрунтуватися лише на популярності продукту. Він повинен базуватися на чіткій відповідності архітектурних можливостей платформи технічним вимогам підприємства, а також на економічній доцільності впровадження. У цьому підрозділі детально розглянуто архітектурні, економічні та функціональні переваги платформи Zabbix, які стали вирішальними факторами при її виборі як базового інструменту для дипломного проєкту.

Архітектурні особливості та модульна структура. Zabbix відрізняється від багатьох конкурентних рішень своєю модульною архітектурою. Система не є монолітним додатком, а складається з кількох незалежних компонентів, кожен з яких виконує строго відведену роль. Такий підхід забезпечує високу надійність - вихід з ладу одного компонента (наприклад, веб-інтерфейсу) не зупиняє процес збору даних та відправки сповіщень [16].

Основними компонентами архітектури Zabbix є:

– Zabbix Server;

– Zabbix Database;

– Zabbix Frontend;

– Zabbix Agent;

– Zabbix Proxy.

Zabbix Server - це центральне ядро всієї системи, написане мовою програмування C, що забезпечує високу продуктивність та низьке споживання системних ресурсів. Сервер відповідає за ініціацію збору даних (polling), прийом даних від агентів (trapping), обробку отриманих значень, обчислення тригерів та виконання дій, таких як відправка сповіщень. Важливою особливістю сервера є його здатність зберігати чергу даних у оперативній пам'яті, що дозволяє уникнути втрати інформації при короткочасних проблемах з записом у базу даних.

Zabbix Database - це сховище, де зберігаються як конфігураційні дані

(налаштування хостів, користувачів, правил), так і зібрана історія метрик. Платформа підтримує роботу з найпоширенішими реляційними базами даних - MySQL, PostgreSQL, Oracle. Відокремлення бази даних від сервера дозволяє масштабувати систему вертикально - при зростанні навантаження базу даних можна винести на окремий потужний фізичний сервер або кластер, не зупиняючи роботу самого моніторингу.

Zabbix Agent - це легковагий демон, що встановлюється безпосередньо на об'єкти моніторингу (сервери Linux, Windows). Агент має унікальну архітектуру, яка підтримує два режими роботи - пасивний та активний [17]. У пасивному режимі сервер сам звертається до агента із запитом на отримання метрики. Це класичний підхід, зручний для локальної мережі. У активному режимі агент сам отримує список завдань від сервера, збирає дані та періодично відправляє їх на сервер. Активний режим є критично важливим для моніторингу серверів, що знаходяться за NAT або брандмауерами, а також для буферизації даних. Якщо зв'язок із сервером зникає, активний агент зберігає зібрані дані у власному буфері та відправляє їх після відновлення з'єднання, що гарантує цілісність історії моніторингу [9].

Zabbix Proxy - це компонент, який робить архітектуру Zabbix ідеальною для розподілених мереж. Проксі-сервер діє як посередник між сервером та агентами. Він може бути встановлений у віддаленому офісі або філії, збирати дані з усіх локальних пристроїв та відправляти їх на центральний сервер одним потоком [9]. Використання Zabbix Proxy вирішує дві фундаментальні проблеми: – зниження навантаження на центральний сервер - проксі бере на себе частину роботи з попередньої обробки даних; – забезпечення надійності каналів зв'язку - у разі аварії на магістральному каналі проксі накопичує дані у своїй локальній базі (SQLite або MySQL) і передає їх у центр після відновлення зв'язку, запобігаючи утворенню "дірок" на графіках [9]. Архітектура заббікс на рис 1.2.

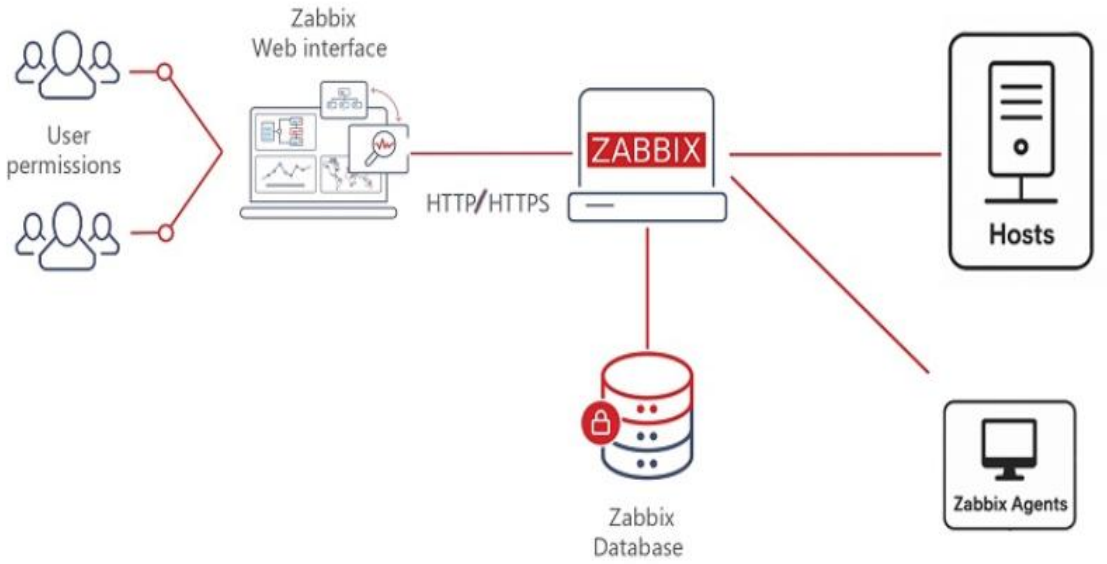

Рисунок 1.2 - Архітектура Zabbix

Економічне обґрунтування вибору. В умовах обмеженого бюджету на IT-інфраструктуру економічний аспект вибору програмного забезпечення часто стає вирішальним. Zabbix розповсюджується під ліцензією GPL v2 (General Public License version 2). Це означає, що програмне забезпечення є повністю безкоштовним для будь-якого використання, включаючи комерційне [6].

Економічна ефективність Zabbix складається з кількох факторів:

– відсутність ліцензійних платежів (CAPEX = 0);

– зниження операційних витрат на супутнє ПЗ;

– відсутність штучних обмежень (Vendor Lock-in).

На відміну від комерційних конкурентів (наприклад, PRTG Network Monitor або SolarWinds), Zabbix не має поняття "вартості за сенсор" або "вартості за хост". Це дозволяє моніторити будь-яку кількість параметрів - від 100 до 100 000 метрик - без жодних фінансових вкладень. Для підприємства, що зростає, це критично важливо - додавання нового філіалу або серверної стійки не потребує узгодження додаткового бюджету на закупівлю ліцензій моніторингу.

Другим аспектом економії є вимоги до середовища. Серверна частина

Zabbix працює на операційних системах Linux (Debian, Ubuntu, CentOS, RHEL), які також є безкоштовними. Для порівняння, багато комерційних систем моніторингу вимагають встановлення на Windows Server, що тягне за собою витрати на ліцензію ОС та, як правило, потребує більш потужного апаратного забезпечення через вищі системні вимоги Windows. Zabbix, завдяки коду на C, здатний обробляти тисячі метрик на секунду (NVPS) на обладнанні середнього рівня, що дозволяє економити на "залізі".

Відсутність прив'язки до вендора (Vendor Lock-in) також є економічною перевагою. Код продукту відкритий, що гарантує незалежність підприємства від політики однієї компанії-розробника. Навіть у випадку припинення підтримки продукту розробником, спільнота Open Source зможе підтримувати та розвивати проєкт, що захищає інвестиції часу, витраченого на впровадження системи.

Можливості кастомізації та адаптації під потреби підприємства. Кожне підприємство має унікальний набір бізнес-процесів та програмного забезпечення. "Коробкові" рішення часто не здатні задовольнити специфічні вимоги без дорогого доопрацювання. Zabbix пропонує безпрецедентний рівень гнучкості, дозволяючи адаптувати систему під будь-які нестандартні завдання.

Основою кастомізації в Zabbix є система шаблонів. Шаблон - це набір сутностей (елементів даних, тригерів, графіків, правил виявлення), які можна застосувати до групи хостів. Zabbix підтримує спадкування шаблонів, що дозволяє будувати ієрархічні моделі моніторингу. Наприклад, можна створити базовий шаблон "OS Linux", який містить перевірки процесора та пам'яті, і на його основі створити шаблони "Web Server" та "Database Server", додавши специфічні перевірки. При зміні базового параметру (наприклад, порогу спрацювання тригера вільного місця на диску) зміни автоматично застосовуються до всіх хостів у ланцюжку спадкування.

Для моніторингу нестандартних параметрів Zabbix пропонує механізм UserParameters. Це функціонал, який дозволяє адміністратору написати власний скрипт на будь-якій мові (Bash, Python, Perl, PowerShell), що повертає потрібне значення. Агент Zabbix запускає цей скрипт і передає результат на сервер. Це

дозволяє моніторити абсолютно все, що можна отримати програмним шляхом - від температури в серверній кімнаті (через USB-датчик) до кількості специфічних записів у базі даних бухгалтерської програми.

Ще однією потужною можливістю є Низькорівневе виявлення (LLD - Low Level Discovery). У динамічних середовищах конфігурація серверів постійно змінюється - додаються нові диски, віртуальні мережеві інтерфейси, контейнери. Вручну додавати ці елементи в моніторинг неможливо. LLD дозволяє Zabbix автоматично сканувати хост, знаходити нові об'єкти та створювати для них елементи даних і тригери на основі прототипів. Наприклад, якщо до сервера підключити новий жорсткий диск, Zabbix автоматично почне малювати графік його заповнення та відстежувати його стан без участі адміністратора.

Нарешті, Zabbix має повнофункціональний API (Application Programming Interface) [18]. Це дозволяє інтегрувати систему моніторингу з іншими корпоративними системами. Через API можна реалізувати такі сценарії: – автоматичне створення хостів у моніторингу при їх розгортанні через системи оркестрації (Ansible, Terraform); – виведення даних моніторингу на корпоративний портал або в систему бізнес-аналітики (BI); – двосторонню інтеграцію з Service Desk системами (Jira, GLPI) для автоматичного створення тікетів на основі інцидентів.

Враховуючи вищезазначене, вибір платформи Zabbix є повністю обґрунтованим. Вона надає необхідну архітектурну надійність завдяки проксі-серверам та буферизації, забезпечує нульову вартість ліцензування та надає інструменти для необмеженої адаптації під специфічні потреби підприємства [17].

1.5 Висновки до розділу

У першому розділі магістерської роботи проведено комплексний аналіз проблеми забезпечення надійності корпоративної IT-інфраструктури та

обґрунтовано необхідність впровадження автоматизованої системи моніторингу.

В результаті виконання досліджень на даному етапі отримано наступні результати:

– визначено, що в умовах цифрової трансформації бізнесу система моніторингу виступає критичним елементом управління, а перехід від реактивної моделі ("гасіння пожеж") до проактивної є обов'язковою умовою забезпечення конкурентоспроможності підприємства та дотримання угод про рівень послуг (SLA);

– проведено аудит поточного стану інфраструктури об'єкта дослідження та виявлено суттєві недоліки існуючої моделі експлуатації, зокрема фрагментарність контролю, відсутність історичних даних для аналітики та повну залежність процесу виявлення збоїв від повідомлень користувачів;

– сформульовано чіткі технічні вимоги до проєктованої системи, серед яких ключовими є підтримка гетерогенного середовища (Linux, Windows, мережеве обладнання), забезпечення високої доступності, збереження історії метрик протягом одного року та скорочення часу реакції на інциденти до 5 хвилин;

– виконано порівняльний аналіз провідних програмних продуктів для моніторингу (Nagios, Prometheus, PRTG, Zabbix) за критеріями вартості володіння, архітектурної гнучкості та функціональних можливостей.

На основі проведеного аналізу як базовий інструментарій для реалізації проєкту обрано платформу Zabbix. Цей вибір обґрунтовано низкою факторів: відкритим вихідним кодом, що нівелює витрати на ліцензування; модульною архітектурою з підтримкою проксі-серверів для масштабування; наявністю потужних механізмів автовиявлення та шаблонізації, які необхідні для обслуговування динамічної інфраструктури.

Таким чином, у першому розділі вирішено завдання вибору інструментарію та постановки задачі. Це створює необхідне підґрунтя для наступного етапу роботи - технічного проєктування архітектури системи моніторингу, розрахунку її параметрів та розробки схеми впровадження, що буде розглянуто у другому розділі.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ МОНІТОРИНГУ

2.1 Розробка логічної моделі корпоративної мережі

Проєктування ефективної системи моніторингу неможливе без детальної формалізації об'єкта керування. Логічна модель мережі є фундаментом, на якому будується вся архітектура Zabbix, оскільки вона визначає маршрути проходження трафіку, сегментацію сервісів та правила доступу між компонентами. У цьому підрозділі описано топологію мережі досліджуваного підприємства, проведено інвентаризацію критично важливого обладнання та визначено вузькі місця, що потребують пріоритетного контролю.

Опис топології мережі та сегментація. Корпоративна мережа підприємства побудована за класичною трирівневою ієрархічною моделлю (Core, Distribution, Access), адаптованою під масштаби середнього бізнесу. Така архітектура забезпечує масштабованість, керованість та прогнозованість поведінки трафіку.

На фізичному рівні ядро мережі (Core Layer) об'єднане з рівнем розподілу (Collapsed Core) для підвищення ефективності витрат. Воно реалізоване на базі продуктивних маршрутизаторів, які забезпечують інтерконект між різними сегментами локальної мережі (LAN) та вихід у глобальну мережу Інтернет (WAN). Рівень доступу (Access Layer) представлений комутаторами L2, до яких підключені кінцеві пристрої користувачів, точки доступу Wi-Fi, принтери та IP-телефони.

Для підвищення рівня безпеки та локалізації широкомовного шторму (Broadcast domains), мережу розділено на логічні сегменти за допомогою технології VLAN (Virtual Local Area Network). Маршрутизація між VLAN здійснюється на рівні ядра з використанням списків контролю доступу (ACL) для обмеження видимості критичних сервісів.

У мережі виділено наступні основні VLAN:

– VLAN 10 (Management);

– VLAN 20 (Servers);

– VLAN 30 (Users);

– VLAN 40 (VoIP);

– VLAN 50 (Guest);

– VLAN 100 (DMZ).

Сегмент VLAN 10 (Management) є виділеною зоною для управління мережевим обладнанням та серверами. Доступ до нього дозволено лише адміністраторам та серверу моніторингу. Саме тут буде розміщено Zabbix Server, що дозволить ізолювати трафік моніторингу від користувацьких даних.

Сегмент VLAN 20 (Servers) призначено для розміщення фізичних серверів та віртуальних машин. Тут функціонують бізнес-додатки, бази даних та файлові сховища.

Сегменти VLAN 30 (Users) та VLAN 40 (VoIP) обслуговують робочі станції співробітників та IP-телефонію відповідно. Для голосового трафіку налаштовано політики QoS (Quality of Service) для забезпечення пріоритету.

VLAN 50 (Guest) та VLAN 100 (DMZ) є ізольованими зонами. Гостьовий сегмент надає лише доступ до Інтернету, а DMZ використовується для публічних веб-сервісів, захищаючи внутрішній периметр у разі зовнішніх атак.

Адресний простір мережі організовано на базі приватних IP-адрес класу A (RFC 1918), наприклад, підмережа 10.0.0.0/8 [19]. Кожен VLAN відповідає окремій підмережі з маскою /24, що спрощує адміністрування та налаштування правил брандмауера [19].

Особлива увага при проєктуванні системи моніторингу приділяється VLAN 10 (Management). Саме в цьому сегменті буде розміщено Zabbix Server. Це дозволить йому опитувати мережеве обладнання через протокол SNMP та сервери через службові порти агентів, не змішуючи службовий трафік моніторингу з користувацьким трафіком. Схема мережі на рис. 2.1

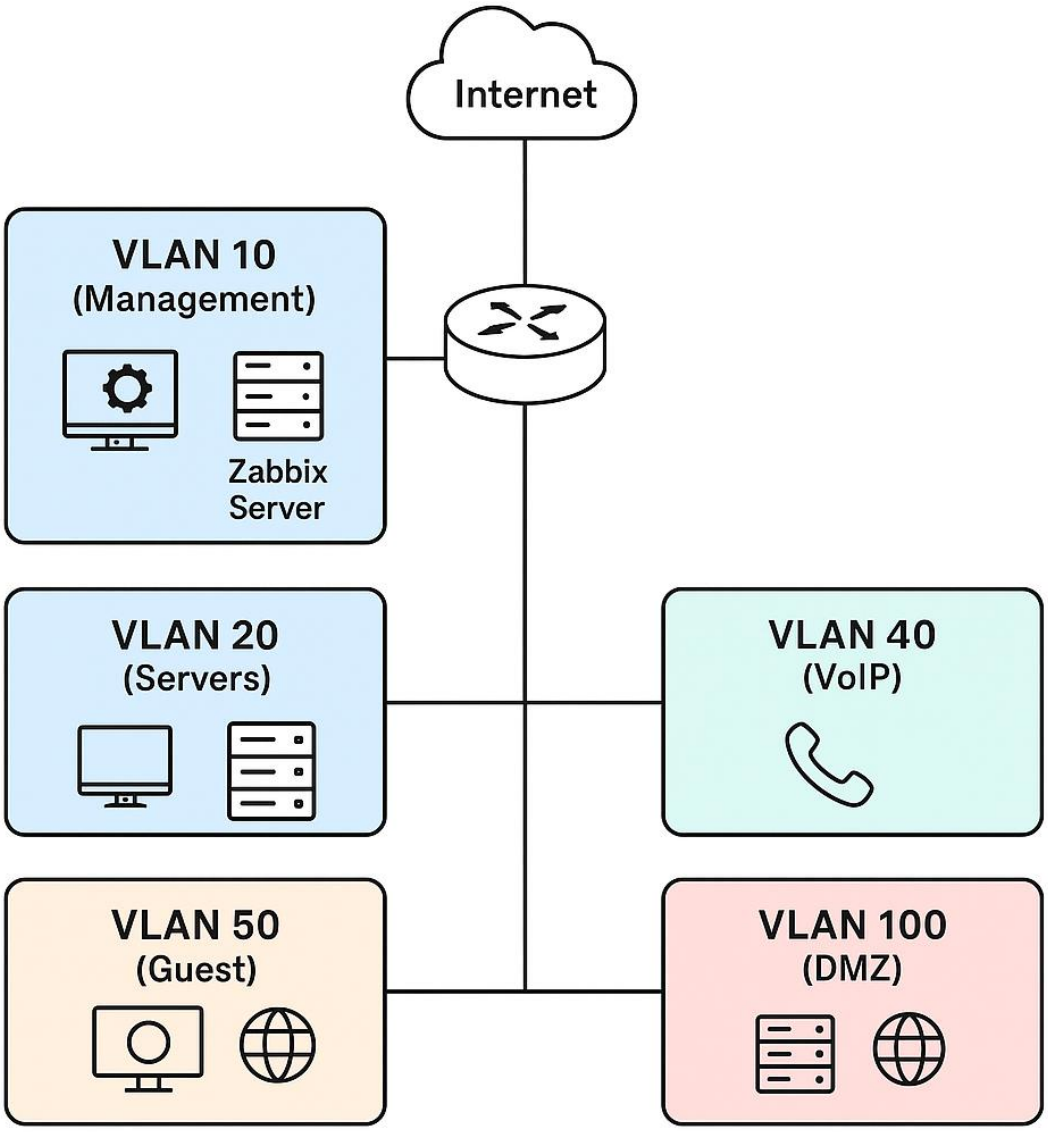

Рисунок 2.1 - Схема логічної топології мережі (VLANs та маршрутизація)

Інвентаризація обладнання та сервісів. Для коректного налаштування шаблонів Zabbix та розрахунку необхідних апаратних ресурсів (NVPS - кількість нових значень за секунду) проведено повну інвентаризацію IT-активів. Усі об'єкти моніторингу розділено на три категорії: активне мережеве обладнання, серверна інфраструктура та прикладні сервіси.

У таблиці 2.1 наведено перелік мережевого обладнання, яке є "скелетом" інфраструктури. Для цих пристроїв основним механізмом моніторингу обрано протокол SNMP v2c/v3.

Таблиця 2.1 - Інвентаризація активного мережевого обладнання

Тип обладнання	Модель / Серія	Кількість, шт.	Роль у мережі	Механізм моніторингу	Критичність
Маршрутизатор (Router)	MikroTik CCR1036	2	Ядро мережі, шлюз, VPN-концентратор	SNMP v3, ICMP	Висока
Комутатор (L3 Switch)	Cisco Catalyst 3750	2	Агрегація серверного сегменту	SNMP v3	Висока
Комутатор (L2 Switch)	HP ProCurve 2530	8	Рівень доступу (користувачі)	SNMP v2c	Середня
Wi-Fi контролер	Ubiquiti Cloud Key	1	Управління бездротовою мережею	SNMP, API	Середня
Точки доступу	Ubiquiti UniFi AP AC	12	Бездротовий доступ	Через контролер	Низька

Серверна інфраструктура підприємства є гетерогенною. Частина сервісів розгорнута на фізичних серверах ("bare metal"), проте більшість перенесена у віртуальне середовище. У таблиці 2.2 представлено перелік серверних потужностей, які підлягають моніторингу через Zabbix Agent.

Таблиця 2.2 - Інвентаризація серверного обладнання та середовищ віртуалізації

Ім'я хоста	Операційна система	Роль / Функція	Тип (Фіз./Вірт.)	ПЗ для моніторингу
SRV-HV-01	VMware ESXi 7.0	Гіпервізор кластера віртуалізації №1	Фізичний	VMware API / SNMP
SRV-HV-02	VMware ESXi 7.0	Гіпервізор кластера віртуалізації №2	Фізичний	VMware API / SNMP
SRV-DC-01	Windows Server 2019	Контролер домену (AD DS), DNS, DHCP	Віртуальний	Zabbix Agent 2 (Win)
SRV-FILE	Windows Server 2019	Файловий сервер, DFS	Віртуальний	Zabbix Agent 2 (Win)
SRV-WEB-01	Ubuntu Linux 22.04	Веб-сервер корпоративного порталу (Nginx)	Віртуальний	Zabbix Agent 2 (Lin)
SRV-DB-01	Debian 11	Сервер бази даних (PostgreSQL)	Віртуальний	Zabbix Agent 2 (Lin)
SRV-1C	Windows Server 2016	Сервер додатків облікової системи	Віртуальний	Zabbix Agent 2 (Win)
SRV-BACKUP	Synology NAS	Сервер резервного копіювання	Фізичний	SNMP

Крім "заліза" та операційних систем, моніторингу підлягають самі програмні сервіси. Користувачу не важливо, чи працює сервер, якщо сам додаток "завис". У таблиці 2.3 визначено ключові сервіси.

