МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
Циклова комісія *комп'ютерних систем та мереж*
(повна назва циклової комісії)

Допустити до захисту
Голова випускової циклової комісії
комп'ютерних систем та мереж
(повна назва циклової комісії)
_____ *Ірина КРАВЧУК*
(підпис) (ім'я, ПРІЗВИЩЕ)
« 10 » 06 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО-ПРОФЕСІЙНОГО СТУПЕНЯ
ФАХОВИЙ МОЛОДШИЙ БАКАЛАВР

Тема: *Структурна схема рішення для моніторингу мережі підприємства*

Група: *321* Спеціальність: *123 «Комп'ютерна інженерія»*

Здобувач освіти _____ *Любов СЕРГАЧОВА*
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ *Ірина ВДОВИЧЕНКО*
(підпис) (ім'я, ПРІЗВИЩЕ)

Консультант з оформлення пояснювальної записки _____ *Оксана ОСАДЧА*
(підпис) (ім'я, ПРІЗВИЩЕ)

Кривий Ріг 2025 р.

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Відділення *комп'ютерної та програмної інженерії*
Циклова комісія *комп'ютерних систем та мереж*
Освітньо-професійний ступінь *фаховий молодший бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Голова випускової циклової комісії
комп'ютерних систем та мереж
(повна назва циклової комісії)
_____ *Ірина КРАВЧУК*
(підпис) (ім'я, ПРІЗВИЩЕ)
« *10* » *06* 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ОСВІТИ

СЕРГАЧОВІЙ Любові Михайлівні
(прізвище, ім'я, по батькові)

1.Тема роботи *Структурна схема рішення для моніторингу мережі*

Керівник роботи *ВДОВИЧЕНКО Ірина Никіфорівна, ктн.,доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по коледжу від « *04* » *04* 2025 року № *50-ст*

2. Строк подання здобувачем освіти роботи з _____ по _____

3. Вихідні дані до роботи *1. Тип мережі – захищена; Сегмент бак - наземний 2. Тип випромінювання – радіовипромінювання; 3. Клас комп. мережі –*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно

Порівняльний аналіз систем моніторингу

Структурна розробка власного рішення для моніторингу

Створення програмного компоненту системи

Порівняльний аналіз систем моніторингу

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

Презентація Microsoft PowerPoint

Презентація Microsoft PowerPoint

6. Консультанти розділів роботи (проекту)

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

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд наявних систем моніторингу*	*01.03.2025-15.03.2025*	*Виконано*
2.	*Аналіз особливостей наявних систем*	*16.03.2025-20.03.2025*	*Виконано*
3.	*Написання вступу та першого розділу*	*21.03.2025-10.04.2025*	*Виконано*
4.	*Аналіз технічних рішень*	*11.04.2025-20.04.2025*	*Виконано*
5.	*Розробка структурної схеми власного рішення для моніторингу*	*21.04.2025-01.05.2025*	*Виконано*
6.	*Написання другого розділу*	*02.05.2025-10.05.2025*	*Виконано*
7.	*Розробка компонента системи*	*11.05.2025-20.05.2025*	*Виконано*
8.	*Написання третього розділу та висновків*	*21.05.2025-30.05.2025*	*Виконано*
9.	*Створення презентації*	*31.05.2025-02.06.2025*	*Виконано*
10.	*Подання роботи на кафедру для перевірки на плагіат*	*02.06.2025-10.06.2025*	*Виконано*
11.	*Захист кваліфікаційної роботи*		

Здобувач освіти _____ *Любов СЕРГАЧОВА*
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ *Ірина ВДОВИЧЕНКО*
(підпис) (ім'я, ПРІЗВИЩЕ)

StrikePlagiarism.com

Дата звіту 6/4/2025

Дата редагування ---

Звіт подібності

метадані

Назва організації

Ukrainian national aviation university

Заголовок

Сергачова_321_на проверку

Автор Науковий керівник / Експерт

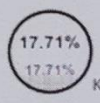
СергачоваКравчук І.

підрозділ

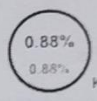
Криворізький Фаховий коледж

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КЦ

25	7190	56534
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	R	0
Інтервали	A	0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)	a	152

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

Колір тексту

10 найдовших фраз

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи “ Структурна схема рішення для моніторингу мережі підприємства”: 52 с., 12 рис., 2 табл., 10 літературних джерел, 2 додатки.

МОНІТОРИНГ, ВІДМОВОСТІЙКІСТЬ , КІБЕРБЕЗПЕКА, КОНФІГУРАЦІЯ, СЕРВЕР, ВТРАТА ДАНИХ

Ця кваліфікаційна робота зосереджена на створенні надійної системи моніторингу корпоративної мережі, що охоплює центральний кампус і географічно розподілені філії. Основний акцент було зроблено на забезпеченні високої доступності та відмовостійкості всіх компонентів.

Для центральної мережі реалізовано конфігурацію активного резервування двох серверів та кластерну базу даних, доповнену двома фронтендами, доступними через балансувальник навантаження. Моніторинг віддалених регіональних офісів здійснюється за допомогою спеціальних агентів.

В рамках дослідження проведено порівняльний аналіз існуючих систем моніторингу, в результаті якого було обрано компоненти Zabbix як фундамент для власного рішення. Сформовані функціональні вимоги до системи включали: збір метрик, наявність веб-інтерфейсу, надсилання сповіщень, підтримку моніторингу розподілених мереж та можливість перевірки статусу.

Для демонстрації можливостей моніторингу розроблено тестову конфігурацію, розгорнуту за допомогою технології контейнеризації Docker. Це включало MariaDB Galera Cluster, Zabbix Server, Zabbix Web та Zabbix Agent, який збирає дані та передає їх на сервер для обробки та зберігання.

Додатково було продемонстровано можливості безагентного моніторингу. Для цього розроблено додаток для автентифікації користувачів на Python з використанням FastApi, що імітував користувацький трафік. Окремий скрипт на Python здійснював паралельний збір метрик цього трафіку та передавав їх до Zabbix через модуль pyzabbix, підтверджуючи ефективність безагентного підходу.

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі комунікаційні мережі становлять фундаментальний базис для глобальної взаємодії та функціонування організацій. Для підтримки конкурентоспроможності та реалізації стратегічних цілей, підприємствам, особливо тим, що мають розподілену структуру, необхідно забезпечувати безперервний доступ співробітників до корпоративних даних та сервісів. Відтак, цілісність інформаційної інфраструктури, що ґрунтується на комунікаційних мережах, є визначальним фактором розвитку та успішності.

Однак, продуктивність мережевих систем піддається впливу численних

дестабілізуючих факторів, таких як зловмисна діяльність, людський фактор, відмови обладнання та програмні помилки. Ці проблеми можуть спричинити значні негативні наслідки, включаючи простої, втрату даних, зниження ефективності, фінансові збитки, репутаційну шкоду та юридичну відповідальність.

Для мінімізації зазначених ризиків критично важливим є безперервний моніторинг корпоративної мережі**.** Цей процес охоплює збір, аналіз та візуалізацію даних про стан мережевої інфраструктури та підключених пристроїв. Основна мета моніторингу полягає у забезпеченні стабільної та ефективної роботи мережі шляхом оперативного виявлення несправностей, аналізу навантаження, гарантування високого рівня безпеки, відстеження статистики використання ресурсів та обґрунтованого планування майбутнього розвитку.

Моніторинг реалізується за допомогою спеціалізованих інструментів і методик, що дозволяють збирати й аналізувати дані про трафік, продуктивність пристроїв та події безпеки. Візуалізація цих даних на інформаційних панелях та у звітах забезпечує адміністраторам можливість оперативного контролю, а автоматизовані сповіщення інформують про виникнення критичних умов. Важливо зазначити, що інструменти моніторингу дозволяють не лише здійснювати контроль у реальному часі, а й аналізувати історичні дані для виявлення довгострокових тенденцій та оптимізації мережевої архітектури.

Дана робота спрямована на демонстрацію фундаментальної ролі сучасних систем моніторингу комунікаційних мереж підприємств, а також на представлення відповідних методів, інструментів та типових завдань. На архітектурному рівні буде запропоновано систему моніторингу, що відрізняється відкритістю архітектури та інтеграцією передових технологій**.** Практична значущість запропонованого архітектурного рішення буде проілюстрована через детальну розробку його програмного компонента.

РОЗДІЛ 1
ПОРІВНЯЛЬНИЙ АНАЛІЗ СИСТЕМ МОНІТОРИНГУ

1.1 Система моніторингу Zabbix

Zabbix — це потужне, безкоштовне програмне забезпечення з відкритим кодом для моніторингу корпоративного рівня. Воно розроблене для відстеження продуктивності та доступності різноманітних IT-ресурсів: мереж, серверів, додатків, служб та інших пристроїв у реальному часі.