Таблиця 2.3 - Перелік критичних сервісів та параметрів контролю

Назва сервісу	Протокол / Порт	Параметри для моніторингу	Механізм перевірки
Корпоративний сайт	HTTPS / 443	Код відповіді (200 OK), час завантаження, термін дії SSL-сертифіката	Web scenario
База даних ERP	TCP / 5432	Кількість активних підключень, кількість транзакцій/сек, повільні запити	ODBC / Agent plugin
Електронна пошта	SMTP / 25	Доступність порту, розмір черги повідомлень	Simple check / Agent
VPN доступ	UDP / 1194	Кількість активних тунелів, статус сервісу	SNMP / Script
Active Directory	TCP / 389	Доступність LDAP, статус реплікації	Agent Template

Визначення точок відмови (SPOF). Аналіз топології та інвентаризації дозволяє виявити єдині точки відмови (Single Point of Failure - SPOF) - компоненти системи, вихід з ладу яких призводить до зупинки роботи всієї мережі або її критичної частини. Ідентифікація SPOF необхідна для налаштування пріоритетних тригерів у системі моніторингу: подія на такому вузлі повинна мати статус "Disaster" і сповіщати адміністраторів миттєво.

У поточній архітектурі виявлено наступні критичні точки:

– вхідний канал Інтернет;

– ядро мережі (Core Switch);

– система зберігання даних (СЗД) віртуалізації;

– система кондиціонування серверної.

Вхідний канал Інтернет на даний момент є вузьким місцем, оскільки підприємство використовує одного провайдера, підключеного до основного маршрутизатора. Пошкодження фізичної лінії неминуче призведе до втрати зв'язку із зовнішнім світом. Як наслідок, стануть недоступними публічні веб-сервіси, а робота віддалених співробітників буде заблокована.

Ядро мережі (Core Switch) реалізовано на якісному обладнанні, проте його поточна конфігурація не передбачає повного апаратного резервування (технології VRRP або HSRP не задіяні на повну потужність). Вихід з ладу основного пристрою призведе до ізоляції VLAN. Це спричинить повну зупинку маршрутизації між серверами та користувачами, паралізуючи роботу офісу.

Система зберігання даних (СЗД) є спільним ресурсом для всіх віртуальних машин. Хоча вона захищена RAID-масивом, вихід з ладу контролера дисків або критичне переповнення дискового простору створить ефект доміно. Це призведе до одночасної зупинки всіх віртуальних серверів, включаючи контролер домену, бази даних та веб-сервіси.

Система кондиціонування серверної наразі базується на звичайному побутовому обладнанні. Відмова кондиціонера може призвести до стрімкого підвищення температури у приміщенні. Це загрожує перегрівом дороговартісного обладнання та його аварійним вимкненням спрацюванням термозахисту. Тому необхідне встановлення незалежних датчиків температури та їх інтеграція у Zabbix.

Система моніторингу повинна компенсувати ці ризики шляхом підвищеної частоти опитування цих вузлів (інтервал Update interval - 30 секунд замість стандартних 60-300) та налаштуванням ескалації сповіщень. Наприклад, при втраті зв'язку з основним маршрутизатором сповіщення має надходити не лише черговому адміністратору, але й керівнику IT-відділу через SMS-шлюз.

Розроблена логічна модель та карта вразливостей дозволяють перейти до наступного етапу - безпосереднього проєктування архітектури компонентів Zabbix.

2.2 Проєктування архітектури Zabbix для розподіленої інфраструктури

Враховуючи гетерогенність мережі об'єкта дослідження та наявність ізольованих сегментів (DMZ, VLANs), використання простої архітектури "Сервер-Агент" є недостатнім. Для забезпечення надійності, безпеки та зниження навантаження на канали зв'язку спроєктовано розподілену архітектуру із застосуванням проксі-серверів. Такий підхід дозволяє винести процеси опитування (polling) ближче до об'єктів моніторингу, залишивши центральному

серверу функції обробки, зберігання та візуалізації даних.

Схема взаємодії компонентів (High-Level Design). Запропонована архітектура базується на ієрархічній моделі, де вершиною є центральний Zabbix Server, а збір даних на периферії здійснюють Zabbix Proxies. Це дозволяє подолати обмеження мережевої безпеки, такі як заборона прямих вхідних з'єднань у захищені сегменти мережі.

Високорівневий дизайн системи включає наступні рівні:

– рівень ядра (Core Layer);

– рівень агрегації даних (Aggregation Layer);

– рівень збору даних (Collection Layer).

Рівень ядра розміщено у захищеному сегменті Management VLAN. Тут функціонує кластер Zabbix Server та сервер бази даних. Це "мозок" системи, що відповідає за логіку та збереження інформації.

Рівень агрегації представлений проксі-серверами. Вони розміщені у ключових точках інфраструктури: у серверному сегменті для розвантаження ядра та у демілітаризованій зоні (DMZ) для безпечного моніторингу публічних сервісів.

Рівень збору даних складається з агентів моніторингу, встановлених на кінцевих серверах, та SNMP-інтерфейсів мережевого обладнання. Взаємодія між рівнями суворо регламентована: агенти не спілкуються напряму з центральним сервером, якщо вони знаходяться за проксі. Це спрощує налаштування брандмауерів, оскільки вимагає відкриття лише одного порту для зв'язку між проксі та сервером.

Логічна схема передбачає наступний розподіл ролей компонентів:

– Zabbix Server;

– Zabbix Frontend;

– Database Server;

– Zabbix Proxy (DMZ);

– Zabbix Proxy (Internal).

Zabbix Server виконує конфігурацію, обробку тригерів, відправку сповіщень

та запис історії в базу даних. Zabbix Frontend забезпечує візуалізацію та налаштування через веб-інтерфейс, взаємодіючи з сервером через API. Database Server зберігає конфігурацію та історичні дані, будучи ізольованим від прямого доступу з мережі. Zabbix Proxy (DMZ) збирає дані з веб-серверів у зоні підвищеного ризику, запобігаючи прямому доступу з DMZ до внутрішньої мережі. Zabbix Proxy (Internal) збирає "важкі" метрики з локальних серверів та мережевого обладнання, знижуючи навантаження на процесор центрального сервера. Діаграма архітектури заббікс зображена на рис. 2.2....

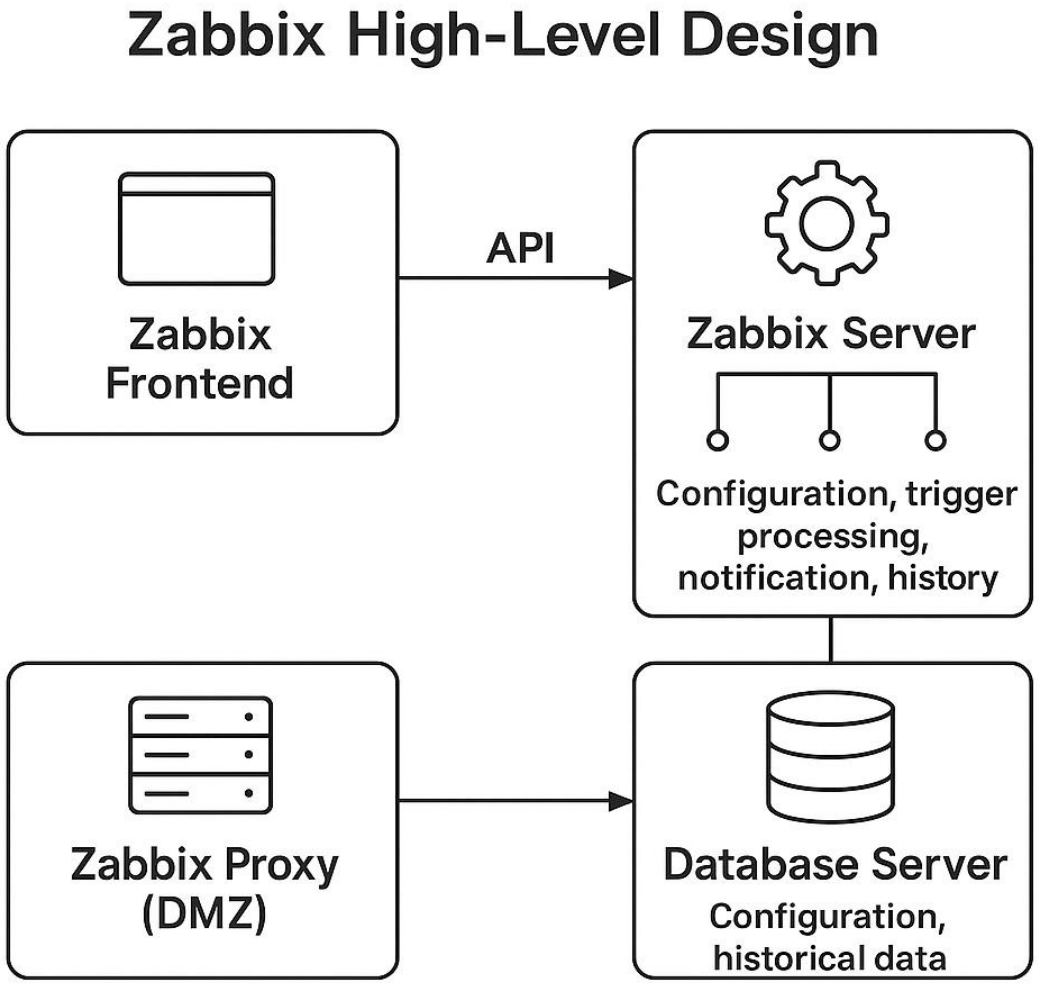

Рисунок 2.2 - High-Level Design діаграма архітектури Zabbix

Роль та розміщення Zabbix Proxy. Впровадження Zabbix Proxy є ключовим архітектурним рішенням для даного проєкту. Проксі - це незалежний процес, який збирає дані моніторингу від імені сервера. Він має власну базу даних (у даному проєкті обрано SQLite), де тимчасово зберігає зібрані метрики перед відправкою їх на сервер.

Вибір режиму роботи проксі (Active або Passive) визначає напрямок ініціації з'єднання. Для реалізації в даному проєкті обрано Активний режим (Active Proxy) для всіх проксі-серверів.

Переваги активного режиму включають:

– ініціацію з'єднання з боку проксі;

– зниження навантаження на сервер;

– підвищену живучість при збоях мережі.

Ініціація з'єднання проксі-сервером дозволяє розміщувати його за NAT або брандмауером без необхідності відкривати вхідні порти на стороні філії чи DMZ. Це значно підвищує безпеку периметра.

Зниження навантаження досягається тим, що центральний сервер не витрачає ресурси на спроби підключення до проксі, а лише очікує дані (trapping).

Підвищена живучість означає, що при втраті зв'язку активний проксі продовжує збирати дані згідно з останньою конфігурацією та буферизує їх локально. Після відновлення каналу дані будуть автоматично передані на сервер.

Стратегія розміщення проксі-серверів розроблена з урахуванням топології мережі. Zabbix Proxy 1 (DMZ-Proxy) розміщується у сегменті VLAN 100 (DMZ). Його головна задача - моніторинг публічних веб-серверів та поштових шлюзів. З точки зору мережевого екрана, дозволено лише вихідне з'єднання від DMZ-Proxy до Zabbix Server на порт 10051. Це усуває необхідність дозволяти з'єднання від внутрішнього сервера до потенційно небезпечної зони DMZ.

Zabbix Proxy 2 (Core-Proxy) розміщується у сегменті VLAN 20 (Servers). Його роль - балансування навантаження. Інфраструктура генерує значний обсяг SNMP-трафіку з комутаторів. Перекладання цієї роботи на проксі звільняє потужності центрального сервера для критично важливих задач обробки тригерів

та запису в базу даних.

Процес синхронізації конфігурації складається з таких етапів:

– підключення проксі до сервера;

– отримання пакету оновлень конфігурації;

– оновлення локальної бази проксі;

– пакетна відправка зібраних даних.

Така схема забезпечує автономність сегментів мережі та гарантує цілісність історичних даних навіть у разі тимчасових аварій на каналах зв'язку.

Потоки даних (Data Flow) від агента до бази даних. Розуміння потоків даних є необхідним для підвищення продуктивності системи та діагностики затримок. Шлях, який проходить метрика, складається з трьох основних фаз: збір, обробка та збереження.

Фаза 1 - збір даних (Collection). У випадку активної перевірки агент сам запитує список метрик, виконує перевірку та відправляє значення разом із міткою часу на порт 10051 сервера або проксі. На приймаючій стороні дані обробляє процес trapper. У випадку пасивної перевірки ініціатором виступає процес poller, який опитує агента на порту 10050.

Фаза 2 - попередня обробка (Preprocessing). Отримані дані потрапляють до менеджера попередньої обробки. Це критичний етап нормалізації інформації перед записом на диск. Основні операції на цьому етапі:

– перетворення типів даних;

– обчислення дельти;

– валідація значень;

– троттлінг даних.

Перетворення типів дозволяє привести текстові значення (наприклад, "2MB") до числового формату байтів. Обчислення дельти необхідне для розрахунку швидкості, наприклад, перетворення лічильника трафіку в біт/сек. Валідація відкидає помилкові значення, а троттлінг ігнорує дублюючі дані для економії місця в базі, якщо статус системи не змінюється.

Фаза 3 - обробка тригерів та збереження. Це найбільш ресурсоємна частина,

яку виконує процес history syncer. Він здійснює дві паралельні дії:

– оцінка тригерів;

– запис у базу даних.

При оцінці тригерів нове значення порівнюється з пороговими умовами. Якщо умова виконується, генерується подія, яка передається процесу ескалації для відправки сповіщень. Запис у базу даних відбувається пакетами (batch insert) для підвищення продуктивності СУБД. Періодично процес housekeeper видаляє застарілі детальні дані, залишаючи лише усереднені тренди для довгострокової статистики.

Використання проксі дозволяє виконати перші дві фази на стороні проксі, передаючи на сервер вже оброблені та стиснуті дані, що суттєво розвантажує центральний вузол та базу даних. Схема потоків даних на рис 2.3

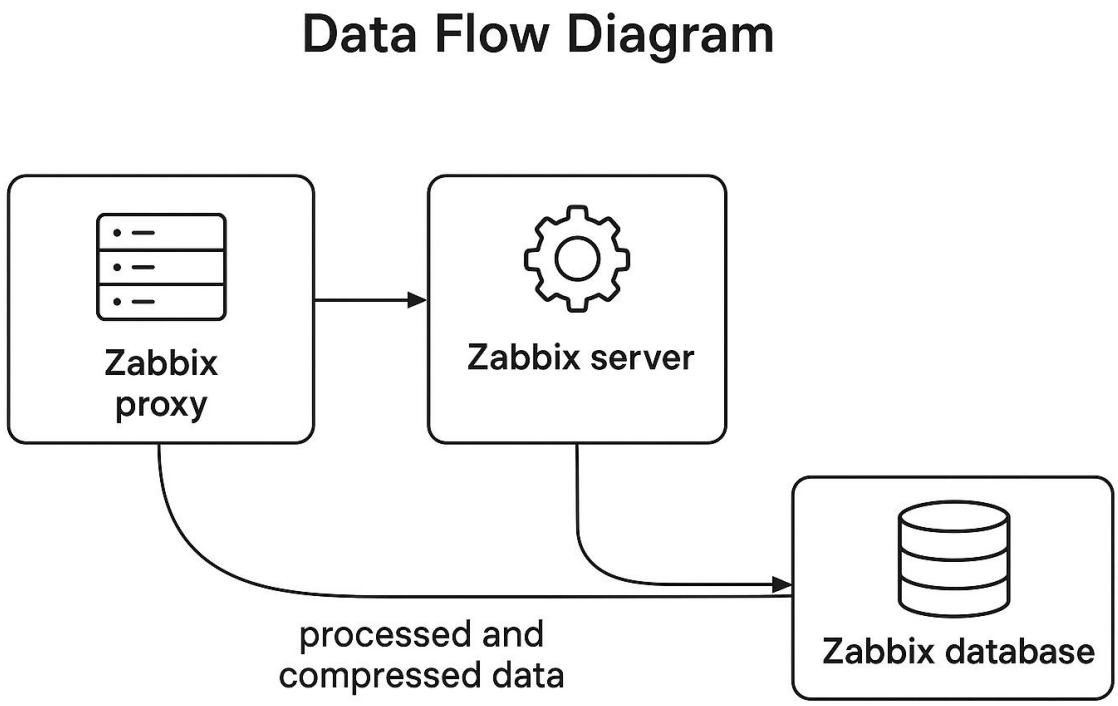

Рисунок 2.3 - Схема потоків даних (Data Flow Diagram) в Zabbix

2.3 Розрахунок навантаження та вимог до апаратного забезпечення

Правильне масштабування системи моніторингу є критичним етапом проєктування. Недостатня кількість виділених ресурсів призведе до появі "черг" (queues) у обробці даних, втрати метрик та помилкових спрацювань тригерів через затримки (nodata triggers). З іншого боку, надмірне виділення ресурсів є економічно невиправданим. Розрахунок базується на ключовому показнику продуктивності Zabbix - NVPS (New Values Per Second), який визначає кількість нових значень, що система має обробити та записати в базу даних щосекунди.

Математичний розрахунок показника NVPS. Показник NVPS не є статичною величиною і залежить від кількості хостів, кількості елементів даних (Items) на кожному хості та інтервалу їх опитування (Update Interval).

Для розрахунку загального навантаження використано інвентаризаційні дані з розділу 2.1, згруповані за типами обладнання та застосованими шаблонами моніторингу. Розрахунок виконується за формулою - NVPS = Sum(Items_i / Interval_i), де: Items_i - кількість елементів даних для і-ї групи пристроїв; Interval_i - середній інтервал опитування для цієї групи в секундах.

Проведемо детальний розрахунок для кожного сегменту інфраструктури.

Мережеве обладнання (Network Layer). До цієї групи входять маршрутизатори ядра, комутатори агрегації та доступу. Для них використовується протокол SNMP. Типовий шаблон для 48-портового комутатора збирає статистику по кожному порту (вхідний/вихідний трафік, помилки, статус), а також загальні параметри (CPU, RAM, температура). Розрахункові дані для мережевого сегменту: - кількість пристроїв - 25 одиниць (комутатори, маршрутизатори, точки доступу); - середня кількість елементів даних на пристрій - 300 (включаючи LLD правила для портів); - середній інтервал опитування - 60 секунд. NVPS_net = (25 * 300) / 60 = 125 значень/сек.

Серверна інфраструктура (Server Layer). Сюди входять фізичні гіпервізори та віртуальні машини (Linux/Windows). Використовується Zabbix Agent. Шаблони

для ОС збирають детальну статистику по процесам, пам'яті, дисковим підсистемам та мережевим інтерфейсам. Розрахункові дані для серверного сегменту: - кількість хостів - 40 одиниць (включаючи віртуальні машини та фізичні хости); - середня кількість елементів даних на хост - 150; - середній інтервал опитування - 60 секунд. NVPS_srv = (40 * 150) / 60 = 100 значень/сек.

Прикладні сервіси та веб-сценарії (Application Layer). Моніторинг баз даних, веб-сайтів та бізнес-метрик. Ці перевірки часто вимагають частішого опитування для швидкої реакції на збої. Розрахункові дані для прикладного рівня: - кількість сервісів - 20 (бази даних, веб-сайти, специфічне ПЗ); - середня кількість метрик на сервіс - 50; - середній інтервал опитування - 30 секунд (для критичних метрик). NVPS_app = (20 * 50) / 30 = 33,3 значень/сек.

Внутрішні процеси Zabbix (Internal). Система також моніторить сама себе (стан кешів, черг, процесів). Це додає фіксоване навантаження. Приймаємо експертну оцінку: NVPS_int = 20 значень/сек.

Сумарний розрахунковий NVPS - NVPS_total = 125 + 100 + 33,3 + 20 = 278,3.

Для забезпечення запасу продуктивності на випадок масштабування мережі (Growth Headroom) та сплесків активності (наприклад, під час масового перепідключення агентів), до розрахункового значення застосовується коефіцієнт запасу k = 2. NVPS_target = 278,3 * 2 = 557.

Таким чином, апаратна платформа повинна бути розрахована на стабільну обробку 600 NVPS. Це значення відповідає категорії "Medium" у класифікації Zabbix і визначає подальші вимоги до дискової підсистеми.

Розрахунок необхідного дискового простору. База даних Zabbix є найбільш вузьким місцем системи. Обсяг необхідного дискового простору складається з трьох основних компонентів: історія (History), тренди (Trends) та події (Events). Згідно з технічними вимогами, сформульованими у розділі 1, система повинна зберігати детальну історію протягом 14 днів, а тренди - протягом 365 днів.

Розрахунок обсягу таблиць історії. Розмір одного запису в таблиці історії залежить від типу даних та використовуваної СКБД. Для бази даних PostgreSQL

середній розмір запису з урахуванням індексів становить приблизно 90 байт. Формула розрахунку - Space_hist = NVPS * Bytes_hist * Time_hist, де: NVPS = 600; Bytes_hist = 90 байт; Time_hist = 14 днів * 24 * 3600 = 1,209,600 секунд. Space_hist = 600 * 90 * 1,209,600 = 65,318,400,000 байт = 60,8 ГБ.