Ключові можливості Zabbix:

Комплексний моніторинг: Zabbix збирає тисячі метрик з фізичних і віртуальних машин. Це дозволяє контролювати різноманітні мережеві служби (HTTP, SMTP, POP3, IMAP), а також сервери та програми.

Централізоване управління: Система забезпечує єдину точку контролю для всіх ваших IT-ресурсів.

Гнучкі сповіщення: При досягненні заданих порогових значень Zabbix автоматично сповіщає адміністраторів. Він також може запускати скрипти або команди для автоматичного вирішення виявлених проблем.

Як працює Zabbix: Архітектура системи

Zabbix використовує клієнт-серверну архітектуру, що складається з кількох взаємодіючих компонентів:

Сервер Zabbix: Це основний елемент системи, що відповідає за збір, аналіз і зберігання даних моніторингу в базі даних. Він також генерує звіти та керує всією системою.

Zabbix Agent: Легкий програмний компонент, який встановлюється на контрольованих пристроях. Агент збирає локальну інформацію та відправляє її на Zabbix Server для подальшого аналізу.

Zabbix Proxy: Проміжний компонент, який використовується для моніторингу віддалених або ізольованих мереж. Він збирає дані від імені сервера Zabbix, буферизує їх і потім передає на сервер, зменшуючи його навантаження.

Веб-інтерфейс: Зручний графічний інтерфейс користувача, що дозволяє налаштовувати систему моніторингу, переглядати тривоги, графіки та створювати

звіти з будь-якого місця та пристрою.

База даних: Zabbix Server використовує базу даних (наприклад, MySQL, PostgreSQL, Oracle) для зберігання всіх даних моніторингу, конфігурації та метаданих.

Система сповіщень: Компонент, який відповідає за відправку сповіщень користувачам при виявленні будь-яких проблем у системі.

Zabbix є комплексним рішенням, що дозволяє IT-фахівцям ефективно контролювати стан інфраструктури, швидко реагувати на інциденти та забезпечувати безперебійну роботу всіх IT-сервісів.

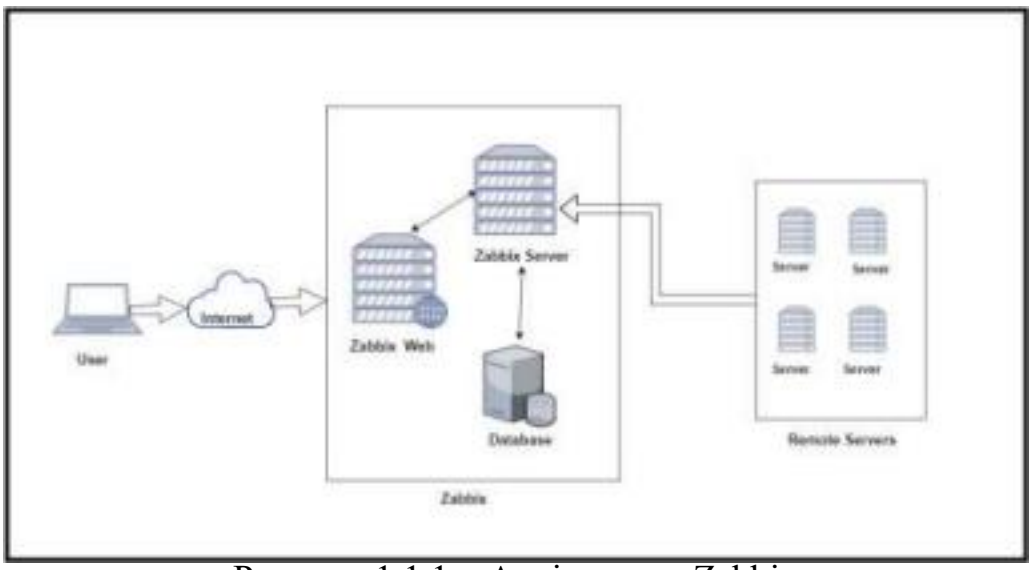


Рисунок 1.1.1 – Архітектура Zabbix

Zabbix: Ключові переваги та можливості

Zabbix є надійним та потужним безкоштовним рішенням для моніторингу, ідеально підходящим для великих корпоративних середовищ. Серед його головних переваг:

• Висока масштабованість: Zabbix дозволяє контролювати навіть дуже великі інфраструктури. Завдяки розподіленому моніторингу, ви можете використовувати кілька Zabbix серверів для відстеження різних сегментів вашої IT-інфраструктури, при цьому всі дані централізовано збираються та аналізуються.