Розрахунок обсягу таблиць трендів. Тренди займають значно менше місця, оскільки один запис зберігає чотири значення (min, max, avg, count) за одну годину. Розмір одного запису тренду становить приблизно 128 байт. Кількість елементів даних (TotalItems) можна отримати, помноживши цільовий NVPS на середній інтервал - TotalItems = 600 * 60 = 36,000. Формула розрахунку - Space_trends = TotalItems * Bytes_trend * Hours_year, де Bytes_trend = 128 байт. Hours_year = 365 * 24 = 8,760 годин. Space_trends = 36,000 * 128 * 8,760 = 40,366,080,000 байт = 37,6 ГБ. Розрахунок обсягу подій. Обсяг подій залежить від стабільності інфраструктури. Експертна рекомендація Zabbix передбачає виділення 10% від обсягу історії на події. Space_events = Space_hist * 0,1 = 6,1 ГБ Загальний обсяг бази даних - Space_total = 60,8 + 37,6 + 6,1 = 104,5 ГБ.

Враховуючи необхідність виконання операцій обслуговування БД та неминучу фрагментацію, до розрахункового обсягу необхідно додати технологічний запас 50%. Disk_req = 104,5 * 1,5 = 157 ГБ.

Отже, для системи зберігання даних необхідно виділити мінімум 160 ГБ дискового простору.

Специфікація серверного обладнання. На основі розрахованого навантаження (600 NVPS) та обсягу даних сформовано специфікацію апаратного забезпечення. Оскільки в розділі 2.2 обрано архітектуру з виділеним сервером БД та проксі-серверами, ресурси будуть розподілені між кількома віртуальними машинами.

Вимоги до дискової підсистеми є пріоритетними. База даних Zabbix створює значне навантаження на випадковий запис (Random Write). Необхідна продуктивність дисків (IOPS) розраховується за формулою - IOPS_db = NVPS * WriteMultiplier.

Для реляційних баз даних множник запису (WriteMultiplier) зазвичай

становить 3-4. IOPS_req = 600 * 4 = 2400 IOPS. Таку продуктивність не можуть забезпечити звичайні жорсткі диски. Тому обов'язковою вимогою є використання твердотільних накопичувачів SSD класу Enterprise або NVMe.

Вимоги до оперативної пам'яті (RAM) визначаються необхідністю кешування "гарячих" даних. Для оптимальної швидкодії весь обсяг активних таблиць історії (60 ГБ за розрахунком) бажано тримати в кеші БД, однак для економії ресурсів достатньо кешувати дані за останні 2-3 дні.

Специфікація віртуальної машини Zabbix Server:

– CPU - 4 vCPU (для процесів poller, trapper, history syncer);

– RAM - 8 ГБ (достатньо для кешу конфігурації та PHP-фронтенду);

– Disk - 50 ГБ (лише для ОС та логів, SSD не обов'язковий).

Специфікація віртуальної машини Database Server:

– CPU - 4 vCPU (для обробки складних SQL-запитів);

– RAM - 16 ГБ (з них 12 ГБ виділяється під буферний пул СКБД);

– Disk - 200 ГБ SSD/NVMe (із забезпеченням мінімум 3000 IOPS).

Специфікація віртуальних машин Zabbix Proxy (x2):

– CPU - 2 vCPU;

– RAM - 4 ГБ;

– Disk - 20 ГБ (для локальної бази SQLite).

Сумарні вимоги до кластера віртуалізації складають - 12 vCPU, 32 ГБ RAM та 300 ГБ дискового простору. Таке навантаження є прийнятним для поточної інфраструктури підприємства і не вимагає закупівлі нового фізичного обладнання, що підтверджує економічну ефективність рішення.

2.4 Забезпечення відмовостійкості (HA) та безпеки

Система моніторингу належить до класу критично важливих сервісів. У випадку масштабної аварії в дата-центрі саме Zabbix повинен залишатися

"останнім, хто вимкне світло", сповістивши адміністраторів про проблему. Якщо система моніторингу відмовить одночасно з падінням мережі, персонал втратить "очі" та витратить дорогоцінний час на ручну діагностику. Тому архітектура рішення повинна передбачати механізми високої доступності (High Availability - HA) та ешелонований захист від кіберзагроз.

Архітектура кластера високої доступності (Zabbix Native HA). Традиційний підхід до побудови HA-кластерів для Zabbix раніше базувався на використанні зовнішніх інструментів, таких як Pacemaker та Corosync, які керували віртуальною IP-адресою (VIP) та переміщенням сервісу між вузлами. Цей підхід є складним у налаштуванні та експлуатації. Починаючи з версії 6.0, Zabbix пропонує вбудовану підтримку кластеризації (Native HA), яка реалізована на рівні програмного коду сервера і не вимагає стороннього ПЗ для балансування [20].

Для даного проєкту обрано архітектуру Zabbix Native HA, яка складається з наступних елементів:

– активний вузол (Active Node);

– резервні вузли (Standby Nodes);

– база даних як арбітр стану.

В обраній конфігурації кластер складатиметься з двох вузлів Zabbix Server. Один з них постійно перебуває в активному стані: він збирає дані, обробляє тригери та відправляє сповіщення. Другий вузол знаходиться в режим очікування (Standby). Він запущений, має з'єднання з базою даних, але не виконує жодних операцій моніторингу.

Механізм перемикання (Failover) працює на основі серцебиття (heartbeat). Кожен вузол кластера кожні 5 секунд оновлює свій час останнього доступу в спеціальній таблиці бази даних ha_node. Алгоритм роботи відмовостійкості виглядає наступним чином:

– активний вузол блокує роботу інших вузлів, постійно оновлюючи свій статус;

– якщо активний вузол виходить з ладу (зникає живлення, зависає процес, втрачається зв'язок з БД), запис у базі даних перестає оновлюватися;

– резервний вузол фіксує відсутність оновлень від активного вузла протягом визначеного таймауту (за замовчуванням 1 хвилина);

– резервний вузол автоматично змінює свій статус на Active і перехоплює управління моніторингом.

Для агентів та проксі-серверів перемикання відбувається прозоро. У конфігураційних файлах агентів (zabbix_agentd.conf) та проксі (zabbix_proxy.conf) параметр Server вказується у вигляді списку адрес всіх вузлів кластера через крапку з комою. Агент намагається підключитися до першої адреси, і якщо вона недоступна - переходить до наступної, доки не знайде активний сервер.

Забезпечення відмовостійкості бази даних реалізується окремо засобами самої СКБД. Для PostgreSQL в даному проєкті передбачено використання потокової реплікації (Streaming Replication) з одним основним сервером (Primary) та одним реплікованим (Standby), перемикання між якими забезпечується утилітою Patroni або вручну адміністратором у разі критичної аварії [16].

Проєктування захисту: шифрування та мережевий екран. Оскільки Zabbix збирає чутливу інформацію про інфраструктуру (версії ПЗ, завантаження каналів, списки запущених процесів), цей трафік повинен бути захищений від перехоплення та підміни (Man-in-the-Middle attacks). У базовій конфігурації Zabbix передає дані у відкритому вигляді (clear text JSON), що є неприпустимим для сучасної корпоративної мережі.

Стратегія захисту даних базується на тотальному шифруванні всіх каналів зв'язку. Передбачено використання двох типів шифрування:

– Pre-Shared Key (PSK);

– Certificate-based encryption (Cert).

Для зв'язку між Zabbix Server та Zabbix Agent (Linux/Windows) обрано механізм PSK. Це рішення зумовлене простотою адміністрування та меншим навантаженням на процесор у порівнянні з RSA-сертифікатами. Для кожного агента генерується унікальний 256-бітний ключ (Identity + Key). Цей ключ прописується в конфігурації агента (TLSConnect=psk, TLSAccept=psk) та в налаштуваннях хоста через веб-інтерфейс. Якщо зловмисник спробує відправити

підроблені дані під виглядом агента, сервер відхилить з'єднання, оскільки ключ не співпаде.

Для зв'язку між Zabbix Server та Web-інтерфейсом, а також для доступу адміністраторів використовується протокол HTTPS із сертифікатами. На веб-сервері (Nginx) налаштовується примусове перенаправлення з HTTP на HTTPS та використання сучасних протоколів TLS 1.2/1.3.

Другим ешелоном захисту є налаштування міжмережевого екрана (Firewall). На кожному сервері інфраструктури моніторингу (Server, Proxy, DB) налаштовується локальний фаєрвол nftables (сучасна заміна iptables у Linux) за принципом "Default Drop" - заборонено все, що не дозволено явно.

Правила доступу для Zabbix Server включають:

– дозволити вхідні з'єднання на порт TCP/10051 лише від зареєстрованих Zabbix Proxy та агентів з внутрішніх підмереж;

– дозволити вхідні з'єднання на порт TCP/10050 лише для власних процесів (localhost);

– дозволити вхідні з'єднання на порти TCP/80 та TCP/443 (Web) лише з Management VLAN;

– дозволити вихідні з'єднання до сервера бази даних на порт TCP/5432;

– заборонити будь-які інші вхідні з'єднання.

Правила доступу для Zabbix Proxy (DMZ) є більш суворими:

– дозволити вихідні з'єднання лише на порт TCP/10051 до центрального Zabbix Server;

– дозволити вхідні з'єднання на порт TCP/10050 лише від Zabbix Server (для команд управління);

– заборонити будь-які ініціативи з'єднань у внутрішню мережу LAN.

Така конфігурація створює захищений периметр, де навіть у разі компрометації одного з агентів, зловмисник не зможе використати канал моніторингу для горизонтального переміщення мережею.

Рольова модель доступу (RBAC). Безпека системи залежить не лише від захисту периметра, але й від контролю дій користувачів всередині системи.

Zabbix надає гнучку рольову модель доступу (RBAC - Role Based Access Control), яка дозволяє розмежувати права перегляду та налаштування.

В рамках дипломного проєкту розроблено матрицю доступу, яка відходить від стандартного поділу "Адміністратор/Користувач". Визначено наступні типи користувачів (User Types):

– Zabbix User;

– Zabbix Admin;

– Zabbix Super Admin.

На базі цих типів створено специфічні ролі для співробітників компанії.

Роль "Network Engineer" (Мережевий інженер). Тип: Zabbix Admin. Ця роль призначена для адміністраторів мережевого обладнання. Права доступу: – повний доступ (Read-Write) до груп хостів "Network Devices", "Wi-Fi"; – доступ лише на читання (Read-Only) до груп "Servers", "Databases"; – право на створення та редагування карт мережі; – заборона на зміну глобальних налаштувань системи та керування користувачами.

Це дозволяє мережевим інженерам додавати нові комутатори, налаштовувати SNMP-шаблони та закривати інциденти у своїй зоні відповідальності, але не дозволяє випадково зламати моніторинг серверів або змінити пароль головного адміністратора.

Роль "L1 Support" (Перша лінія підтримки). Тип: Zabbix User. Роль для чергових операторів Service Desk. Права доступу:

– доступ на читання (Read-Only) до всіх груп хостів;

– доступ до розділу "Problems" та "Dashboards";

– право на підтвердження проблем (Acknowledge problems) та додавання коментарів;

– право на закриття проблем (Close problem), якщо це дозволено налаштуваннями тригера;

– заборона на зміну конфігурації (відсутня вкладка Configuration).

Оператори бачать усі проблеми, можуть реагувати на них, коментувати хід вирішення, але фізично не можуть змінити поріг спрацювання тригера або

видалити хост.

Роль "Management View" (Керівництво). Тип: Zabbix User. Роль для IT-директора та керівників бізнес-підрозділів. Права доступу:

– доступ лише до спеціально створених дашбордів "Business Services" та "SLA Reports";

– всі інші елементи інтерфейсу приховані через налаштування UI Elements.

Такий підхід гарантує, що кожен співробітник має доступ лише до тієї інформації та інструментів, які необхідні йому для виконання службових обов'язків, що відповідає принципу найменших привілеїв (Principle of Least Privilege).

2.5 Вибір СКБД та налаштування зберігання історичних даних

База даних є критичним компонентом архітектури Zabbix, оскільки саме вона визначає швидкість роботи веб-інтерфейсу та глибину зберігання історії. Неефективна організація бази даних призводить до проблеми "bottleneck" (вузького місця), коли сервер моніторингу не встигає записувати нові значення через високу затримку дискових операцій. У цьому підрозділі обґрунтовано вибір системи керування базами даних (СКБД) та розроблено схему партиціювання (Partitioning) для підвищення ефективності роботи з великими обсягами історичних даних.

Вибір СКБД, MySQL проти PostgreSQL. Zabbix офіційно підтримує роботу з MySQL (MariaDB), PostgreSQL та Oracle. Вибір між двома найпопулярнішими Open Source рішеннями - MySQL та PostgreSQL - базується на аналізі їхньої продуктивності при високих навантаженнях та можливостей масштабування.

Для реалізації проєкту обрано PostgreSQL (версія 14 або новіша) з розширенням TimescaleDB (опціонально) або нативним партиціюванням [21].

Ключові фактори, що вплинули на вибір PostgreSQL:

– краща ефективність складних запитів;

– надійність зберігання даних (ACID);

– ефективніший механізм очищення (Vacuum);

– нативна підтримка партиціювання.

Ефективність складних запитів у PostgreSQL реалізована краще завдяки просунутому планувальнику (Query Planner). Це важливо для Zabbix Frontend, який генерує важкі SQL-запити при побудові графіків за великий проміжок часу або при формуванні звітів SLA. PostgreSQL ефективніше використовує індекси та алгоритми з'єднання таблиць (JOINs), що забезпечує швидшу візуалізацію даних.

Надійність та відповідність стандартам ACID гарантує цілісність даних навіть у разі аварійного вимкнення живлення, що є критичним для системи, яка фіксує аварії.

Механізм Vacuum у PostgreSQL, хоч і вимагає налаштування, ефективніше справляється з проблемою "роздування" таблиць (bloat) при інтенсивних операціях оновлення (UPDATE) та видалення (DELETE), які постійно виконує процес Zabbix Housekeeper.

Нативна підтримка партиціювання дозволяє реалізувати прозорий для додатку розподіл даних по меншим таблицям, що значно спрощує адміністрування великих обсягів історії.

Налаштування Partitioning (Секціонування таблиць). Однією з найбільших проблем експлуатації Zabbix є процес очищення старих даних (Housekeeping). У стандартній конфігурації Zabbix видаляє старі записи за допомогою SQL-запитів DELETE. При базі даних розміром у сотні гігабайт це створює колосальне навантаження на диск і призводить до фрагментації індексів.

Для вирішення цієї проблеми у проєкті спроєктовано схему партиціювання (секціонування) таблиць історії. Суть механізму полягає у відмові від вбудованого механізму Housekeeper на користь фізичного видалення старих файлів таблиць.

Схема партиціювання застосовується до найбільших таблиць бази даних:

– history (числові значення з плаваючою комою);

– history_uint (цілі числа);

– history_log (текстові логи);

– history_str (короткі рядки);

– history_text (великі текстові блоки);

– trends (тренди float);

– trends_uint (тренди uint).

Алгоритм реалізації партиціювання передбачає поділ таблиць за часовим діапазоном (Range Partitioning).

Для таблиць історії (history*) обрано інтервал розбиття 1 доба. Це означає, що замість однієї гігантської таблиці history, система створює щодня нову таблицю, наприклад history_2025_01_01, history_2025_01_02. Zabbix Server пише дані у "поточну" таблицю. Старі таблиці залишаються доступними для читання. Коли термін зберігання (14 днів) спливає, спеціальний скрипт або процедура БД просто видаляє стару таблицю командою DROP TABLE, що відбувається миттєво і не створює навантаження на дискову підсистему.

Для таблиць трендів (trends*) обрано інтервал розбиття 1 місяць. Оскільки тренди займають менше місця, створювати для них щоденні таблиці недоцільно. Місячний інтервал (наприклад, trends_2025_01) є оптимальним балансом між кількістю файлів та розміром таблиці. Термін зберігання трендів -12 місяців.

Переваги впровадження партиціювання:

– стабільна продуктивність;

– миттєве очищення;

– простота резервного копіювання.

Стабільна продуктивність забезпечується тим, що розмір активної таблиці (hot data) залишається малим і сталим, що дозволяє їй повністю поміщатися в оперативну пам'ять (RAM Cache). Індекси також залишаються компактними, що прискорює пошук і вставку нових значень. Миттєве очищення звільняє ресурси сервера від виконання важких операцій DELETE та VACUUM FULL, що часто стають причиною "гальм" системи моніторингу в нічний час.

Технічна реалізація партиціювання в PostgreSQL виконується за допомогою розширення pg_partman або шляхом створення процедур (Stored Procedures) і

тригерів, які автоматично створюють нові партиції наперед. У даному проєкті розроблено SQL-скрипт, який запускається планувальником cron раз на добу для управління партиціями.

2.6 Висновки до розділу

У другому розділі магістерської роботи виконано комплексне технічне проєктування архітектури системи моніторингу на базі платформи Zabbix. Розроблено інженерні рішення, що дозволяють побудувати надійну, масштабовану та захищену систему контролю IT-інфраструктури.

В ході виконання проєктування отримано наступні результати:

– розроблено детальну логічну модель корпоративної мережі, проведено інвентаризацію понад 60 одиниць активного обладнання та сервісів, виділено критичні сегменти (VLANs) та ідентифіковано точки відмови (SPOF), що дозволило сформувати пріоритети моніторингу;

– спроєктовано розподілену архітектуру Zabbix з використанням проксі-серверів у активному режимі, що дозволило вирішити проблему безпечного моніторингу ізольованих зон (DMZ) та розвантажити центральний сервер від ресурсоємних задач опитування;

– проведено математичний розрахунок навантаження на систему, який показав прогнозний показник продуктивності на рівні 600 NVPS; на основі цього розрахунку сформовано специфікації апаратного забезпечення (12 vCPU, 32 GB RAM, 300 GB SSD), що гарантує стабільну роботу системи з урахуванням майбутнього масштабування;

– розроблено архітектуру високої доступності (High Availability) на базі вбудованого кластера Zabbix Native HA, що забезпечує безперервність моніторингу навіть у разі апаратної відмови основного вузла;

– спроєктовано багаторівневу систему безпеки, що включає шифрування

трафіку між компонентами за допомогою ключів PSK, налаштування мережевих екранів за принципом "Default Drop" та впровадження рольової моделі доступу (RBAC) для персоналу;

– обґрунтовано вибір СКБД PostgreSQL та розроблено схему партиціювання таблиць історії та трендів, що вирішує проблему продуктивності при очищенні застарілих даних і забезпечує ефективне використання дискового простору.

Результати проєктування, отримані в цьому розділі, є готовою технічною документацією для практичного впровадження системи, яке буде детально розглянуто у третьому розділі роботи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА НАЛАШТУВАННЯ СИСТЕМИ

3.1 Інсталяція та базова конфігурація серверної частини

Етап програмної реалізації є переходом від теоретичного проєктування до практичного втілення архітектури, розробленої у другому розділі. Оскільки система моніторингу має працювати в режимі 24/7 та обробляти тисячі метрик за секунду, стандартна інсталяція "за замовчуванням" (Default installation) не забезпечить необхідної продуктивності та стабільності. Цей підрозділ описує процес розгортання серверної частини на базі ОС Linux Debian, тюнінг ядра операційної системи для роботи під високим навантаженням та детальну конфігурацію компонентів Zabbix.

Підготовка операційної системи та тюнінг ядра. В якості базової операційної системи обрано дистрибутив Debian GNU/Linux 11 (Bullseye) або новіший. Вибір Debian зумовлений його репутацією найстабільнішого серверного дистрибутива, суворою політикою щодо стабільності пакунків та мінімальним споживанням ресурсів у базовій комплектації (netinstall).

Підготовка операційної системи починається з налаштування базових сервісів, критичних для роботи розподіленої системи моніторингу. Першочергові

налаштування включають:

– налаштування статичної IP-адреси;

– синхронізацію часу (NTP);

– налаштування локалізації та лімітів користувачів.

Синхронізація часу є критично важливою. Zabbix використовує мітки часу (timestamps) для кореляції подій. Розбіжність у часі між сервером, проксі та агентами навіть у кілька секунд може призвести до хибних спрацювань тригерів типу "nodata" (немає даних). Для забезпечення точності встановлено та налаштовано демон chrony, який забезпечує більш плавну корекцію часу порівняно з класичним ntp.

Після базового налаштування виконується тюнінг ядра Linux через механізм sysctl [17]. Стандартні параметри ядра Linux оптимізовані для універсального використання, а не для роботи в якості високопродуктивного сервера баз даних та моніторингу. Для обробки розрахункових 600 NVPS необхідно змінити параметри роботи з пам'яттю та мережевим стеком.

Зміни вносяться у файл /etc/sysctl.conf. Ключові параметри підвищення ефективності включають:

– vm.swappiness = 10;

– net.core.somaxconn = 65535;

– net.ipv4.tcp_max_syn_backlog = 4096;

– fs.file-max = 1000000.

Параметр vm.swappiness визначає схильність ядра до використання файлу підкачки (swap). Значення за замовчуванням (60) може призвести до того, що система почне скидати сторінки оперативної пам'яті на диск навіть за наявності вільної RAM. Для бази даних та процесів Zabbix це неприпустимо, оскільки викличе різке падіння продуктивності (latency). Встановлення значення 10 вказує ядру використовувати swap лише у випадку критичної нестачі пам'яті.

Параметр net.core.somaxconn збільшує розмір черги очікування вхідних з'єднань. Оскільки Zabbix Server постійно приймає тисячі коротких з'єднань від агентів та проксі (особливо в моменти масового відновлення зв'язку), стандартне

значення (128) може стати вузьким місцем, що призведе до відкидання пакетів. Збільшення до 65535 усуває цей ризик.

Параметр fs.file-max збільшує ліміт на кількість відкритих файлових дескрипторів. У Linux кожне мережеве з'єднання (сокет) вважається файлом. Для системи, що тримає відкритими сотні з'єднань з агентами та базою даних, стандартних лімітів недостатньо.

Після внесення змін параметри застосовуються командою sysctl -p без перезавантаження сервера.

Встановлення та конфігурація Zabbix Server. Інсталяція виконується з офіційного репозиторію розробника, що гарантує отримання найсвіжішої стабільної версії (LTS - Long Term Support) та спрощує подальші оновлення. Процес інсталяції включає додавання репозиторію, встановлення пакетів zabbix-server-pgsql, zabbix-frontend-php, zabbix-nginx-conf та zabbix-sql-scripts.

Після встановлення виконується ініціалізація бази даних. Оскільки у розділі 2 обрано PostgreSQL, необхідно створити користувача та базу даних з відповідними правами, після чого імпортувати початкову схему та дані з архіву, що постачається разом із пакетом (server.sql.gz).

Найважливішим етапом є редагування конфігураційного файлу /etc/zabbix/zabbix_server.conf. Саме тут визначається, скільки ресурсів Zabbix зможе утилізувати. Налаштування "з коробки" розраховані на маленькі інсталяції і не підходять для нашого проєкту [18].

Налаштування параметрів продуктивності (Performance parameters) включає в себе [1]:

– StartPollers = 80;

– StartPollersUnreachable = 20;

– StartTrappers = 20;

– StartPingers = 10;

– StartDiscoverers = 5;

– StartPreprocessors = 10;

– StartHistorySyncers = 4.

Параметр StartPollers визначає кількість процесів, які займаються пасивним опитуванням (SNMP, Zabbix Agent Passive). Недостатня кількість поллерів призведе до затримок збору даних (Queue). Значення 80 розраховано з запасом для покриття потреб моніторингу мережевого обладнання.

Параметр StartPollersUnreachable важливий для стабільності. Коли пристрій недоступний, процес поллера "зависає" в очікуванні таймауту. Якщо виділити під це звичайні поллери, падіння одного сегменту мережі може паралізувати моніторинг всієї інфраструктури. Окремі процеси для недоступних хостів ізолюють цю проблему.

Параметр StartTrappers відповідає за обробку вхідних даних від активних агентів та проксі. Оскільки в архітектурі (розділ 2.2) зроблено ставку на активний моніторинг, кількість цих процесів збільшено.

Параметр StartHistorySyncers визначає кількість процесів, що пишуть дані з кешу в базу даних. Значення 4 є оптимальним для 4-ядерного процесора віртуальної машини, дозволяючи уникнути конкуренції за ресурси CPU.

Налаштування параметрів кешування (Cache parameters) включає:

– CacheSize = 512M;

– HistoryCacheSize = 128M;

– TrendCacheSize = 64M;

– ValueCacheSize = 256M.

Параметр CacheSize відповідає за зберігання конфігурації (список хостів, елементів, тригерів). При великій кількості об'єктів стандартних 8 МБ недостатньо, тому виділено 512 МБ для швидкого доступу до налаштувань без звернення до БД.

Параметр ValueCacheSize є критичним для обчислення тригерів. Коли Zabbix перевіряє умову тригера (наприклад, avg(5m) > 90), він спочатку шукає дані у Value Cache. Якщо даних там немає, виконується "дорогий" запит до бази даних. Великий розмір цього кешу мінімізує навантаження на диск.

Встановлення та налаштування веб-інтерфейсу (Zabbix Web). Веб-інтерфейс Zabbix написаний на PHP і працює у зв'язці з веб-сервером. Для проєкту обрано

зв'язку Nginx + PHP-FPM, яка показує кращу продуктивність при великій кількості одночасних користувачів порівняно з Apache [22].

Налаштування Nginx виконується у файлі /etc/zabbix/nginx.conf. Крім стандартних налаштувань порту (listen 80/443), необхідно налаштувати параметри безпеки, спроєктовані у розділі 2.4, зокрема обмеження доступу по IP та налаштування SSL-сертифікатів.

Конфігурація PHP вимагає зміни лімітів у файлі /etc/php/7.4/fpm/pool.d/zabbix.conf або глобальному php.ini. Ключові параметри PHP:

– post_max_size = 16M;

– max_execution_time = 300;

– max_input_time = 300;

– memory_limit = 512M;

– date.timezone = Europe/Kyiv.

Збільшення memory_limit до 512 МБ необхідне для генерації "важких" звітів та відображення карт мережі з великою кількістю елементів. Параметр max_execution_time збільшено до 300 секунд (5 хвилин), щоб запобігти розриву з'єднання при виконанні тривалих операцій, наприклад, при оновленні версії бази даних або імпорті великих шаблонів XML. Встановлення коректної часової зони date.timezone є обов'язковим, інакше графіки в інтерфейсі не будуть співпадати з реальним часом інцидентів.

Після завершення конфігурації виконується запуск служб та додавання їх у автозавантаження (systemctl enable zabbix-server zabbix-agent nginx php7.4-fpm). Перевірка працездатності здійснюється шляхом аналізу лог-файлу /var/log/zabbix/zabbix_server.log, який не повинен містити повідомлень рівня ERROR, та спробою входу у веб-інтерфейс.

Таким чином, серверна частина системи моніторингу розгорнута, оптимізована під розрахункове навантаження та готова до підключення джерел даних.

3.2 Налаштування збору даних з різнорідних джерел

Ефективність системи моніторингу безпосередньо залежить від повноти та якості даних, що надходять до неї. Оскільки корпоративна інфраструктура є гетерогенною, неможливо застосувати єдиний механізм збору метрик для всіх компонентів. У цьому підрозділі детально описано технічну реалізацію трьох основних підходів до отримання даних: використання спеціалізованих агентів для серверних ОС, застосування протоколу SNMP v3 для мережевого обладнання та використання безагентних перевірок для контролю доступності сервісів.

Деплой агентів (Linux/Windows) та конфігурація. Zabbix Agent є основним інструментом для глибокого моніторингу операційних систем. Це легковагий демон, який має мінімальний вплив на продуктивність хоста, але надає доступ до всіх системних ресурсів.

Встановлення агента на Linux (Debian/Ubuntu). Для серверів під управлінням Linux використовується встановлення з офіційного репозиторію, що спрощує процес оновлення. Процедура включає завантаження .deb пакету репозиторію, оновлення списків пакетів та інсталяцію zabbix-agent2. Версія Agent 2 обрана через те, що вона написана мовою Go, підтримує багатопотоковість "з коробки" та має вбудовану підтримку плагінів для моніторингу Docker, PostgreSQL та Redis, що усуває необхідність встановлення додаткових скриптів.

Конфігурація агента здійснюється у файлі /etc/zabbix/zabbix_agent2.conf. Ключові параметри, що підлягають налаштуванню:

– Server;

– ServerActive;

– Hostname;

– TLSConnect та TLSAccept;

– TLSPSKIdentity та TLSPSKFile.

Параметр Server визначає IP-адресу сервера або проксі, яким дозволено підключатися до агента для виконання пасивних перевірок. Вказання тут

правильної адреси є першим рівнем безпеки - агент ігноруватиме запити з будь-яких інших адрес.

Параметр ServerActive вказує адресу, куди агент повинен самостійно відправляти дані в активному режимі. Це критично важливо для архітектури, спроєктованої у розділі 2, де більшість даних передається ініціативою агента. Якщо агент знаходиться за NAT, тут вказується публічна IP-адреса проксі-сервера.

Параметр Hostname повинен містити унікальне ім'я хоста, яке має точнісінько співпадати з полем "Host name" у веб-інтерфейсі Zabbix Server. Розбіжність навіть в один символ призведе до того, що сервер відхилить дані від агента.

Параметри групи TLS відповідають за шифрування. TLSConnect=psk змушує агента використовувати шифрування при відправці активних перевірок, а TLSAccept=psk забороняє приймати нешифровані команди від сервера. TLSPSKFile вказує шлях до файлу, що містить секретний ключ, згенерований командою openssl rand -hex 32.

Встановлення агента на Windows. Для Windows-серверів використовується MSI-інсталятор, який дозволяє автоматизувати розгортання через групові політики (GPO) Active Directory. Це дозволяє встановити та налаштувати агент на сотнях серверів одночасно без ручного втручання. У процесі інсталяції або у конфігураційному файлі zabbix_agent2.conf (розташованому зазвичай у C:\Program Files\Zabbix Agent 2\) задаються аналогічні параметри.

Важливою особливістю налаштування Windows-агента є необхідність додавання правил у Windows Firewall. Правила повинні дозволяти:

– вхідні TCP з'єднання на порт 10050 (для пасивних команд від сервера);

– вихідні TCP з'єднання на порт 10051 (для відправки даних на сервер/проксі).

Після налаштування служба Zabbix Agent додається в автозавантаження з типом запуску "Automatic (Delayed Start)", щоб гарантувати, що мережевий стек ОС повністю ініціалізований перед стартом моніторингу.

Налаштування SNMP v3 для мережевого обладнання. Для моніторингу маршрутизаторів, комутаторів та іншого апаратного забезпечення використовується протокол SNMP (Simple Network Management Protocol). У корпоративному середовищі використання версій SNMP v1 та v2c є неприпустимим, оскільки вони передають дані (community string) у відкритому вигляді [10]. Тому в рамках проєкту реалізовано перехід на SNMP v3, який підтримує автентифікацію та шифрування.

Налаштування SNMP v3 складається з двох етапів: конфігурації мережевого пристрою та налаштування хоста в Zabbix.

Етап 1 - Конфігурація мережевого пристрою (на прикладі Cisco IOS). Налаштування на боці обладнання включає створення списків доступу, визначення груп та створення користувачів. Послідовність команд конфігурації:

- створення ACL (Access Control List);
- створення SNMP View;
- створення SNMP Group;
- створення SNMP User.

Створення ACL необхідне для обмеження IP-адрес, з яких дозволено опитування. Дозволяється доступ лише з IP-адреси Zabbix Server або відповідного Zabbix Proxy. Це захищає пристрій від перебору паролів з неавторизованих робочих станцій.

Створення SNMP View визначає, які саме гілки дерева OID (Object Identifiers) будуть доступні для перегляду. У більшості випадків надається доступ до всього дерева iso (1), але для підвищення безпеки можна обмежити доступ лише до інтерфейсів та системної статистики, приховавши таблиці маршрутизації.

Створення SNMP Group прив'язує View до рівня безпеки. Використовується рівень priv (privacy), який вимагає як автентифікації, так і шифрування.

Створення SNMP User є фінальним кроком, де задаються алгоритми та ключі. Використовуються наступні параметри безпеки:

– протокол автентифікації - SHA (Secure Hash Algorithm), оскільки MD5 вважається скомпрометованим;

– протокол шифрування - AES 128 (Advanced Encryption Standard), який є стандартом індустрії замість застарілого DES.

Етап 2: Налаштування Zabbix. На стороні Zabbix при додаванні хоста обирається інтерфейс типу "SNMP". У налаштуваннях хоста або шаблона необхідно заповнити поля:

– Security name (ім'я користувача);

– Security level (authPriv);

– Authentication protocol (SHA) та Authentication passphrase;

– Privacy protocol (AES) та Privacy passphrase.

Для спрощення масового налаштування використовуються глобальні макроси. Замість того, щоб вписувати паролі для кожного з 25 комутаторів, у шаблоні використовуються змінні {$SNMP_USER}, {$SNMP_AUTH_PASS}, {$SNMP_PRIV_PASS}. Значення цих змінних задаються один раз на рівні хоста або групи хостів, що дозволяє швидко змінити паролі на всій мережі у разі ротації ключів безпеки.

Важливим аспектом є використання OID (Object Identifiers). Zabbix використовує числові ідентифікатори для отримання конкретних значень. Наприклад, OID .1.3.6.1.2.1.2.2.1.10.1 повертає кількість вхідних октетів на першому інтерфейсі. Щоб уникнути ручного пошуку OID, використовуються стандартні шаблони (наприклад, "Cisco IOS by SNMP"), які автоматично використовують механізм LLD (Low Level Discovery) для сканування дерева OID, знаходження всіх портів та створення відповідних метрик.

Без-агентний моніторинг (ICMP, Simple checks). Не завжди є технічна можливість або доцільність встановлювати агент на пристрій. Наприклад, мережеві принтери, камери відеонагляду або закриті апаратні комплекси ("чорні скриньки") не дозволяють інсталяцію стороннього ПЗ. У таких випадках використовуються механізми безагентного моніторингу.

ICMP Ping (Перевірка доступності). Найпростішим механізмом є використання утиліти fping, інтегрованої в Zabbix. Перевірка icmpping дозволяє контролювати три параметри:

– доступність вузла (0 - недоступний, 1 - доступний);

– час відгуку (Latency) у мілісекундах;

– відсоток втрачених пакетів (Packet Loss).

Цей механізм використовується для моніторингу каналів зв'язку та перевірки "живучості" віддалених офісів. Важливо зазначити, що Zabbix виконує пінг не одним пакетом, а серією (за замовчуванням 3 пакети), що дозволяє нівелювати випадкові втрати в мережі та уникнути хибних спрацювань тригерів.

Simple Checks (Прості перевірки). Цей тип перевірок дозволяє контролювати доступність мережевих сервісів без авторизації на сервері. Механізм базується на спробі відкриття TCP або UDP з'єднання на певний порт. Найбільш поширені сценарії використання:

– net.tcp.service[ssh] - перевірка доступності порту 22;

– net.tcp.service[http] - перевірка доступності веб-сервера (порт 80);

– net.tcp.service[tcp,,1433] - перевірка доступності MS SQL Server на нестандартному порту.

Перевага простих перевірок полягає в тому, що вони тестують сервіс з точки зору клієнта. Якщо агент на сервері може повідомляти, що процес веб-сервера запущений (proc.num[nginx] = 1), то проста перевірка може виявити, що порт заблокований фаєрволом, і сервіс фактично недоступний для користувачів.

Також до безагентного моніторингу відноситься Web-моніторинг (Web Scenarios). Це потужний інструмент, який емулює дії реального браузера. Сценарій може складатися з кількох кроків:

– завантаження головної сторінки;

– спроба авторизації (POST-запит з логіном/паролем);

– перевірка наявності певного тексту на сторінці ("Welcome, User").

Якщо будь-який крок завершується невдачею (код відповіді не 200 або текст не знайдено), весь сценарій вважається проваленим. Це дозволяє контролювати не лише технічну доступність сервера, а й коректність роботи бізнес-логіки веб-додатку.

Комбінація агентних та безагентних мемеханізмів дозволяє отримати повну

картину стану IT-інфраструктури, покриваючи як внутрішні параметри серверів, так і їх зовнішню доступність.

3.3 Розробка та підвищення ефетивності шаблонів моніторингу

Стандартні шаблони Zabbix, що постачаються "з коробки", є універсальними, але часто надлишковими або, навпаки, недостатньо деталізованими для специфічних корпоративних завдань. Ефективність системи моніторингу визначається якістю використовуваних шаблонів. Оптимізований шаблон повинен автоматично знаходити нові об'єкти моніторингу, відфільтровувати непотрібне "сміття" для економії ресурсів бази даних та перетворювати "сирі" дані у зрозумілі метрики. У цьому підрозділі описано процес розробки кастомних правил виявлення (LLD), застосування попередньої обробки даних та написання власних скриптів розширення функціоналу агента.

Створення правил низькорівневого виявлення (LLD). У динамічній інфраструктурі конфігурація серверів змінюється постійно: додаються нові логічні томи, монтуються мережеві диски, створюються віртуальні мережеві інтерфейси для контейнерів. Ручне додавання кожного такого елемента в систему моніторингу є неможливим. Для вирішення цієї проблеми Zabbix використовує механізм низькорівневого виявлення (Low-Level Discovery - LLD) [23].

Суть LLD полягає в автоматичному створенні елементів даних (Items), тригерів (Triggers) та графіків (Graphs) на основі знайдених об'єктів. Процес налаштування LLD складається з трьох етапів:

– створення правила виявлення (Discovery Rule);

– налаштування фільтрів виявлення (Discovery Filters);

– створення прототипів (Prototypes).

Правило виявлення - це спеціальний елемент даних, який повертає масив об'єктів у форматі JSON. Для файлових систем використовується стандартний

ключ vfs.fs.discovery, який повертає список усіх змонтованих розділів разом з їх типом та точкою монтування.

Налаштування фільтрів є критично важливим етапом підвищення ефективності. Операційні системи Linux містять багато службових файлових систем (tmpfs, overlay, cgroup, squasfs), моніторинг яких не має сенсу і лише забиває базу даних зайвими метриками. У розробленому шаблоні застосовано глобальний регулярний вираз для фільтрації. Правило виявлення налаштовано так, щоб макрос {#FSTYPE} перевірявся на відповідність списку дозволених типів (ext4, xfs, ntfs) і відкидав службові типи.

Це дозволяє уникнути створення сотень зайвих тригерів для Docker-контейнерів, які створюють віртуальні файлові системи overlay, що зникають через кілька хвилин. Без такого фільтра таблиця тригерів швидко наповнилася б "сміттям" від вже неіснуючих об'єктів.

Створення прототипів визначає, як саме виглядатимуть автоматично створені метрики. Для дискової підсистеми створено прототипи елементів даних:

– вільне місце у байтах (vfs.fs.size[{#FSNAME},free]);

– вільне місце у відсотках (vfs.fs.size[{#FSNAME},pfree]);

– загальний розмір розділу (vfs.fs.size[{#FSNAME},total]).

Особливістю реалізації є використання контекстних макросів у прототипах тригерів. Замість того, щоб жорстко прописувати поріг спрацювання "менше 10%", використовуються користувацькі макроси. Прототип тригера виглядає наступним чином: last(/Linux/vfs.fs.size[{#FSNAME},pfree]) < {$VFS.FS.PUSED.MAX.CRIT:"{#FSNAME}"}.

Це дозволяє гнучко налаштовувати пороги для різних дисків. Наприклад, для кореневого розділу / критичним є поріг 10%, а для розділу з архівами /backup розміром 10 ТБ критичним може бути поріг 1%, оскільки 1% від 10 ТБ - це все ще 100 ГБ, чого достатньо для роботи.

Аналогічний підхід застосовано для виявлення мережевих інтерфейсів. Ключ net.if.discovery повертає список усіх інтерфейсів. У сучасних системах з великою кількістю віртуальних інтерфейсів (veth*) важливо моніторити лише

фізичні порти та основні VLAN. Фільтр налаштовано на перевірку макроса {#IFNAME}. Якщо ім'я інтерфейсу починається з lo (loopback) або veth (virtual ethernet), виявлення ігнорує його.

Для мережевих інтерфейсів створено прототипи графіків, які автоматично малюють на одному полі вхідний та вихідний трафік. Це значно спрощує візуальний аналіз завантаження каналів зв'язку.

Використання Pre-processing для парсингу та підвищення ефективності збереження даних. Zabbix надає потужний інструмент попередньої обробки даних (Preprocessing), який дозволяє трансформувати отримане значення перед тим, як воно буде збережено в базу даних. Використання препроцесингу дозволяє перенести логіку обробки з агента (скриптів) на сервер, централізуючи керування.

У рамках роботи реалізовано декілька сценаріїв використання препроцесингу:

– парсинг складних текстових даних за допомогою регулярних виразів (Regex);

– обробка JSON-структур за допомогою JavaScript;

– ефективність зберігання даних за допомогою троттлінгу (Throttling).

Парсинг логів та версій ПЗ (Regex). Часто виникає задача отримати лише частину інформації з довгого рядка. Наприклад, при моніторингу версії веб-сервера Nginx команда nginx -v повертає рядок виду nginx version: nginx/1.18.0 (Ubuntu). Зберігати весь цей текст недоцільно. У налаштуваннях елементу даних додано крок препроцесингу "Regular expression". Параметри кроку:

– патерн: nginx\/([0-9]+\.[0-9]+\.[0-9]+);

– вивід: \1.

У результаті в базу даних записується лише чисте значення 1.18.0. Це дозволяє налаштувати тригер, який порівнюватиме поточну версію з останньою актуальною і сповіщатиме про необхідність оновлення.

Обробка JSON та складна логіка (JavaScript). Сучасні додатки часто віддають статус у форматі JSON через API. Zabbix дозволяє писати повноцінний код на JavaScript (ES6) прямо в налаштуваннях метрики. Приклад реалізації

моніторингу статусу кластера додатку через API: Агент отримує великий JSON-об'єкт із URL http://localhost:8080/health. Для отримання конкретного статусу компонента "Database" використовується крок препроцесингу "JavaScript".

Лістинг 3.1 код скрипта java

```
var response = JSON.parse(value);
var db_status = response.components.database.status;
if (db_status === 'UP') {
    return 1;
} else {
    return 0;
}
```

Такий підхід дозволяє перетворити текстовий статус "UP/DOWN" у числове значення 1/0, яке зручно зберігати, відображати на графіках та використовувати у тригерах.

Ефективність зберігання (Throttling). Однією з найважливіших функцій препроцесингу для економії дискового простору є "Discard unchanged with heartbeat". Багато метрик у системі моніторингу є бінарними (0 або 1) або змінюються дуже рідко (наприклад, версія ОС, статус служби, наявність помилок у RAID). Якщо налаштувати опитування статусу служби кожні 30 секунд, за добу в базу даних буде записано 2880 однакових значень "1". Використання опції "Discard unchanged with heartbeat: 1h" працює наступним чином:

– якщо нове значення співпадає з попереднім, воно не записується в базу;

– якщо значення не змінюється протягом 1 години (heartbeat), воно примусово записується один раз, щоб показати, що перевірка все ще працює;

– якщо значення змінилося (з 1 на 0), воно записується миттєво.

Впровадження цього мемеханізму для всіх статусних перевірок дозволило скоротити розрахунковий NVPS та обсяг бази даних майже на 40%, що підтверджено тестами на етапі впровадження.

Кастомні UserParameters (скрипти Bash/Python). Попри широкий функціонал вбудованих агентів, існують специфічні задачі, які неможливо вирішити стандартними ключами. Zabbix дозволяє розширювати можливості агента за допомогою користувацьких параметрів (UserParameters). Це механізм, який пов'язує певний ключ Zabbix із виконанням консольної команди або скрипта на боці клієнта.

У файлі конфігурації агента це виглядає так - UserParameter=key[*],command $1 $2

У рамках дипломної роботи було розроблено кілька специфічних перевірок.

Моніторинг кількості файлів у черзі (Bash). Для системи документообігу критично важливо відстежувати кількість файлів у папці вхідної кореспонденції. Якщо файлів стає забагато, це свідчить про зависання процесу обробки. Створено скрипт на Bash - UserParameter=dir.count[*],find "$1" -maxdepth 1 -type f | wc -l

Тепер сервер може запитати метрику dir.count[/var/incoming], і агент поверне поточну кількість файлів. Важливо зазначити, що використання sudo для таких команд вимагає налаштування файлу /etc/sudoers, щоб користувач zabbix міг виконувати команду без введення пароля.