• Гнучкість та кастомізація: Система оснащена потужним API, що дає змогу створювати власні шаблони моніторингу. Ці шаблони можна легко застосовувати до окремих хостів або цілих груп, що значно спрощує налаштування та управління.

• Широкі можливості моніторингу: Zabbix підтримує різні методи моніторингу, включаючи моніторинг за допомогою агентів, без агентів та на основі SNMP. Він також сумісний з різними протоколами та технологіями, такими як ICMP, TCP, UDP, HTTP, FTP, SSH, SMTP та DNS, що забезпечує всебічний контроль над вашою інфраструктурою.

• Інтуїтивна візуалізація: Зібрані дані можна відображати у вигляді графіків, екранів, карт та оглядів, що робить аналіз інформації швидким і зрозумілим.

Загалом, Zabbix є винятково ефективним інструментом для підтримки критично важливих систем у робочому стані завдяки своїм широким можливостям моніторингу, простоті використання, миттєвим сповіщенням, масштабованості та гнучким можливостям звітності.

Система моніторингу Nagios

Nagios — це ще одне безкоштовне програмне забезпечення з відкритим кодом, призначене для безперервного моніторингу систем, мереж та інфраструктури. Створений у 1999 році, Nagios зарекомендував себе як одна з найпопулярніших систем моніторингу.

Його основна функція — контроль за належним функціонуванням всієї IT інфраструктури, включаючи системи, програми, сервіси та бізнес-процеси. У разі виникнення проблем Nagios негайно сповіщає технічний персонал, дозволяючи їм вжити заходів до того, як збій вплине на бізнес-процеси, кінцевих користувачів або клієнтів.

Архітектура Nagios:

Nagios використовує архітектуру клієнт-сервер. Зазвичай, сервер Nagios працює на центральному хості, а плагіни встановлюються на віддалених хостах,

які потрібно контролювати. Плагіни Nagios збирають необхідні дані та передають їх до планувальника процесів, який потім відображає цю інформацію через графічний інтерфейс користувача.

Основні компоненти Nagios:

• Планувальник процесів: Це бек-енд (серверна частина) Nagios, що відповідає за виконання плагінів та надсилання сповіщень на основі отриманих результатів.

• Графічний інтерфейс користувача (GUI): Веб-інтерфейс Nagios, через який користувачі можуть переглядати веб-сторінки, згенеровані CGI, та керувати компонентами IT-інфраструктури.

• Плагіни Nagios: Діляться на два типи:

○ Стандартні плагіни: Готові плагіни, що постачаються разом з Nagios, призначені для моніторингу поширених системних компонентів. ○ Користувацькі плагіни: Плагіни, створені користувачами для моніторингу специфічних компонентів або додатків, унікальних для їхньої організації.

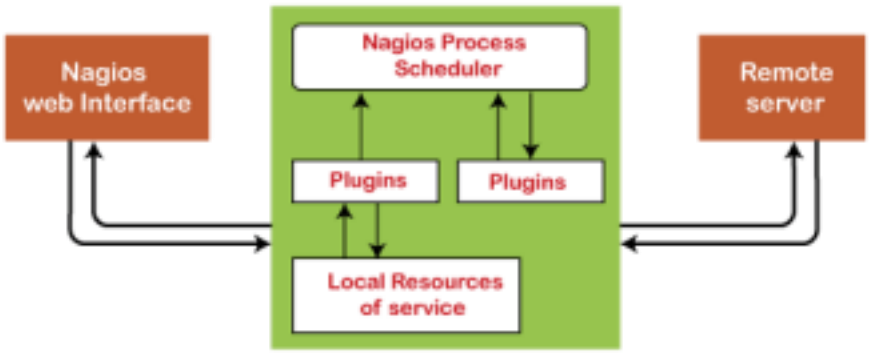


Рисунок 1.2.1 – Архітектура Nagios

Nagios: Основні функції та можливості моніторингу

Nagios — це комплексний інструмент моніторингу з широким спектром функцій, що дозволяють ефективно контролювати IT-інфраструктуру. До його ключових можливостей належать:

• Всебічний моніторинг: Nagios забезпечує моніторинг процесів, сервісів, операційних систем, мережевих протоколів, системних метрик та всіх компонентів

інфраструктури. Це включає контроль за мережевими службами, такими як SMTP, HTTP, FTP, SSH.