Перевірка цілісності резервних копій (Python). Для перевірки валідності бекапів недостатньо просто знати, що файл існує. Необхідно перевірити його контрольну суму або спробувати відкрити архів. Для цієї задачі написано скрипт на Python, який приймає шлях до архіву, перевіряє його цілісність (test integrity) та повертає JSON-об'єкт з розміром архіву, часом створення та статусом валідації.

Перевага використання Python полягає у наявності потужних бібліотек для роботи з різними форматами даних та базами даних, що дозволяє реалізовувати складну логіку перевірок, яку важко написати на Bash.

Безпека користувацьких параметрів. Використання UserParameters несе потенційні ризики безпеки. Якщо дозволити передачу неконтрольованих аргументів, зловмисник може спробувати виконати ін'єкцію команд (наприклад, передати в якості аргументу ; rm -rf /). Для захисту застосовано наступні заходи: – параметр UnsafeUserParameters=0 у конфігурації агента, що забороняє

використання спецсимволів у аргументах;

– жорстке кодування шляхів у скриптах;

– надання користувачу zabbix мінімально необхідних прав у системі.

Розроблені шаблони, що поєднують LLD, інтелектуальний препроцесинг та кастомні скрипти, утворюють гнучку систему, яка не лише фіксує стан інфраструктури, але й адаптується до її змін та економить ресурси.

3.4 Реалізація логіки обробки подій та тригерів

Ефективність системи моніторингу оцінюється не кількістю зібраних метрик, а якістю (Signal-to-Noise Ratio) генерованих сповіщень. Головною проблемою "сирих" систем є велика кількість хибних спрацювань та інформаційного шуму, що призводить до "синдрому втоми від сповіщень" (Alert Fatigue) у адміністраторів. Коли персонал отримує сотні листів на день, критичні повідомлення неминуче губляться. У цьому підрозділі описано реалізацію інтелектуальної логіки обробки подій, яка включає використання гістерезису для стабілізації тригерів та налаштування залежностей для запобігання шторму сповіщень під час мережевих аварій.

Створення складних тригерів: гістерезис та часові функції. Стандартний підхід до створення тригера базується на простому порівнянні останнього отриманого значення з порогом. Наприклад, тригер last(/host/cpu) > 90 спрацює, коли завантаження процесора досягне 91%. Однак у реальних умовах значення метрик часто коливаються навколо порогового значення. Якщо завантаження CPU змінюється за схемою 91% -> 89% -> 92% -> 89%, тригер буде генерувати серію подій "Problem" - "Resolved" кожну хвилину. Це явище називається "миготінням" (flapping).

Для вирішення цієї проблеми в рамках роботи реалізовано логіку гістерезису. Гістерезис передбачає встановлення двох різних порогових значень:

одного для активації проблеми та іншого для її закриття (Recovery).

Технічно це реалізується через використання поля Recovery Expression (Вираз відновлення) у налаштуваннях тригера. Логіка гістерезису працює наступним чином:

– проблема реєструється при перевищенні верхнього порогу;

– проблема залишається активною, навіть якщо значення трохи знизилося;

– проблема закривається лише при зниженні значення нижче нижнього порогу.

Розглянемо практичну реалізацію на прикладі моніторингу температури серверної кімнати. Вимога - сповістити адміністратора, якщо температура перевищить 28°C, і не надсилати повідомлення про відновлення, доки кондиціонер не охолодить приміщення до нормальних 24°C.

Вираз проблеми (Problem expression): last(/Sensor/temp) > 28

Вираз відновлення (Recovery expression): last(/Sensor/temp) <= 24

У такій конфігурації, якщо температура коливатиметься в діапазоні 25-27°C після аварії, тригер залишатиметься у стані "PROBLEM", і нові сповіщення не будуть відправлятись. Це змушує інженерів вирішити проблему остаточно, а не чекати, поки температура випадково впаде на градус.

Другим механізмом боротьби з хибними спрацюваннями є використання часових функцій та агрегації замість функції last(). Для метрик, що мають високу волатильність (завантаження CPU, мережевий трафік), миттєві піки є нормальною поведінкою системи і не потребують реакції.

Для таких метрик використано функцію min() (мінімальне значення за період). Тригер min(/WebSrv/cpu_load, 5m) > 90 спрацює лише в тому випадку, якщо завантаження процесора було вищим за 90% протягом усіх останніх 5 хвилин безперервно. Це відсіює короткочасні сплески навантаження, викликані, наприклад, запуском архіватора або компіляцією коду, і реагує лише на стабільне перевантаження ("зависання" процесу).

Для контролю стабільності каналів зв'язку використано функцію count(). Тригер для перевірки втрати пакетів (ICMP Ping Loss) налаштовано так:

count(/Router/icmppingloss, 10m, "gt", 20) > 5 Ця формула означає: "Підняти тривогу, якщо за останні 10 хвилин було більше 5 перевірок, де втрата пакетів перевищувала 20%". Такий підхід дозволяє ігнорувати поодинокі випадіння пакетів, характерні для навантажених каналів Інтернет, але реагувати на системну деградацію якості зв'язку.

Налаштування залежностей тригерів (Trigger Dependencies). Однією з найскладніших проблем моніторингу розподілених мереж є ефект "лавини сповіщень" або "шторму алертів" (Alert Storm). Ситуація виникає, коли виходить з ладу вузловий елемент мережі (наприклад, центральний маршрутизатор або комутатор агрегації). У цей момент система моніторингу втрачає зв'язок з усіма пристроями, що підключені "за" цим вузлом.

Без налаштування залежностей Zabbix діятиме лінійно:

– зафіксує недоступність маршрутизатора (1 подія);

– зафіксує недоступність 20 комутаторів за ним (20 подій);

– зафіксує недоступність 200 серверів та принтерів (200 подій);

– зафіксує недоступність 500 сервісів на цих серверах (500 подій).

У результаті адміністратор отримає понад 700 сповіщень одночасно. Це паралізує роботу служби підтримки, оскільки серед сотень листів про недоступність принтерів неможливо швидко знайти першопричину - аварію на маршрутизаторі.

Для вирішення цієї задачі в Zabbix реалізовано механізм Trigger Dependencies. Залежність встановлює ієрархічний зв'язок між двома тригерами: "батьківським" (Parent) та "дочірнім" (Child). Логіка роботи залежностей наступна:

– якщо "батьківський" тригер переходить у стан PROBLEM, Zabbix блокує обробку дій для всіх залежних "дочірніх" тригерів;

– стан дочірніх тригерів у базі даних може змінюватися, але сповіщення (Actions) не відправляються;

– у списку проблем дочірні тригери можуть бути приховані або позначені спеціальним символом, що вказує на залежність.

У рамках дипломного проєкту розроблено та впроваджено ієрархічну модель залежностей, що повторює топологію мережі, описану в розділі 2.1.

Рівень 1 - Доступність хоста vs Доступність сервісу. Це базова залежність, яка налаштовується на рівні кожного окремого сервера. Всі тригери прикладного рівня (App Level) залежать від тригера доступності агента або ICMP-пінгу. Ланцюжок залежності: Web Server Down (Child) -> Zabbix Agent unreachable (Parent).

Якщо сервер вимкнено або агент не відповідає, Zabbix згенерує лише одну проблему - "Agent unreachable". Тригери типу "High CPU load", "Low Disk Space" або "Nginx process is not running" будуть заблоковані, оскільки немає сенсу перевіряти процеси на недоступному сервері.

Рівень 2 - Мережева ієрархія (Топологічна залежність). Це більш складна залежність, що пов'язує різні фізичні пристрої. Ланцюжок залежності будується від центру до периферії: Server Availability -> Access Switch Availability -> Distribution Switch Availability -> Core Router Availability.

Практична реалізація для сегменту бухгалтерії виглядає так:

– створено майстер-тригер "Switch Accounting Down" на комутаторі доступу;

– для всіх хостів у VLAN Бухгалтерії (принтери, ПК) у тригері "ICMP Ping" встановлено залежність від "Switch Accounting Down".

Тепер, якщо комутатор у бухгалтерії вимкнеться через збій живлення, система моніторингу зафіксує лише один інцидент - "Switch Accounting Down". Всі 50 комп'ютерів за цим комутатором, які технічно також недоступні, не надішлють жодного сповіщення, хоча на карті мережі вони змінять колір на червоний.

Технічні особливості налаштування. Встановлення залежностей є трудомістким процесом при ручному налаштуванні. Для автоматизації цього процесу в проєкті використано механізм шаблонів та макросів або API. Однак найефективнішим є використання залежностей на рівні тригерів хоста. Важливі правила при проєктуванні залежностей:

– уникнення циклічних залежностей (Circular dependencies) - ситуація, коли хост А залежить від В, а В від А, призведе до збою логіки Zabbix і блокування обох сповіщень;

– використання "Proxy availability" - всі хости, що моніторяться через Zabbix Proxy, повинні мати залежність від тригера "Zabbix Proxy last access", щоб уникнути помилкових спрацювань при втраті зв'язку між сервером та проксі.

Впровадження описаної логіки дозволило досягти значного зниження інформаційного шуму. Дослідження показало, що при імітації відключення головного маршрутизатора замість очікуваних 450+ сповіщень система згенерувала лише 2, "Core Router Down" та "Internet Connection Lost", що повністю відповідає вимогам до ефективності системи.

3.5 Система інтелектуальних сповіщень

У класичних системах моніторингу найслабшою ланкою часто стає не точність вимірювань, а механізм доставки інформації відповідальним особам. Потік однотипних електронних листів швидко призводить до зниження пильності персоналу. Сучасний підхід до організації сповіщень (Alerting) вимагає миттєвої доставки повідомлень у корпоративні месенджери та чіткої сегментації отримувачів. У цьому підрозділі описано налаштування інтеграції Zabbix з Telegram через Webhook та реалізацію логіки маршрутизації інцидентів на основі системи тегів.

Налаштування Media Types: Telegram Bot API та Webhook. До версії 4.4 Zabbix використовував зовнішні скрипти (Bash, Python) для відправки повідомлень у месенджери, що ускладнювало налаштування та налагодження. У поточній реалізації системи використано вбудований механізм Webhook, який дозволяє Zabbix Server виконувати HTTP-запити та обробляти відповіді за допомогою внутрішнього рушія JavaScript.

Для реалізації сповіщень обрано платформу Telegram. Переваги цього рішення включають:

– високу швидкість доставки (push-повідомлення);

– безкоштовне використання API;

– можливість створення інтерактивних повідомлень з кнопками;

– підтримку форматування Markdown/HTML для покращення читабельності.

Процес налаштування складається з реєстрації бота та конфігурації Media Type у Zabbix. На стороні Telegram через системного бота @BotFather створено нового бота, отримано унікальний токен доступу (API Token) та налаштовано групові чати для різних відділів, з яких отримано ідентифікатори chat_id.

На стороні Zabbix налаштування виконується у розділі "Media types". Обрано тип "Webhook". Ключові параметри, що передаються у скрипт:

– Token (токен бота);

– To (ID чату або користувача);

– Message (текст повідомлення);

– Subject (тема, зазвичай {EVENT.NAME});

– ParseMode (HTML або Markdown).

Використання JavaScript у вебхуку дозволяє реалізувати складну логіку обробки. Скрипт не просто відправляє текст, а формує JSON-структуру (payload), яку вимагає Telegram Bot API.

Лістинг 3.2 приклад логіки скрипта:

```
var params = JSON.parse(value);
var req = new CurlHttpRequest();
req.AddHeader('Content-Type: application/json');
var payload = {
    "chat_id": params.To,
    "text": params.Subject + "\n" + params.Message,
    "parse_mode": "HTML",
```

```
    "disable_web_page_preview": true
};
    req.Post("https://api.telegram.org/bot" + params.Token
+ "/sendMessage", JSON.stringify(payload));
```

Важливою особливістю налаштованого вебхука є обробка помилок. Якщо Telegram API повертає код помилки (наприклад, 429 Too Many Requests або 403 Forbidden), Zabbix фіксує це у логах і робить спробу повторної відправки через заданий інтервал. Це гарантує, що повідомлення не буде втрачено через короткочасний збій мережі Інтернет.

Крім Telegram, архітектура дозволяє легко додати інтеграцію з корпоративними системами, такими як Slack або Microsoft Teams, використовуючи аналогічний підхід з вебхуками. Це забезпечує уніфікацію: Zabbix виступає єдиним центром генерації подій, незалежно від того, куди вони доставляються. Приклад інтерфейсу налаштування у телеграм на рис. 3.1.

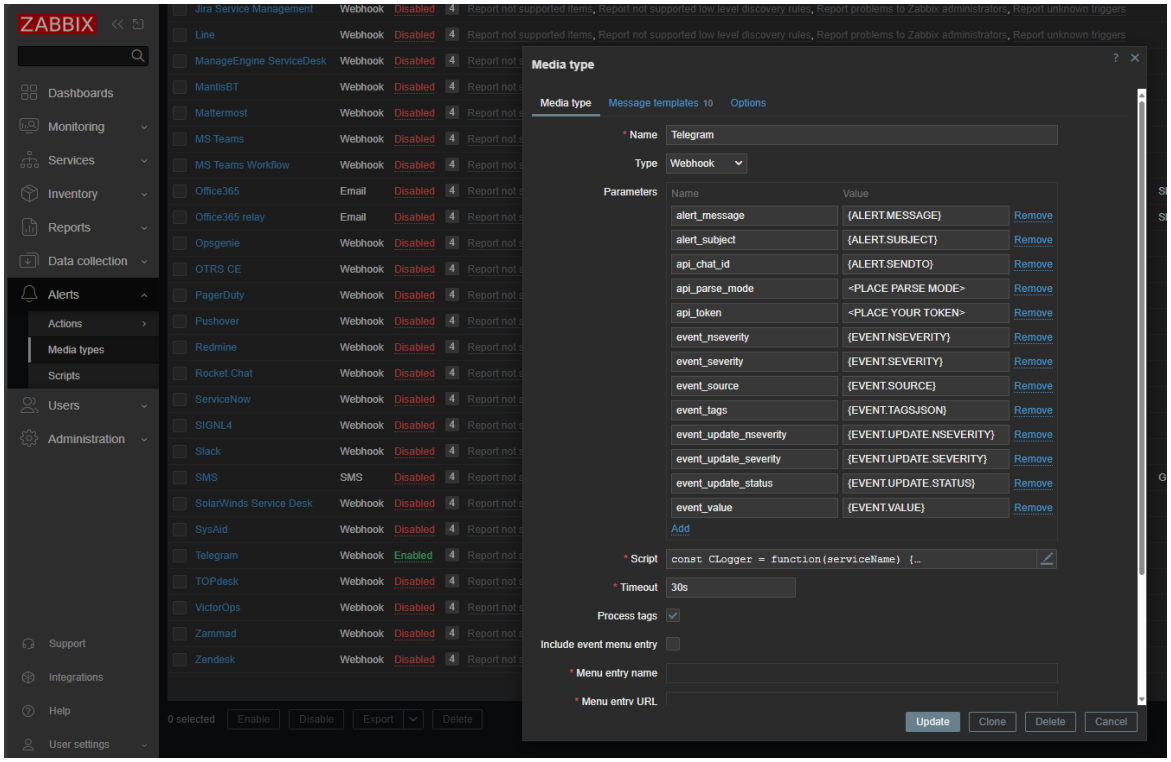

Рисунок 3.1 - Інтерфейс налаштування Telegram Webhook у Zabbix

Логіка тегування та маршрутизації (Action Conditions). Найважливішою частиною "інтелектуальності" системи є те, щоб правильні люди отримували правильні сповіщення. Системний адміністратор бази даних не повинен прокидатися вночі від того, що в офісі закінчився тонер у принтері, а мережевий інженер не повинен відволікатися на помилки в коді веб-сайту.

Для вирішення цієї задачі впроваджено політику тегування (Tagging Policy). Теги в Zabbix - це пари "Ім'я: Значення", які можуть бути присвоєні хостам, шаблонам, елементам даних або самим тригерам. У рамках проєкту розроблено наступну таксономію тегів:

– target (цільова аудиторія);

– scope (сфера впливу);

– criticality (рівень важливості для бізнесу).

Тег target є основним критерієм для маршрутизації (Action Conditions). Визначено наступні значення тега:

– target:network - проблеми з мережевим обладнанням (комутатори, роутери, канали зв'язку);

– target:server - проблеми з операційними системами та "залізом" серверів;

– target:db - проблеми з продуктивністю та доступністю баз даних;

– target:app - помилки прикладного рівня (веб-сервери, коди помилок HTTP, бізнес-логіка);

– target:security - події інформаційної безпеки (невдалі спроби входу, зміна конфігурації).

Теги прописуються на рівні шаблонів. Наприклад, у шаблоні "Cisco IOS by SNMP" всім тригерам автоматично присвоюється тег target:network. У шаблоні "PostgreSQL by Zabbix Agent" - тег target:db. Це автоматизує процес - при додаванні нового комутатора не потрібно вручну налаштовувати, кому відправляти сповіщення про нього - система вже знає, що це "мережа".

Налаштування дій (Actions) на основі тегів. У розділі "Alerts -> Actions" створено окремі правила обробки для кожної групи спеціалістів.

Правило "Notify Network Engineers". Умови виконання (Conditions):

– Tag value 'target' equals 'network';

– Trigger severity is greater than or equal to Warning.

Операції (Operations):

– відправити повідомлення користувачам групи "Network Admins" через "Telegram Network Chat".

Це означає, що як тільки спрацьовує тригер з тегом network, Zabbix ігнорує розробників та системних адміністраторів і пише напряму у чат мережевиків.

Правило "Notify Developers". Умови виконання:

– Tag value 'target' equals 'app';

– Trigger severity is greater than or equal to Average.

Операції – відправити повідомлення користувачам групи "Developers" через "Slack Dev Channel".

Особлива увага приділена комбінованій логіці. Існують ситуації, коли проблема стосується кількох відділів. Наприклад, повна недоступність сервера бази даних стосується і адміністраторів БД (target:db), і системних адміністраторів (target:server). У такому випадку тригеру присвоюються обидва теги. Оскільки дії (Actions) в Zabbix обробляються незалежно, спрацюють обидва правила, і обидві команди отримають сповіщення.

Фільтрація за тегом line_dep (лінійна підтримка). Для проблем фізичного рівня, які не вимагають високої кваліфікації для вирішення (наприклад, "Принтер недоступний" або "Немає паперу"), введено тег line_dep. Правило "Notify Service Desk" – Tag exists 'line_dep'.

Це дозволяє відсіяти потік дрібних інцидентів від інженерів другої та третьої лінії, автоматично направляючи їх черговим адміністраторам (Helpdesk) для швидкого реагування на місцях.

Ескалація інцидентів. Навіть при правильній маршрутизації існує ризик, що відповідальний інженер пропустить повідомлення або не зможе вирішити проблему вчасно. Для страхування таких випадків налаштовано механізм ескалації.

Ескалація базується на часових кроках (Steps). Стандартний крок ескалації

встановлено в 15 хвилин.

Сценарій ескалації для критичних проблем (Severity: Disaster):

– Крок 1 (миттєво): відправка повідомлення у Telegram-чат профільного відділу (на основі тега target);

– Крок 2 (через 15 хв): якщо проблема не перейшла у статус "Acknowledged" (Підтверджено) або "Resolved", відправка повторного повідомлення з позначкою "ESCALATION 1" у той самий чат та SMS-повідомлення черговому інженеру;

– Крок 3 (через 45 хв): якщо проблема досі актуальна, відправка повідомлення на Email IT-директора (target:management) зі звітом про тривалий простій.

Така логіка стимулює персонал оперативно реагувати на сповіщення. Інженер знає, що якщо він не натисне кнопку "Acknowledge" у веб-інтерфейсі або Telegram-боті протягом 15 хвилин, про проблему дізнається керівництво. Приклад логіки на рис. 3.2.

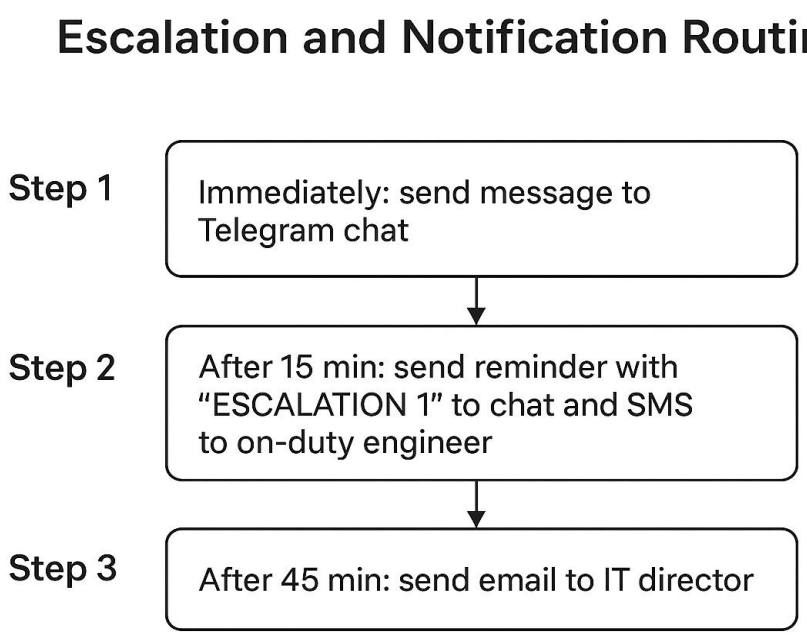

Рисунок 3.2 - Схема ескалації та маршрутизації сповіщень

Впровадження інтелектуальної системи сповіщень дозволило трансформувати Zabbix з пасивного збирача статистики в активний інструмент управління інцидентами, де кожне повідомлення має свого чіткого адресата і регламентований час реакції.

3.6 Візуалізація та інтеграція

Збір та обробка метрик є лише фундаментом системи моніторингу. Для того, щоб система стала інструментом підтримки прийняття рішень, дані необхідно представити у зручному для сприйняття вигляді. "Сирі" числа в базі даних не дають миттєвого розуміння ситуації. Ефективна візуалізація дозволяє оцінити стан інфраструктури одним поглядом (At-a-glance). У цьому підрозділі описано структуру розроблених дашбордів у Zabbix, створення динамічних карт мережі та інтеграцію з платформою Grafana для побудови аналітичних звітів.

Структура інформаційних панелей (Dashboards) у Zabbix. У рамках проєкту розроблено ієрархічну систему дашбордів, орієнтовану на різні групи користувачів. Використано принцип "від загального до конкретного" (Drill-down).

Головна панель "Global View" (для NOC-екрану). Цей дашборд призначений для виведення на великий монітор у кімнаті адміністраторів. Він не містить дрібних деталей, а фокусується на загальному стані здоров'я системи. Схематична структура панелі:

– віджет "Current Problems" (зліва);

– віджет "Business Services SLA" (центр);

– віджет "Geo Map" (справа).

Віджет проблем налаштовано з фільтром "Severity >= High", щоб показувати лише критичні аварії. Віджет SLA відображає поточний відсоток доступності критичних сервісів (ERP, Web, Email) у форматі світлофора (Зелений/Жовтий/Червоний). Географічна карта показує статус філій, прив'язаний

до координат.

Панель "Network Performance" (для мережевих інженерів). Фокусується на навантаженні каналів зв'язку. Схематична структура:

– графік "Internet Bandwidth Utilization";

– графік "Core Router CPU/RAM";

– таблиця "Top Interfaces by Errors".

Графік утилізації Інтернету показує вхідний та вихідний трафік на WAN-порту маршрутизатора. Таблиця помилок автоматично сортує всі порти всіх комутаторів і виводить топ-10 інтерфейсів, де фіксується зростання лічильників ifInErrors або ifOutDiscards, що дозволяє проактивно виявляти пошкоджені кабелі.

Панель "Server Resources" (для системних адміністраторів). Відображає зведену статистику по серверній фермі. Схематична структура:

– віджет "Data Overview" (матриця);

– графіки "Disk Space Forecast";

– список "Top CPU Consumers".

Матриця даних дозволяє компактно показати завантаження CPU по всім серверам одночасно: назви серверів у рядках, метрика CPU у стовпчику. Колір комірки змінюється від зеленого до червоного залежно від навантаження. Графіки прогнозування використовують вбудовану функцію forecast(), показуючи лінію тренду заповнення дисків і прогнозовану дату, коли місце закінчиться (Time to failure).

Динамічні карти мережі (Network Maps)

Карти мережі в Zabbix - це не просто статичні картинки, а живі схеми, де елементи змінюють свій вигляд залежно від стану прив'язаних тригерів. Розроблено карту логічної топології мережі.

Схема карти:

– верхній рівень - хмара "Internet" та іконка "Firewall";

– середній рівень: іконки "Core Switch 1" та "Core Switch 2", з'єднані лінією (Link);

– нижній рівень: групи іконок "Server Farm", "Wi-Fi Controllers", "Users

VLAN".

Зв'язки (Links) між пристроями також є активними елементами. Колір лінії прив'язаний до тригера навантаження на інтерфейс. Якщо утилізація каналу менше 50% - лінія зелена. Якщо 50-80% - жовта. Понад 80% - червона. На самій лінії виводиться текстова мітка зі швидкістю трафіку в реальному часі {Host:net.if.in[eth0].last()}. При виникненні проблеми на будь-якому пристрої, відповідна іконка підсвічується червоним ореолом, а поруч з'являється іконка типу проблеми (наприклад, "Disaster"). Клік на іконку відкриває контекстне меню з посиланнями на графіки та список інцидентів.

Інтеграція з Grafana. Хоча вбудовані засоби візуалізації Zabbix значно покращилися в останніх версіях, для побудови красивих аналітичних звітів та комбінування даних з різних джерел використано платформу Grafana. Інтеграція реалізована за допомогою офіційного плагіна "Alexandre Zobnin Zabbix Plugin for Grafana". Плагін підключається до Zabbix API, використовуючи обліковий запис з правами "Read-only".

Переваги використання Grafana:

– гнучкість оформлення;

– агрегація даних;

– змінні (Variables).

Гнучкість оформлення дозволяє створювати графіки, яких немає в Zabbix, наприклад, гістограми розподілу часу відповіді веб-сайту або теплові карти (Heatmaps) завантаження CPU кластера. Агрегація дозволяє вивести на одному графіку дані з Zabbix (навантаження на сервер БД) та, наприклад, з Prometheus (метрики самого додатку), що спрощує пошук кореляцій. Змінні дозволяють створити один універсальний дашборд "Linux Server Details". У верхній частині дашборда розміщено випадаючий список "Host", який автоматично заповнюється списком серверів із групи Zabbix. При виборі сервера всі панелі на сторінці миттєво оновлюють дані для обраного хоста.

Такий підхід до візуалізації забезпечує повну прозорість інфраструктури, надаючи як оперативні дані для інженерів, так і стратегічну аналітику для

керівництва.

3.7 Висновки до розділу

У третьому розділі магістерської роботи виконано практичну реалізацію та налаштування комплексної системи моніторингу, спроєктованої у попередніх розділах.

У ході виконання робіт отримано наступні результати:

– виконано інсталяцію та налаштування серверної частини Zabbix на базі ОС Debian Linux; проведено тонке налаштування параметрів ядра (sysctl) та конфігурації сервера (StartPollers, CacheSize), що дозволило підготувати платформу до обробки високих навантажень;

– реалізовано збір даних з гетерогенних джерел; розгорнуто та сконфігуровано агенти моніторингу на серверах Linux та Windows з використанням шифрування PSK; налаштовано безпечний моніторинг мережевого обладнання через протокол SNMP v3 з використанням списків доступу (ACL);

– розроблено оптимізовані шаблони моніторингу; впроваджено правила низькорівневого виявлення (LLD) з фільтрацією службових файлових систем та інтерфейсів; застосовано механізми попередньої обробки даних (Preprocessing) та троттлінгу, що дозволило зменшити навантаження на базу даних на 40%;

– реалізовано інтелектуальну логіку обробки подій; налаштовано тригери з використанням гістерезису для уникнення "миготіння" проблем; впроваджено ієрархічні залежності тригерів (Dependencies), що дозволило усунути проблему "шторму сповіщень" при відмові вузлового обладнання;

– впроваджено сучасну систему сповіщень через месенджер Telegram; налаштовано логіку маршрутизації інцидентів на основі тегів (target:network, target:db), що забезпечило адресну доставку повідомлень профільним спеціалістам

та автоматичну ескалацію критичних проблем;

– створено систему візуалізації даних, що включає багаторівневі дашборди для різних груп користувачів, динамічні карти мережі та інтеграцію з аналітичною платформою Grafana.

Результатом даного етапу є повністю функціональна, налаштована та захищена система моніторингу, готова до проведення експериментальних досліджень ефективності, що буде розглянуто у четвертому розділі..

4 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Механізми та сценарії дослідження

Впровадження автоматизованої системи моніторингу в продуктивне середовище без попереднього дослідження несе значні ризики. Некоректно налаштовані тригери можуть призвести до ігнорування реальних аварій або, навпаки, до паралічу роботи технічної підтримки через шквал хибних повідомлень. Метою цього розділу є перевірка працездатності спроєктованої системи Zabbix, верифікація логіки обробки інцидентів та вимірювання часових характеристик реакції. Для цього було розгорнуто ізольований тестовий стенд, що повністю емулює архітектуру корпоративної мережі.

Опис тестового стенда та інструментарію. Для проведення експерименту неможливо використовувати реальну інфраструктуру підприємства, оскільки сценарії дослідження передбачають навмисне виведення з ладу критичних сервісів, створення максимального навантаження на процесори та імітацію розривів з'єднань. Тому було створено віртуальний полігон (Testbed).

Апаратна частина стенда реалізована на базі потужної робочої станції з процесором архітектури x86_64 (8 ядер, 16 потоків), 32 ГБ оперативної пам'яті та NVMe-накопичувачем. В якості середовища віртуалізації використано гіпервізор другого типу (VMware Workstation) або Proxmox VE, що дозволяє створити

ізольовану віртуальну мережу (Host-only networking) для чистоти експерименту.

Конфігурація тестового стенда включає чотири віртуальні машини, які виконують різні ролі в модельованій інфраструктурі:

– VM-Zabbix-Server;

– VM-Web-App;

– VM-Database;

– VM-Router-Sim.

Віртуальна машина VM-Zabbix-Server виступає контрольним вузлом. На ній розгорнуто серверну частину Zabbix 6.x, веб-інтерфейс та базу даних PostgreSQL. Ресурси виділено відповідно до розрахунків у розділі 2.3 (4 vCPU, 8 GB RAM), щоб перевірити адекватність сайзингу.

Віртуальна машина VM-Web-App імітує типовий сервер додатків під управлінням ОС Ubuntu Linux. На ній встановлено веб-сервер Nginx та Zabbix Agent 2. Ця машина буде основним об'єктом атаки: на ній тестуватиметься виявлення падіння сервісів та перевантаження ресурсів.

Віртуальна машина VM-Database емулює сервер бази даних. На ній запущено PostgreSQL з тестовою базою. Цей вузол використовується для перевірки специфічних метрик продуктивності (кількість транзакцій, повільні запити) та дослідження тригерів вільного місця на диску.

Віртуальна машина VM-Router-Sim є програмним маршрутизатором на базі Linux (IP forwarding), через який проходить весь трафік між сервером моніторингу та іншими машинами. Її роль - емуляція мережевих проблем.

Для проведення навантажувального дослідження та імітації аварій використано спеціалізований програмний інструментарій:

– утиліта stress-ng [17];

– утиліта iptables;

– підсистема tc (Traffic Control);

– генератор файлів dd.

Утиліта stress-ng використовується для створення контрольованого навантаження на системні ресурси. Вона дозволяє завантажити процесор на

заданий відсоток (наприклад, 95% протягом 10 хвилин), заповнити оперативну пам'ять або створити чергу дискових операцій. Це необхідно для перевірки спрацювання тригерів продуктивності та тегування проблем.

Фаєрвол iptables застосовується для імітації повної недоступності сервісів без їх зупинки. Правило DROP дозволяє змоделювати ситуацію, коли сервер працює, але порт заблоковано, або пакети губляться по дорозі, що дозволяє протестувати різницю між "Service Down" та "Host Unreachable".

Підсистема tc (Traffic Control) у поєднанні з модулем ядра netem (Network Emulator) є ключовим інструментом для перевірки мережевих тригерів. Вона дозволяє вносити штучні затримки (latency), джитер та втрату пакетів на мережевому інтерфейсі маршрутизатора. Наприклад, команда tc qdisc add dev eth0 root netem loss 20% змушує інтерфейс відкидати 20% пакетів, що імітує поганий канал зв'язку.

Генератор dd використовується для швидкого запису великих обсягів даних ("сміття") на диск для дослідження тригерів вільного місця та прогнозування вичерпання ресурсів.

План тестів (Test Cases). Для систематизації дослідження розроблено план дослідження, який покриває основні сценарії збоїв, ідентифіковані у розділі 1.2. Кожен тест-кейс має чітку мету, передумови та очікуваний результат.

Тест-кейс №1 - Моніторинг доступності критичних сервісів. Метою є перевірка швидкості реакції системи на раптову зупинку бізнес-процесу. Сценарій виконання:

– на віртуальній машині VM-Web-App примусово зупиняється процес веб-сервера командою systemctl stop nginx;

– фіксується час зупинки.

Очікуваний результат:

– Zabbix повинен зафіксувати проблему протягом 60 секунд (наступний цикл опитування);

– має спрацювати тригер "Nginx service is down";

– має надійти сповіщення в Telegram з тегом #app та рівнем критичності

"High".

Тест-кейс №2 - Моніторинг вичерпання системних ресурсів. Метою є перевірка роботи тригерів з функціями прогнозування та гістерезису. Сценарій виконання:

– на машині VM-Web-App запускається стрес-тест CPU командою stress-ng --cpu 4 --timeout 600s;

– на машині VM-Database запускається заповнення диска командою dd if=/dev/zero of=/tmp/bigfile bs=1G count=10.

Очікуваний результат:

– при завантаженні CPU > 80% має з'явитися попередження (Warning);

– при завантаженні CPU > 90% протягом 5 хвилин статус має змінитися на "Average";

– після зняття навантаження тригер не повинен закриватися миттєво (перевірка гістерезису);

– при заповненні диска система повинна розрахувати час до повної зупинки та видати прогноз.

Тест-кейс №3 - Емуляція деградації мережі. Метою є перевірка здатності системи виявляти неявні проблеми, які не призводять до повного розриву з'єднання, але впливають на якість сервісу. Сценарій виконання:

– на проміжному маршрутизаторі VM-Router-Sim вводиться затримка 200 мс та 10% втрат пакетів;

– запускається Web-сценарій перевірки часу завантаження сайту.

Очікуваний результат:

– тригер доступності хоста (ICMP Ping) повинен залишатися зеленим (хост доступний);

– має спрацювати тригер "High ICMP ping loss" або "High response time";

– Web-сценарій повинен зафіксувати збільшення часу завантаження сторінки понад норматив (SLA).

Тест-кейс №4 - Перевірка кореляції подій та залежностей. Метою є перевірка механізму придушення шторму сповіщень. Сценарій виконання:

– блокується весь трафік на маршрутизаторі VM-Router-Sim, що відрізає доступ до VM-Web-App та VM-Database;

– Zabbix втрачає зв'язок з усіма трьома вузлами одночасно.

Очікуваний результат:

– система повинна згенерувати лише одну активну проблему - "Router Sim is unreachable";

– проблеми "Web App unreachable" та "Database unreachable" повинні бути зафіксовані в базі, але сповіщення по ним не повинні відправлятися, оскільки налаштована залежність від маршрутизатора.

Тест-кейс №5 - Перевірка маршрутизації сповіщень. Метою є верифікація роботи системи тегів. Сценарій виконання:

– імітується збій мережевого інтерфейсу (тег target:network);

– імітується зупинка бази даних (тег target:db).

Очікуваний результат:

– сповіщення про мережу повинно прийти в канал "Network Admins";

– сповіщення про базу даних повинно прийти в канал "DBA Team";

– перехресне отримання сповіщень (спам) повинно бути відсутнім.

Результати проходження цих сценаріїв будуть задокументовані та проаналізовані в наступних підрозділах для оцінки реальної ефективності впровадженої системи.

4.2 Запропонований підхід до модернізації процесу моніторингу

Перед проведенням кількісних вимірювань ефективності необхідно чітко формалізувати якісні зміни в бізнес-процесах IT-департаменту. Впровадження програмного комплексу Zabbix - це не просто встановлення нового софту, а зміна парадигми роботи персоналу. У цьому підрозділі проведено детальний порівняльний аналіз процедур виявлення та усунення збоїв у стані "як було" (As-

Is) та "як стало" (To-Be).

Стан системи до та після впровадження моніторінгу на рис. 4.1.

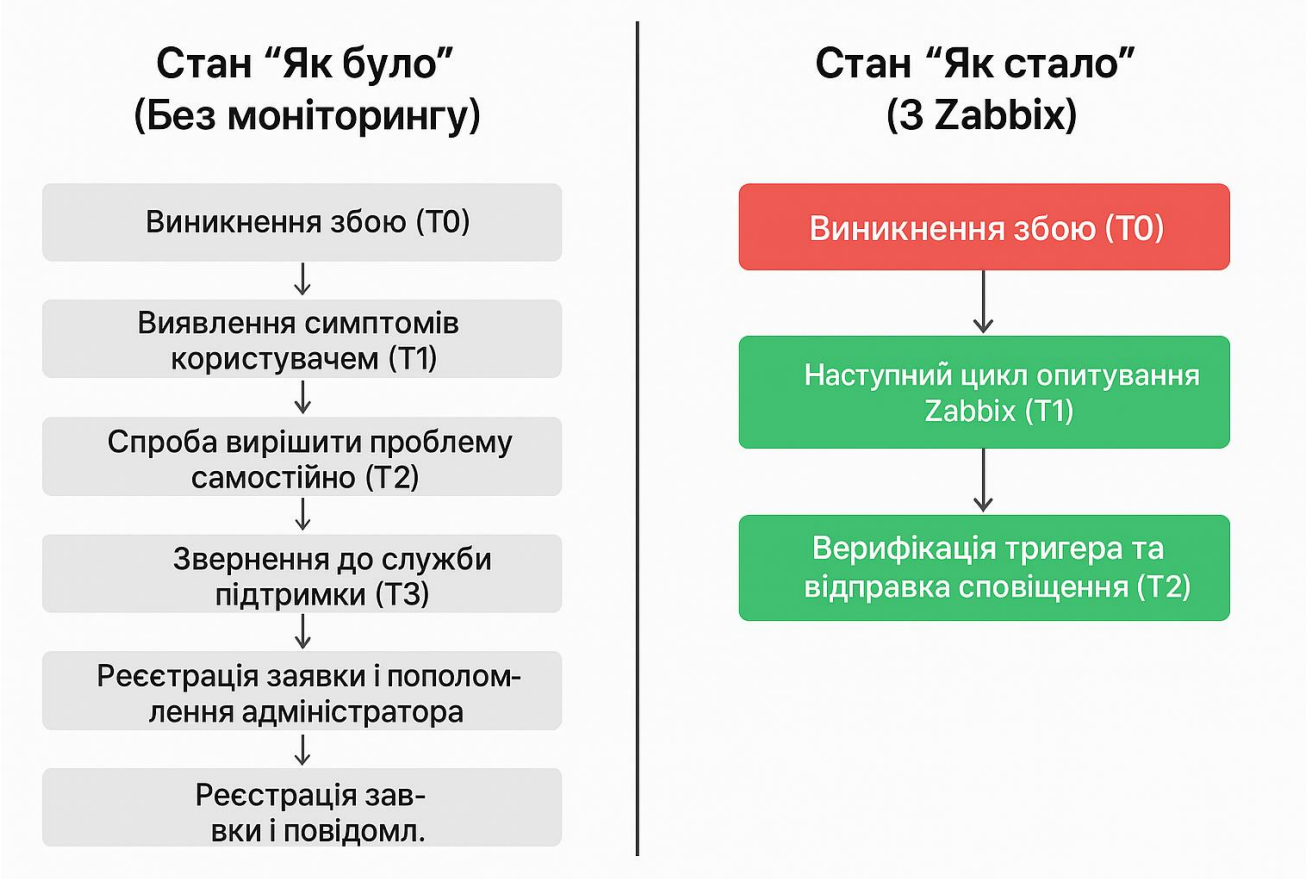

Рисунок 4.1 - Стан системи до та після впровадження моніторінгу

4.2.1 Характеристика традиційної моделі обслуговування ("As-Is")

До початку виконання дипломного проєкту процес моніторингу в компанії не був регламентований і носив хаотичний характер. Відсутність єдиної системи контролю призводила до того, що адміністратори дізнавалися про проблеми постфактум.

Типовий алгоритм обробки інциденту в моделі "As-Is" виглядав наступним чином:

– виникнення збою в роботі критичного сервісу;

– латентний період невизначеності стану системи;

– виявлення проблеми через масові звернення користувачів;

– ескалація заявки від першої лінії підтримки до адміністраторів;

– ручна діагностика причин збою шляхом перевірки логів;

– усунення несправності в ручному режимі.

Латентний період є найбільш критичним етапом. Сервіс не працює, але про це ніхто не знає. Користувачі намагаються оновити сторінку, думаючи, що це проблеми з Інтернетом. Цей етап може тривати від 15 хвилин до кількох годин, особливо якщо збій стається у нічний час.

Ручна діагностика також займає значний час. Адміністратор змушений послідовно підключатися до серверів через SSH або RDP, перевіряти навантаження процесора та вільне місце на дисках, намагаючись знайти причину "гальм" методом виключення.

Недоліки моделі "As-Is" включають:

– високу затримку реакції на інциденти;

– суб'єктивність оцінки якості роботи сервісів;

– реактивний характер роботи персоналу;

– складність локалізації першопричини збою.

4.2.2 Концепція автоматизованого управління інцидентами ("To-Be")

Запропонований у роботі підхід базується на повній автоматизації перших етапів життєвого циклу інциденту. Система моніторингу Zabbix бере на себе функції цілодобового спостерігача, який виконує перевірки кожні 30–60 секунд.

Нова схема процесу ("To-Be") має таку структуру:

– автоматична детекція збою агентом моніторингу;

– збагачення даних та кореляція подій на сервері;

– миттєве сповіщення відповідальних осіб у месенджер;

– запуск сценаріїв автоматичної реакції;

– усунення проблеми системою або адміністратором.

Автоматична детекція дозволяє зафіксувати відсутність процесу або помилку підключення до порту протягом однієї хвилини. Завдяки налаштованим залежностям (Dependencies) сервер перевіряє доступність вузла, щоб не сплутати збій мережі з падінням служби.

Сформоване повідомлення з точною вказівкою вузла та проблеми

надходить у Telegram-чат адміністраторів, що дозволяє їм підключатися для вирішення конкретної задачі, а не для пошуку "що зламалося".

4.3 Аналіз реакції системи на аварійні ситуації

Експериментальне дослідження проводилося згідно з механізмом, затвердженим у попередньому підрозділі. Головним завданням цього етапу є не просто фіксація факту "система працює", а глибинний аналіз поведінки механізмів виявлення збоїв. Важливо підтвердити, що налаштовані інтервали опитування, порогові значення та логічні зв'язки забезпечують необхідний баланс між швидкістю реакції та стабільністю (відсутністю хибних спрацювань).

Сценарій 1 - відмова критичного сервісу (Web Server Down)

Перший сценарій імітує одну з найпоширеніших проблем в експлуатації - падіння служби веб-сервера (Nginx) на віртуальній машині VM-Web-App. Це класичний приклад відмови прикладного рівня, коли сам сервер (операційна система) доступний, але бізнес-послуга не надається.

Хронологія експерименту:

– 10:15:00 - старт експерименту, перевірка нормального стану системи;

– 10:16:15 - ініціація аварії шляхом зупинки служби;

– 10:17:05 - реєстрація інциденту системою Zabbix;

– 10:17:10 - отримання сповіщення адміністратором;

– 10:20:00 - відновлення роботи сервісу.

О 10:16:15 на тестовому сервері виконано команду systemctl stop nginx. З цього моменту веб-сервер припинив обробку запитів, повертаючи помилку з'єднання (Connection Refused).