• Управління журналами: Система має вбудовану систему управління журналами, що робить її ідеальним сховищем даних для аналізу. • Візуалізація даних: Nagios підтримує плагіни для обробки та відображення графічних даних, що дозволяє наочно представити зібрану інформацію.

• Моніторинг ресурсів сервера: Дозволяє відстежувати пам'ять, процесор, дисковий простір та системні журнали.

• Віддалений моніторинг: Підтримує безпечний віддалений моніторинг через зашифровані тунелі SSL та SSH.

• Керування продуктивністю: Допомагає контролювати та усувати проблеми з продуктивністю сервера.

• Проактивне планування: Nagios дає змогу планувати модернізацію інфраструктури, запобігаючи збоям, спричиненим застарілими системами.

• Автоматичне виправлення: Має можливість автоматично виправляти проблеми у надзвичайних ситуаціях.

• Кросплатформність: Може працювати на будь-якій операційній системі.

• Моніторинг баз даних: Підтримує моніторинг різних серверів баз даних, таких як SQL Server та MySQL.

• Зручний веб-інтерфейс: Надає додатковий веб-інтерфейс для зручного перегляду сповіщень, файлів журналів та іншої інформації.

Система моніторингу Prometheus

Prometheus — це технологія з відкритим кодом, розроблена для моніторингу та сповіщень у хмарних середовищах, включаючи Kubernetes. Створений командою SoundCloud у 2012 році, зараз він підтримується Фондом Cloud Native Computing Foundation (CNCF). Prometheus призначений для збору та моніторингу метрик з різних джерел, таких як сервери, бази даних, програми та мережеві

пристрої.

Він підтримує кілька джерел даних і протоколів, зокрема HTTP, HTTPS, DNS, SNMP та JMX. Система збирає та зберігає великі обсяги даних часових рядів, забезпечуючи видимість продуктивності складних IT-середовищ у реальному часі.

Prometheus написаний мовою програмування Go і має розподілену архітектуру. На відміну від традиційних підходів, Prometheus "збирає" дані замість того, щоб чекати їх надходження, використовуючи виявлення сервісів для пошуку цілей. Зібрані дані надсилаються на панель інструментів для візуалізації та обробляються за допомогою PromQL (мову запитів Prometheus). Сповіщення надсилаються до Менеджера сповіщень, який, у свою чергу, доставляє їх користувачам.

Ключові компоненти системи Prometheus:

• Сервер Prometheus: Основний компонент, що збирає багатовимірні дані у часових рядах, аналізує та агрегує їх.

• Pushgateway (Шлюз Prometheus): Використовується як проміжне джерело для отримання метрик від короткоживучих завдань або тих, що недоступні

звичайними засобами. Він діє як посередник, перетворюючи дані індикатора у формат, зрозумілий для Prometheus.

• Alertmanager (Менеджер сповіщень): Окремий компонент, відповідальний за керування сповіщеннями. Він отримує сповіщення від сервера Prometheus, керує їхнім життєвим циклом (включаючи дедуплікацію та групування за серйозністю), а також перенаправляє їх до відповідних програм, таких як електронна пошта, PagerDuty та Slack.

• Експортери Prometheus: Це сторонні інструменти, які допомагають отримувати метрики, коли вони не можуть бути зібрані безпосередньо. Зазвичай написані на Go, Python або Java, вони взаємодіють із сервером Prometheus, використовуючи протокол HTTP.