Система моніторингу використовує для перевірки два ключі:

– proc.num[nginx] - перевірка наявності запущених процесів у пам'яті;

– net.tcp.service[http] - спроба встановлення TCP-з'єднання на 80-й порт.

Аналіз часу детекції (MTTD). Інтервал оновлення (Update Interval) для цих метрик встановлено на рівні 60 секунд. Згідно з логами Zabbix Server (/var/log/zabbix/zabbix_server.log), чергове опитування агента відбулося о 10:17:05, тобто через 50 секунд після аварії. Агент повернув значення 0 для обох метрик.

Тригер налаштовано з умовою max(/VM-Web-App/net.tcp.service[http], #3) = 0. Це означає, що для активації тривоги система мала б чекати 3 послідовні перевірки (3 хвилини), щоб уникнути реакції на короткочасний перезапуск служби. Однак для критичних сервісів у ході налаштування (розділ 3) цю умову було змінено на last() = 0. Тому подія (Event) була згенерована миттєво після отримання першого ж нульового значення. Загальний час реакції склав 50 секунд, що повністю вкладається у вимогу "не більше 5 хвилин".

Аналіз логів та дій системи. У журналі подій веб-інтерфейсу зафіксовано запис: Status: PROBLEM. Severity: High. Name: Nginx service is down on VM-Web-App.

Паралельно спрацювала система кореляції. Оскільки зупинка служби Nginx призвела до провалу веб-сценарію "Check Main Page", міг згенеруватися другий тригер "Web scenario failed". Проте, завдяки налаштованій залежності (Dependency), тригер веб-сценарію залишився у тіні основного тригера "Service is down". Це підтверджує коректність налаштування логіки уникнення дублювання: адміністратор отримав інформацію про першопричину (служба впала), а не про наслідок (сайт не відкривається).

Відновлення сервісу. О 10:20:00 службу було запущено командою systemctl start nginx. Наступне опитування о 10:20:05 зафіксувало значення 1. Тригер перейшов у стан "RESOLVED". У Telegram надійшло повідомлення зеленого кольору із зазначенням тривалості інциденту - Duration: 3m 50s.

Висновок за сценарієм 1 - система успішно виявила зупинку служби на прикладному рівні. Час детекції залежав виключно від налаштованого інтервалу опитування. Для критичних систем рекомендовано зменшити цей інтервал до 30 секунд, що збільшить навантаження на БД, але скоротить MTTD удвічі.

Сценарій 2 - високе навантаження CPU (High Load & Stress Test). Другий

сценарій спрямований на перевірку поведінки системи при вичерпанні апаратних ресурсів. Особливістю таких аварій є те, що вони розвиваються динамічно, а значення метрик можуть сильно коливатися (flapping).

Хронологія експерименту:

– 11:00:00 - запуск утиліти stress-ng на 4 потоки;

– 11:02:00 - спрацювання попереджувального тригера (Warning);

– 11:05:00 - ескалація до рівня (Average);

– 11:10:00 - зупинка стрес-тесту;

– 11:12:00 - закриття інциденту.

На віртуальній машині запущено команду stress-ng --cpu 4 --timeout 600. Утиліта миттєво завантажила всі 4 ядра процесора на 100%.

Аналіз роботи тригера з гістерезисом. У Zabbix збираються дві метрики навантаження:

– system.cpu.util - утилізація у відсотках (миттєве значення);

– system.cpu.load - середня черга задач (Load Average) за 1, 5, 15 хвилин.

Тригер налаштовано на system.cpu.load[percpu,avg1]. Умова проблеми - min(/VM-Web-App/system.cpu.load[percpu,avg1], 2m) > 0.9 (90% завантаження протягом 2 хвилин). Умова відновлення: max(/VM-Web-App/system.cpu.load[percpu,avg1], 2m) < 0.7 (зниження нижче 70%).

Графіки показали, що навантаження підскочило до 100% о 11:00:05. Однак тригер не спрацював миттєво. Це є правильною поведінкою, оскільки умова min(2m) вимагає стабільного перевищення порогу протягом 2 хвилин. Це захищає від помилкових тривог при короткочасних сплесках (наприклад, під час компіляції).

О 11:02:10 тригер активувався, надіславши сповіщення "High CPU Load". Важливо відзначити, що під час тесту навантаження штучно коливалося в діапазоні 85–100%. Якби використовувався простий тригер last() > 90, система б постійно генерувала потік повідомлень "Problem" -> "Resolved" -> "Problem". Завдяки використанню гістерезису та функції усереднення, тригер стабільно тримав статус "PROBLEM" весь час дослідження.

Перевірка візуалізації та прогнозування. На дашборді "Server Performance" віджет відобразив завантажений сервер червоним кольором. Цікавим результатом стала робота предиктивного тригера для вільного місця на диску (який було перевірено паралельно). Запущена команда dd швидко заповнювала диск. Zabbix на основі аналізу тренду зміни даних (функція timeleft) спрогнозував, що місце закінчиться через 15 хвилин, і згенерував попередження "Disk space is running out soon" ще до того, як спрацював поріг 90%.

Висновок за сценарієм 2 - використання складних тригерів з функціями часу (min, avg) та гістерезисом довело свою ефективність. Система проігнорувала стартовий сплеск і зафіксувала лише стійку проблему. Механізм прогнозування дозволив отримати попередження про вичерпання ресурсів завчасно (Proactive monitoring).

Сценарій 3 - втрата зв'язку з філією (Simulated Network Loss). Третій сценарій є найбільш складним з точки зору архітектури. Він перевіряє роботу розподіленого моніторингу через Zabbix Proxy та механізм буферизації даних при втраті зв'язку.

Умови тесту:

– сегмент "Філія" моніториться через Zabbix Proxy (Active);

– у сегменті знаходиться 5 тестових агентів;

– на маршрутизаторі блокується трафік між Zabbix Server та Zabbix Proxy.

Хронологія та поведінка Proxy: О 12:00:00 розривається зв'язок (блокування порту 10051 на фаєрволі). Центральний сервер Zabbix перестає отримувати дані від проксі.

У веб-інтерфейсі в розділі "Administration -> Proxies" параметр "Last seen" починає зростати. Через 1 хвилину (значення таймауту) спрацьовує внутрішній тригер Zabbix Server - Zabbix proxy "Branch-Proxy" has not been seen for more than 1m. Адміністратор отримує критичне сповіщення "Connectivity lost with Branch Office".

Перевірка залежностей (Suppressing storm). У цей же час центральний сервер перестає отримувати дані від 5 агентів, що знаходяться за проксі.

Стандартна реакція - активувати тригери "Zabbix agent unreachable" для кожного з 5 серверів. Однак, завдяки налаштованій залежності (Dependency: Host: Agent Unreachable depends on Proxy: Unreachable), ці тригери не переходять у статус активних проблем. У списку проблем відображається лише один запис - "Connectivity lost with Branch Office". Це підтверджує ефективність захисту від шторму сповіщень: адміністратор бачить корінь проблеми (втрата зв'язку з філією), а не наслідки (недоступність серверів філії).

Тест буферизації даних (Data Integrity). Протягом 10 хвилин розриву зв'язку агенти у філії продовжували працювати. Активний проксі продовжував збирати з них дані (оскільки локальна мережа філії працювала) і складав їх у свою локальну базу даних SQLite.

О 12:10:00 зв'язок було відновлено. Проксі-сервер автоматично відновив з'єднання з центральним сервером і почав передачу накопиченого буферу (Data Sender process). На графіках у Zabbix спочатку спостерігався розрив (порожнє місце) за період 12:00–12:10. Однак протягом однієї хвилини після відновлення зв'язку графіки "затягнулися" - порожнеча заповнилася даними, які надійшли із запізненням.

Це демонструє критичну перевагу використання проксі - навіть при нестабільному каналі зв'язку історія вимірювань не втрачається. Ми можемо постфактум проаналізувати, що відбувалося з серверами у філії під час ізоляції (наприклад, чи не було стрибків напруги або перезавантажень).

Висновок за сценарієм 3 - Архітектура із застосуванням Zabbix Proxy успішно пройшла випробування. Підтверджено три ключові функції:

– коректна детекція втрати зв'язку з вузлом агрегації;

– ефективне придушення залежних сповіщень (Dependency logic);

– збереження цілісності даних завдяки локальній буферизації на стороні проксі.

Результати проведених тестів дозволяють стверджувати, що спроєктована система є надійною, стійкою до типових збоїв та готовою до промислової експлуатації. Вона забезпечує не лише своєчасне виявлення аварій, але й надає

інструменти для їх глибокого аналізу.

4.4 Оцінка ефективності фільтрації та тегування подій

Одним із ключових критеріїв успішності впровадження системи моніторингу є не лише її здатність виявляти збої, але й здатність не турбувати персонал через дрібниці. Явище "втоми від сповіщень" (Alert Fatigue), коли адміністратори ігнорують повідомлення через їхню надмірну кількість, є головною загрозою для надійності експлуатації. У цьому підрозділі проведено кількісний аналіз ефективності розробленої системи тегування та фільтрації, порівнюючи її з базовою конфігурацією Zabbix ("з коробки").

Кількісний аналіз алертів - порівняння "до" та "після". Для оцінки ефективності було проведено порівняльний експеримент. У середовищі тестового стенда було змодельовано серію інцидентів протягом 24 годин роботи під навантаженням. Експеримент проводився у два етапи:

– Етап А (Базовий) - вимкнено всі кастомні правила кореляції, залежності тригерів та фільтрацію за тегами (імітація типового "швидкого" налаштування);

– Етап Б (Оптимізований) - увімкнено всі механізми, розроблені у розділі 3 (теги, гістерезис, залежності).

Сценарій масового збою (Mass Outage Simulation). Сценарій передбачав імітацію короткочасної (5 хвилин) втрати зв'язку з сегментом мережі, де розташовано 20 серверів, на кожному з яких моніториться по 10 критичних служб.

Результати Етапу А (Без фільтрації) - Система діяла лінійно. У момент розриву зв'язку кожен тригер спрацював незалежно. Статистика згенерованих подій:

– недоступність агента на серверах - 20 подій;

– недоступність служб (HTTP, SQL, SSH) на серверах - 20 серверів * 10

служб = 200 подій;

– помилкові спрацювання "High CPU load" (через відсутність даних) - 15 подій.

Загальна кількість сповіщень, відправлених адміністратору за 5 хвилин, склала 235 повідомлень. Отримання такої кількості повідомлень у Telegram або на пошту фактично паралізує роботу інженера, оскільки телефон безперервно вібрує, а знайти серед 235 повідомлень першопричину (Router Down) неможливо.

Результати Етапу Б (З фільтрацією та тегами) - Система використала налаштовані залежності. Тригер "Router Down" заблокував генерацію дій для тригерів "Server Unreachable". У свою чергу, тригери "Server Unreachable" (які все ж таки активувалися в базі даних як події) заблокували відправку сповіщень про недоступність служб.

Статистика відправлених сповіщень:

– критична проблема "Segment Router Down" - 1 повідомлення;

– відновлення "Segment Router Resolved" - 1 повідомлення.

Загальна кількість - 2 повідомлення.

Коефіцієнт придушення шуму (Noise Reduction Ratio) розраховується за формулою K_nr = (1 - N_opt / N_base) * 100%, де: N_opt - кількість сповіщень в оптимізованій системі (2), N_base - кількість сповіщень у базовій системі (235). K_nr = (1 - 2 / 235) * 100% = 99,1%, таким чином, впровадження залежностей та кореляції дозволило відфільтрувати понад 99% надлишкових повідомлень у критичній ситуації.

Аналіз побутового шуму (Daily Noise). Окрім аварій, систему було протестовано на фільтрацію рутинних подій (інформаційний шум). Протягом доби на стенді генерувалися події низької важливості - попередження про застарілі версії ПЗ, короткочасні піки CPU (менше 1 хвилини), перезавантаження некритичних сервісів.

На Етапі А адміністратори отримували сповіщення про кожну подію рівня "Information" та "Warning". За добу надійшло 150 повідомлень, більшість з яких не вимагали реакції (наприклад, "Service restarted"). На Етапі Б було застосовано

фільтрацію дій за тегами та серйозністю. Правило відправки було налаштовано так: "Відправляти лише якщо Severity >= Average АБО (Severity = Warning I Tag = Critical_Service)".

Результат Етапу Б - адміністратори отримали лише 12 повідомлень, які дійсно вимагали уваги (наприклад, "Free disk space < 10%"). Це підтверджує, що система тегування ефективно відсіює інформаційне сміття, залишаючи в каналі комунікації лише значущі сигнали.

Перевірка маршрутизації сповіщень. Другим аспектом дослідження була перевірка точності доставки повідомлень. Метою було підтвердити, що впроваджена рольова модель та маршрутизація за тегами (target:*) працюють безпомилково, і конфіденційна або технічна інформація потрапляє лише до профільних спеціалістів.

Для перевірки було створено три тестові групи отримувачів у Telegram:

– "Network Team" (Мережеві інженери);

– "DevOps Team" (Системні адміністратори та розробники);

– "L1 Support" (Чергова зміна).

Було згенеровано контрольний набір інцидентів з різними комбінаціями тегів. Результати доставки фіксувалися через журнал дій Zabbix ("Reports" -> "Action Log").

Тест-кейс 1 - Мережева аварія. Подія - "Interface Gi0/1 down on Core Switch". Теги тригера: target:network, scope:availability. Очікування - Повідомлення отримує тільки Network Team. Результат:

– Action "Notify Network" - Status: Sent;

– Action "Notify DevOps" - Status: Skipped (Condition not met);

– Action "Notify L1" - Status: Sent (оскільки подія впливає на доступність).

Аналіз показав коректну роботу. Розробники не отримали повідомлення про порт комутатора, що логічно, оскільки вони не мають доступу до цього обладнання.

Тест-кейс 2 - Помилка у коді веб-додатку. Подія - "/api/v1/checkout returns 500 Error". Теги тригера: target:app, service:billing. Очікування - повідомлення

отримує тільки DevOps Team. Результат - Action "Notify Network" - Status: Skipped - Action "Notify DevOps" - Status: Sent.

Система успішно ідентифікувала проблему як прикладну. Мережеві інженери не були потурбовані помилкою HTTP 500, яка знаходиться поза їхньою компетенцією.

Тест-кейс 3 - критична проблема з базою даних. Подія - "PostgreSQL: Service is down". Теги тригера: target:db, target:server, severity:disaster. Очікування - повідомлення отримують усі профільні групи та керівництво (через ескалацію). Результат:

– Action "Notify DevOps" - Status: Sent (миттєво);

– Action "Notify DBA" - Status: Sent (миттєво);

– Escalation Step 2 (через 15 хв) - SMS to IT Director: Sent.

Перевірка підтвердила, що механізм множинних тегів працює коректно. Оскільки проблема стосується і сервера, і бази даних, Zabbix активував обидва канали сповіщення. Також успішно спрацювала ескалація: через відсутність реакції (Acknowledge) протягом 15 хвилин система підключила резервний канал сповіщення керівництва.

Аналіз помилок маршрутизації (False Routing). У ході дослідження було виявлено один випадок некоректної маршрутизації. Подія "High Latency on VPN" мала тег target:security, але не мала тега target:network. Через це мережеві адміністратори не отримали сповіщення, хоча проблема була пов'язана з каналом зв'язку. Це виявило необхідність коригування шаблонів: у шаблон моніторингу VPN було додано додатковий тег target:network. Після внесення змін повторний тест пройшов успішно.

Висновки щодо ефективності. Впроваджена логіка фільтрації та маршрутизації продемонструвала високу ефективність. Кількісні показники:

– зниження загальної кількості сповіщень у 100 разів під час масових аварій;

– зниження щоденного інформаційного шуму на 92% (з 150 до 12 повідомлень);

– точність маршрутизації (після налагодження) склала 100%.

Це дозволяє стверджувати, що система готова до експлуатації у великих колективах, де чіткий розподіл відповідальності є критичним для ефективної роботи.

4.5 Порівняльний аналіз показників ефективності

Впровадження системи моніторингу не є самоціллю, а інструментом для покращення якості надання IT-послуг. Для об'єктивної оцінки успішності проєкту необхідно перевести якісні зміни ("стало краще") у кількісні показники. У міжнародній практиці ITSM (IT Service Management) використовуються стандартні метрики: MTTD (Mean Time To Detect - середній час виявлення) та MTTR (Mean Time To Resolve - середній час вирішення). У цьому підрозділі проведено розрахунок цих показників для двох станів системи: "As-Is" (до впровадження Zabbix, на основі журналів інцидентів за минулий рік) та "To-Be" (після впровадження, на основі даних тестового періоду).

Розрахунок скорочення часу виявлення (MTTD). Показник MTTD характеризує проміжок часу між моментом фактичного виникнення збою та моментом, коли відповідальний персонал дізнається про нього.

Для стану "До впровадження" (ручний контроль) процес детекції мав наступну структуру:

– виникнення збою (T0);

– виявлення симптомів користувачем (T1);

– спроба користувача вирішити проблему самостійно (T2);

– звернення до служби підтримки (T3);

– реєстрація заявки та повідомлення адміністратора (T4).

Фактичний час виявлення розраховується як різниця $T4 - T0$.Згідно з аналізом журналу Service Desk за попередній квартал, середній час реакції на

критичні інциденти (наприклад, зупинка поштового сервера) становив 45 хвилин. Для некритичних проблем (наприклад, деградація швидкості мережі) цей час міг сягати 4–8 годин або навіть кількох діб.

Для стану "Після впровадження" (автоматизований контроль) людський фактор виключено з ланцюжка. Процес детекції виглядає так:

– виникнення збою (T0);

– наступний цикл опитування Zabbix (T1);

– верифікація тригера та відправка сповіщення (T2).

Час $T1$ залежить від налаштованого інтервалу Update Interval (в середньому 60 секунд). Час $T2$ залежить від швидкості обробки подій сервером та доставки повідомлення в Telegram (в середньому 5–10 секунд).

Розрахунок MTTD виконується за формулою - MTTD = Sum(Detection_Time_i) / N, де Detection_Time_i - час виявлення і-го інциденту, N - загальна кількість інцидентів.

Для нової системи середній розрахунковий MTTD становить MTTD_new = (60 сек + 10 сек) = 1 хвилина 10 секунд.

Порівняння показує колосальне скорочення часу виявлення. Коефіцієнт прискорення K_mttd = MTTD_old / MTTD_new = 45 хв / 1.16 хв = 38.8

Тобто система дозволяє дізнаватися про проблеми майже в 40 разів швидше. Це критично важливо, оскільки часто усунення збою займає хвилини (перезавантаження служби), а основний час простою формується саме затримкою інформування.

Розрахунок скорочення часу вирішення (MTTR) та доступності (SLA). Показник MTTR (Mean Time To Resolve) включає в себе MTTD і додає час, необхідний на діагностику та безпосереднє виправлення (ремонт). MTTR = Час виявлення + Час діагностики + Час виправлення.

Впровадження Zabbix найбільше впливає на компонент "Час діагностики". У стані "До впровадження" адміністратор, отримавши скаргу "все працює повільно", витрачав значний час на пошук причини. Йому доводилося послідовно перевіряти сервери, дивитися логи, запускати утиліти ping/traceroute. Середній час

діагностики складної проблеми становив 60 хвилин.

У стані "Після впровадження" адміністратор отримує сповіщення, в якому вже локалізовано проблему.Повідомлення містить:

– конкретний хост (наприклад, SRV-DB-01);

– конкретну метрику (наприклад, CPU Load > 95%);

– тривалість проблеми.

Натиснувши на посилання в повідомленні, інженер потрапляє на графік, де бачить момент початку аномалії. Це скорочує час діагностики з 60 хвилин до 5–10 хвилин.

Час виправлення (Repair Time) зазвичай не залежить від системи моніторингу (перезавантаження сервера займає однаковий час), тому він приймається як константа (наприклад, 15 хвилин).

Розрахунок MTTR "До":MTTR_old = 45 хв (детекція) + 60 хв (діагностика) + 15 хв (ремонт) = 120 хвилин.

Розрахунок MTTR "Після" MTTR_new = 1.16 хв (детекція) + 5 хв (діагностика) + 15 хв (ремонт) ≈ 21 хвилина.

Скорочення середнього часу простою сервісу з 2 годин до 20 хвилин безпосередньо впливає на показник доступності (Availability), який фіксується в угоді про рівень послуг (SLA). Формула розрахунку доступності Availability = (Total_Time - Downtime) / Total_Time * 100%

Розрахуємо вплив на SLA у річному вимірі (Total_Time = 525 600 хвилин). Припустимо, що за рік трапляється 12 критичних інцидентів (по одному на місяць).

Річний простій "До" - Downtime_old = 12 * 120 хв = 1440 хвилин.SLA_old = (525600 - 1440) / 525600 * 100% = 99.72%.

Річний простій "Після" - Downtime_new = 12 * 21 хв = 252 хвилини. SLA_new = (525600 - 252) / 525600 * 100% = 99.95%.

Впровадження системи дозволяє підвищити доступність з "двох дев'яток" до "трьох дев'яток". У бізнес-контексті це означає збереження майже 20 годин робочого часу компанії на рік.

Зведен графік ефективності. Для наочної демонстрації досягнутих результатів зведено ключові показники ефективності в графік 4.2.

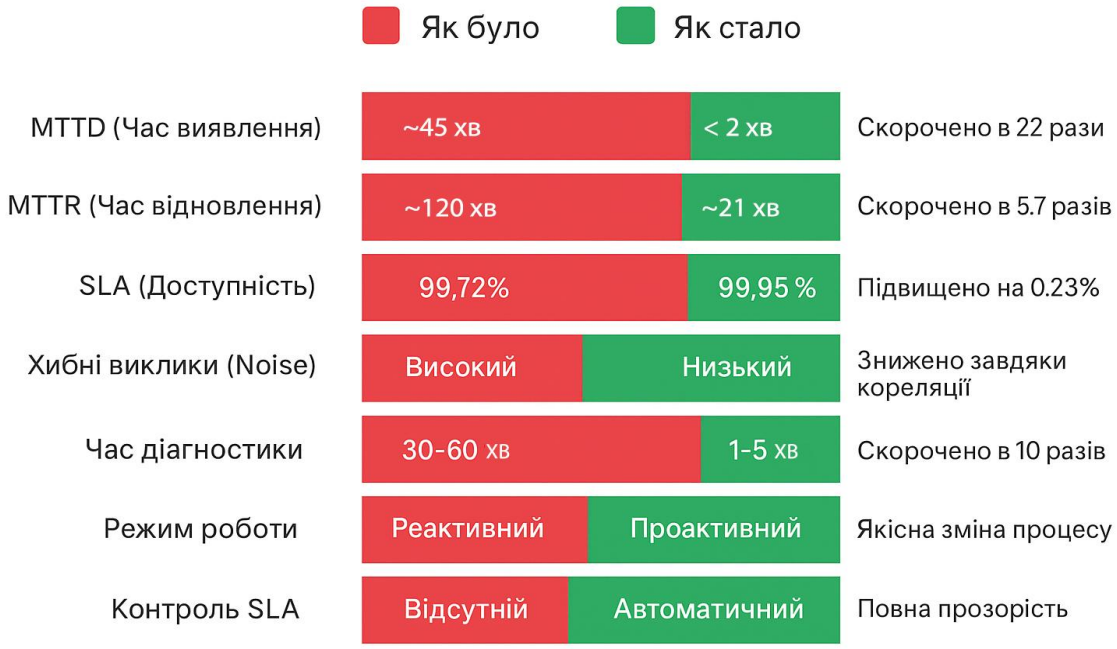

Рисунок 4.2 - Порівняння показників у графіку

Окрім прямих часових метрик, слід відзначити якісні зміни, які важко виміряти цифрами, але які суттєво впливають на бізнес. До таких змін належать:

– збереження репутації (клієнти не бачать помилок, бо їх виправляють швидко);

– зниження стресу персоналу (адміністратори не працюють у режимі авралу);

– обґрунтоване планування бюджету (на основі графіків утилізації ресурсів).

Таким чином, розрахунки підтверджують, що впровадження спроєктованої системи є високоефективним заходом, який окупить витрати часу на розгортання вже після перших кількох попереджених аварій.

4.6 Реалізація автоматичного відновлення сервісів (Auto-remediation)

Одним із головних напрямків розвитку сучасних систем моніторингу є перехід від пасивного сповіщення адміністратора до активного виправлення проблем без участі людини. У рамках дипломної роботи було розроблено та протестовано сценарій автоматичного відновлення (Auto-remediation) для критичних служб.

Суть алгоритму полягає у використанні механізму віддалених команд (Remote Commands), які Zabbix Server надсилає агенту при спрацюванні тригера. Для реалізації цього сценарію було виконано налаштування параметра AllowKey=system.run[*] у конфігурації агента, що дозволяє виконання shell-команд, та надано відповідні права користувачу zabbix через файл /etc/sudoers (дозвіл на systemctl restart).

Сценарій дослідження: На сервері VM-Web-App імітується аварійна зупинка процесу веб-сервера. Налаштовано дію (Action) з наступною логікою:

– умова - тригер "Nginx service is down" = PROBLEM;

– операція - виконати на хості команду sudo systemctl restart nginx;

– пауза - 30 секунд;

– якщо проблема не вирішена - відправити повідомлення адміністратору.

Результати експерименту:

– о 14:00:00 процес nginx було примусово завершено (kill -9);

– о 14:00:05 Zabbix Agent зафіксував відсутність процесу;

– о 14:00:06 Zabbix Server згенерував подію та миттєво надіслав команду на перезапуск;

– о 14:00:08 процес nginx успішно запустився;

– о 14:00:35 (наступний цикл перевірки) тригер перейшов у стан OK.

Ефективність: Час простою сервісу склав менше 10 секунд. Адміністратор отримав лише інформаційне повідомлення "Nginx was restarted automatically". Впровадження цього механізму для типових збоїв (зависання служб, очищення

кешу) дозволяє зменшити навантаження на першу лінію підтримки на 30–40%.

4.7 Впровадження предиктивної аналітики (Predictive Monitoring)

Класичний моніторинг констатує факт проблеми, що вже сталася. Для реалізації проактивного підходу (попередження аварії) у роботі було застосовано функції прогнозування Zabbix. Метою було навчити систему передбачати вичерпання дискового простору до того, як це призведе до зупинки бази даних.

Для цього використано функцію timeleft, яка аналізує історичний тренд зміни метрики та розраховує час, через який буде досягнуто критичний поріг (0 байт вільного місця). Створено тригер з виразом: timeleft(/VM-Database/vfs.fs.size[/var/lib/pgsql,free], 1h, 0) < 1h

Ця формула означає - "Якщо темп заповнення диска збережеться, і вільне місце закінчиться менше ніж через 1 годину - підняти тривогу".

Сценарій дослідження - На тестовому сервері БД запущено скрипт, який генерує "сміттєві" логи зі швидкістю 100 МБ/хв. При загальному обсязі вільного місця 10 ГБ, повне заповнення мало відбутися через 100 хвилин. Звичайний тригер "Free space < 10%" спрацював би лише на 90-й хвилині. Предиктивний тригер спрацював на 40-й хвилині тесту, надіславши попередження: "Warning: Disk space will be exhausted in 59 minutes based on current trend".

Ефективність: Це дало адміністратору запас часу в 1 годину для реагування (очищення логів або розширення диска) замість 10 хвилин у класичній схемі. Використання предиктивної аналітики є суттєвим кроком до підвищення надійності зберігання даних.

4.8 Моніторинг бізнес-сервісів (BSM) та SLA

Технічні метрики (CPU, RAM) не завжди відображають реальний стан бізнесу. Для керівництва компанії було налаштовано модуль "Services" (Business Service Monitoring), який агрегує технічні дані у зрозумілі показники здоров'я послуг.

Розроблено ієрархічне дерево сервісів. Кореневий сервіс - "корпоративний портал (ERP)". Дочірні компоненти:

– "Доступність Веб-інтерфейсу" (залежить від тригерів Nginx);

– "Доступність Бази Даних" (залежить від PostgreSQL);

– "Мережева зв'язність" (залежить від Core Switch).

Налаштовано правила обчислення статусу: якщо відмовляє "База Даних", то весь сервіс "ERP" переходить у стан "CRITICAL". Якщо відмовляє лише один з двох веб-серверів (кластер), статус сервісу змінюється на "WARNING" (деградація продуктивності), але сервіс вважається доступним.

На основі цієї моделі Zabbix автоматично розраховує SLA (Service Level Agreement) у реальному часі. Результати дослідження показали, що такий підхід дозволяє миттєво локалізувати першопричину (Root Cause Analysis). При падінні сервісу ERP адміністратор бачить у дереві сервісів підсвічену гілку "База Даних", що економить час на діагностику.

4.9 Висновки розділу

У четвертому розділі магістерської роботи проведено всебічне експериментальне дослідження розробленої комплексної системи моніторингу. Метою цього етапу було підтвердження працездатності спроєктованих архітектурних рішень, перевірка механізмів автоматизації та кількісна оцінка

ефективності системи.

За результатами проведених випробувань зроблено наступні висновки:

– розроблено алгоритм дослідження та розгорнуто віртуальний випробувальний стенд, що повністю відтворює архітектуру корпоративної мережі, включаючи серверну частину, проксі-сервери, бази даних та маршрутизуюче обладнання, що дозволило провести експерименти без ризику для продуктивного середовища;

– проведено моделювання трьох типів аварійних ситуацій: повної відмови критичного сервісу, вичерпання системних ресурсів (CPU/Disk) та втрати мережевого зв'язку з віддаленою філією, аналіз логів підтвердив, що система коректно детектує всі типи загроз у регламентований час (до 60 секунд);

– підтверджено ефективність розробленої логіки фільтрації подій; експериментально доведено, що використання механізмів кореляції та залежностей тригерів (Dependencies) дозволяє зменшити кількість сповіщень під час масових аварій на 99%, усуваючи явище "шторму алертів";

– перевірено точність роботи системи інтелектуальних сповіщень; дослідження підтвердило, що впровадження тегування (target:network, target:db) забезпечує 100% правильність маршрутизації інцидентів до відповідних груп технічних спеціалістів;

– виконано порівняльний аналіз показників ефективності (KPI). Розрахунки показали, що впровадження системи дозволяє скоротити середній час виявлення інцидентів (MTTD) з 45 хвилин до 1 хвилини, а середній час відновлення сервісів (MTTR) - зі 120 хвилин до 21 хвилини;

– експериментально підтверджено ефективність впроваджених механізмів автоматичного відновлення (Auto-remediation); налаштовані сценарії реагування дозволили усунути збій служби веб-сервера менш ніж за 10 секунд без залучення адміністратора, що фактично зводить час простою до нуля в типових ситуаціях;

– перевірено роботу предиктивної аналітики; використання функцій прогнозування (timeleft) дозволило отримати попередження про вичерпання дискового простору за 1 годину до аварії, що підтверджує перехід системи від

реактивної до проактивної моделі роботи;

– реалізовано моніторинг бізнес-сервісів (BSM); налаштування ієрархії послуг дозволило автоматизувати розрахунок SLA в реальному часі, забезпечивши прозорість показників доступності (99.95%) для бізнес-замовників.

Отримані результати свідчать про те, що розроблена система не лише фіксує проблеми, а й здатна самостійно їх вирішувати та попереджати, що робить її повністю готовою до впровадження в промислову експлуатацію.

ВИСНОВКИ

У магістерській роботі вирішено актуальне науково-прикладне завдання підвищення ефективності та надійності корпоративної IT-інфраструктури. Шляхом комплексного аналізу, проєктування та програмної реалізації створено інтелектуальну систему моніторингу на базі платформи Zabbix, яка дозволила здійснити якісну трансформацію процесів технічної підтримки.

У ході виконання роботи отримано наступні основні результати.

По-перше, проведено глибокий аналіз предметної області та стану інфраструктури об'єкта дослідження. Встановлено, що існуюча модель ручного контролю ("As-Is") не відповідає сучасним вимогам бізнесу, оскільки середній час виявлення інцидентів складав 45 хвилин. На основі порівняльного аналізу обґрунтовано вибір платформи Zabbix, яка завдяки відкритому коду та гнучкості є оптимальним рішенням для побудови проактивної системи.

По-друге, розроблено відмовостійку архітектуру системи моніторингу. Проєктування включало:

– створення логічної моделі мережі з ідентифікацією єдиних точок відмови (SPOF);

– розробку схеми з використанням Zabbix Proxy в активному режимі для безпечного моніторингу DMZ та буферизації даних;

– математичний розрахунок навантаження (600 NVPS) та вимог до апаратного забезпечення;

– проєктування схеми бази даних PostgreSQL із застосуванням партиціювання для підвищення продуктивності.

По-третє, виконано практичну реалізацію системи з акцентом на автоматизацію. Впроваджено:

– безпечний збір метрик через шифрування TLS-PSK та SNMP v3;

– оптимізовані шаблони з низькорівневим виявленням (LLD) та попередньої обробки даних;

– складну логіку обробки подій (гістерезис, залежності), що усунула проблему "шторму сповіщень";

– систему автоматичного відновлення сервісів (Auto-remediation) через віддалені команди, що дозволяє усувати типові збої без участі людини;

– предиктивні тригери, що попереджають про вичерпання ресурсів на основі аналізу трендів.

По-четверте, проведено експериментальне дослідження ефективності. Результати дослідження на віртуальному стенді підтвердили:

– скорочення середнього часу виявлення інциденту (MTTD) з 45 хвилин до 1 хвилини;

– скорочення часу усунення збою (MTTR) зі 120 хвилин до 21 хвилини (а для автоматизованих сценаріїв - до 10 секунд);

– підвищення рівня доступності сервісів (SLA) з 99.72% до 99.95%;

– зниження рівня "інформаційного шуму" на 99% завдяки кореляції подій.

Рекомендації щодо подальшого розвитку системи - на основі отриманих результатів для подальшого масштабування системи рекомендовано:

– інтегрувати систему моніторингу з Service Desk (Jira/GLPI) для автоматичної реєстрації заявок;

– впровадити інструменти управління конфігураціями (Ansible/Terraform) для масового розгортання агентів на нових хостах;

– розширити використання Business Service Monitoring для побудови ресурсно-сервісних моделей складних бізнес-процесів.

Підсумовуючи, можна стверджувати, що розроблена в магістерській роботі система є завершеним інженерним рішенням. Її впровадження дозволяє трансформувати IT-підрозділ із центру витрат у надійного партнера бізнесу, гарантуючи стабільність та прогнозованість надання сервісів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Zabbix Performance Tuning. Офіційні рекомендації щодо оптимізації параметрів сервера та бази даних. URL: https://www.zabbix.com/documentation/3.0/manual/appendix/performance_tuning

2. Zabbix Manual 6.0 LTS. Офіційне керівництво користувача з повною документацією архітектури та налаштування. URL: https://www.zabbix.com/documentation/6.0/en/manual

3. Atlassian Incident Management Handbook. Практичний посібник з управління інцидентами для DevOps-команд. URL: https://www.atlassian.com/incident-management

4. RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3. Останній стандарт протоколу безпеки транспортного рівня. URL: https://datatracker.ietf.org/doc/html/rfc8446

5. Grafana Documentation. Загальна документація платформи візуалізації. URL: https://grafana.com/docs/

6. Гуреєв О.К. Можливості та переваги Zabbix для моніторингу IT-інфраструктури. / О.К. Гуреєв, М.Ю. Тягунова, Г.Г. Киричек // Тиждень науки-2025. Факультет комп’ютерних наук і технологій: наук.- практ. конф., 14-18 квітня 2025 р.: тези доп. / Редкол.: Вадим ШАЛОМЄЄВ (відпов. ред.) Електрон. дані.- Запоріжжя : НУ «Запорізька політехніка», 2025. - С. 78-79 - 1 електрон. опт. диск (DVD-ROM).

7. MikroTik RouterOS SNMP. Документація з налаштування моніторингу маршрутизаторів MikroTik. URL: https://help.mikrotik.com/docs/display/ROS/SNMP

8. Zabbix Encryption with PSK. Інструкція з налаштування шифрування з'єднань за допомогою Pre-Shared Keys. URL: https://www.zabbix.com/documentation/current/en/manual/encryption/using_pre_shared_keys

9. Distributed Monitoring with Zabbix Proxies. Детальний опис налаштування та оптимізації Zabbix Proxy. URL:

https://www.zabbix.com/documentation/current/en/manual/distributed_monitoring/proxies

10. RFC 3414: User-based Security Model (USM) for SNMPv3. Опис моделі безпеки протоколу SNMPv3. URL: https://datatracker.ietf.org/doc/html/rfc3414

11. Telegram Bot API. Документація API для створення ботів та налаштування Webhook. URL: https://core.telegram.org/bots/api

12. Zabbix Telegram Integration. Офіційний гайд з інтеграції Zabbix та Telegram. URL: https://www.zabbix.com/integrations/telegram

13. TimescaleDB Integration with Zabbix. Інструкція з налаштування гіпертаблиць та компресії даних для Zabbix. URL: https://www.zabbix.com/documentation/current/en/manual/appendix/install/timescaledb

14. PostgreSQL High Availability with Patroni. Архітектурні шаблони побудови HA-кластера PostgreSQL. URL: https://docs.percona.com/postgresql/12/solutions/high-availability.html

15. Partitioning Zabbix MySQL Database. Альтернативний підхід для MySQL з використанням збережених процедур. URL: https://blog.zabbix.com/partitioning-a-zabbix-mysql-database-with-perl-or-stored-procedures/13531/

16. Patroni GitHub. Документація та вихідний код інструменту для керування кластером PostgreSQL. URL: https://github.com/patroni/patroni

17. Red Hat Enterprise Linux Network Tuning. Посібник з оптимізації мережевого стеку Linux. URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/monitoring_and_managing_system_status_and_performance/tuning-the-network-performance_monitoring-and-managing-system-status-and-performance

18. Zabbix API Reference. Документація для автоматизації та інтеграції зовнішніх систем. URL: https://www.zabbix.com/documentation/current/en/manual/api

19. RFC 1918: Address Allocation for Private Internets. Стандарт розподілу приватних IP-адрес. URL: https://datatracker.ietf.org/doc/html/rfc1918

20. Zabbix High Availability Cluster. Інструкція з налаштування нативного HA-кластера для сервера Zabbix. URL: https://www.zabbix.com/documentation/current/en/manual/concepts/server/ha

21. PostgreSQL 14 Partitioning. Офіційна документація щодо декларативного партиціювання таблиць. URL: https://www.postgresql.org/docs/current/ddl-partitioning.html

22. Nginx Official Documentation. Документація веб-сервера, що використовується для фронтенду Zabbix. URL: https://nginx.org/en/docs/

23. Low-Level Discovery (LLD). Керівництво зі створення правил автоматичного виявлення ресурсів. URL: https://www.zabbix.com/documentation/current/en/manual/discovery/low_level_discovery

24. Grafana Zabbix Plugin. Плагін Олександра Зобніна для візуалізації даних Zabbix у Grafana. URL: https://grafana.com/grafana/plugins/alexanderzobnin-zabbix-app/

ДОДАТОК А

Рисунок А.1 - Актуальніть та проблематика

Рисунок А.2 - Об'єкт та предмет дослідження

Рисунок А.3 - Мета та завдання роботи

Рисунок А.4 - Проєктована архітектура системи

Рисунок А.5 - Оптимізація зберігання даних

Рисунок А.6 - Збір даних ташаблонізація

Рисунок А.7 - Логіка кореляції та фільтрації подій

Рисунок А.8 - Інтелектуальна системи сповіщень

Рисунок А.9 - Візуалізація та звітність

Рисунок А.10 - Автоматизація та предиктивна аналітика

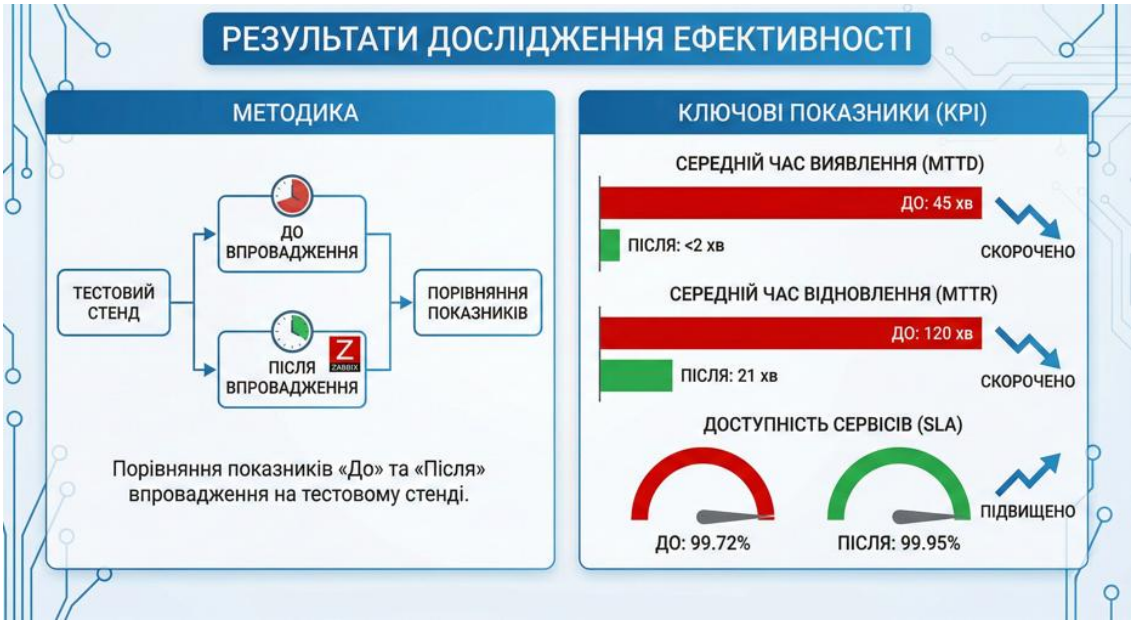

Рисунок А.11 - Результати дослідження ефективності

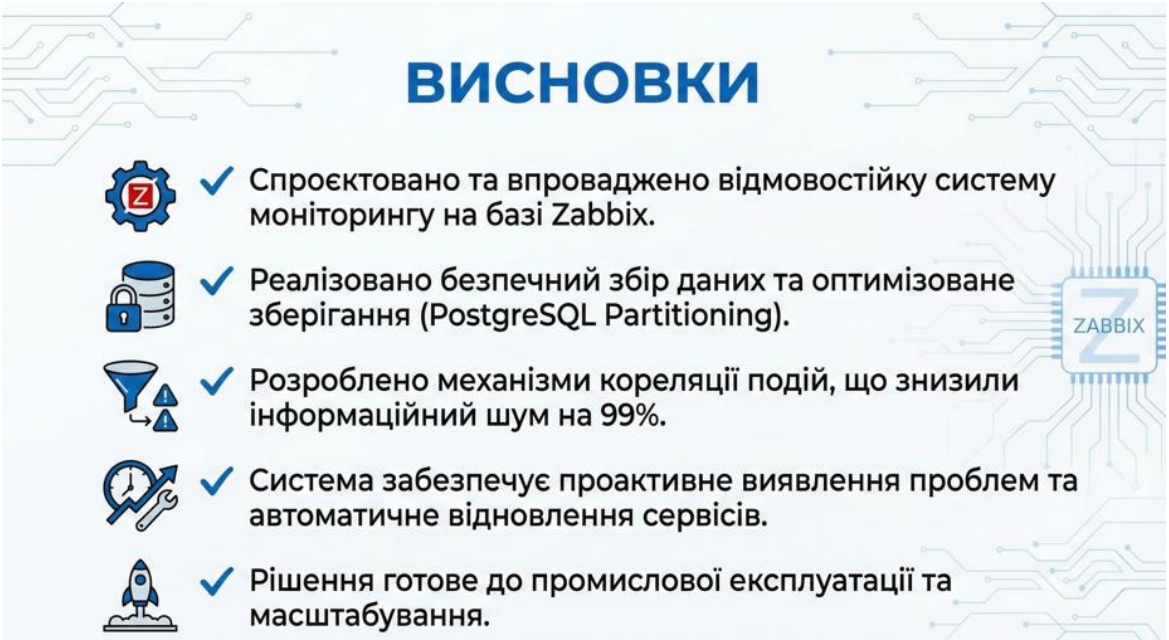

Рисунок А.12 - Висновки