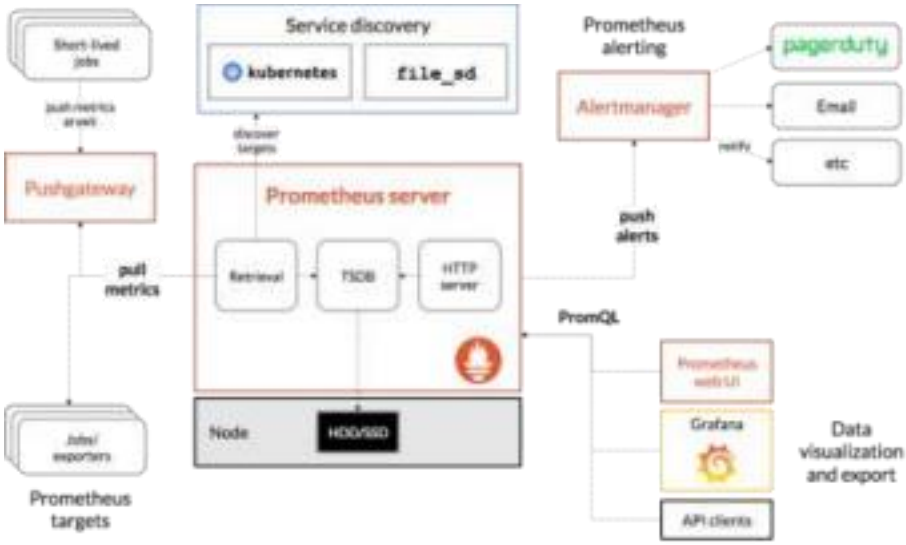


Рисунок 1.3.1 – Архітектура Prometheus [4]

Основні характеристики Prometheus

Prometheus виділяється серед інших систем моніторингу завдяки своїм унікальним характеристикам:

Багатовимірна модель даних: Система використовує дані часових рядів, які ідентифікуються за назвами метрик та парами "ключ-значення". Це дозволяє гнучко аналізувати дані та отримувати більш детальні інсайти.

Гнучка мова запитів PromQL: PromQL (Prometheus Query Language) — це потужна та гнучка мова, спеціально розроблена для роботи з багатовимірними моделями даних Prometheus. Вона дозволяє виконувати складні запити та агрегацію даних.

Автономність вузлів (без залежності від спільного сховища): Кожен сервер Prometheus працює автономно, без необхідності спільного сховища даних. Це підвищує надійність та спрощує розгортання розподілених систем.

Модель збору даних "витягування" (pull model): Prometheus активно "витягує" дані (метрики) з контрольованих систем через HTTP, що є відмінністю від традиційних систем, де агенти "відштовхують" дані.

Вбудована візуалізація: Prometheus надає широкий спектр типів діаграм та інформаційних панелей для візуалізації зібраних даних, хоча для більш складних візуалізацій часто використовується інтеграція з Grafana.

Тип моніторингу	Zabbix	Nagios	Prometheus
Моніторинг мережі	Так	Так	Так
Серверний моніторинг	Так	Так	Так
Хмарний моніторинг	Так	Ні	Так
Системний моніторинг	Так	Так	Так
Моніторинг баз даних	Так	Ні	Так
Моніторинг IT-інфраструктури	Так	Так	Так
Моніторинг контейнерів	Так	Ні	Так
Моніторинг додатків	Так	Ні	Так

Рисунок 1.4.1-типи моніторингу

Порівняльний аналіз Zabbix, Nagios та Prometheus

Zabbix, Nagios та Prometheus — це популярні рішення з відкритим кодом для моніторингу IT-інфраструктури, що відстежують стан серверів, мереж, віртуальних машин та інших компонентів. Усі три системи є високомасштабованими та пропонують широкий спектр можливостей, включаючи збір даних, оповіщення, візуалізацію, налаштування та розширення. Проте, кожна з них має свої унікальні переваги та недоліки, що робить їх придатними для різних потреб.

Стратегії збору даних та архітектура

Zabbix та Nagios базуються на клієнт-серверній архітектурі. Агенти встановлюються на моніторингових системах для збору та передачі даних на центральний сервер. Вони також підтримують моніторинг без агентів. Zabbix вимагає зовнішньої бази даних (наприклад, MySQL або PostgreSQL), що додає трохи складності в налаштуванні порівняно з Nagios.

Prometheus використовує інший підхід: його сервер періодично "витягує" метрики з контрольованих систем. Він також використовує PromQL для отримання

та агрегації даних часових рядів з багатовимірної моделі даних у реальному часі. Ця модель робить Prometheus особливо придатним для моніторингу хмарних систем та мікросервісних архітектур, де ресурси динамічно створюються та знищуються.

Усі три системи підтримують виявлення аномалій та прогнозний моніторинг, але з різними підходами.

Типи моніторингу та протоколи

Усі інструменти підтримують як пасивну, так і активну перевірку. Пасивна перевірка передбачає отримання даних від моніторингової системи, тоді як активна — надсилання запитів для збору даних.

Вони також підтримують SNMP моніторинг, що є стандартом для мережевих пристроїв (маршрутизаторів, комутаторів).

Специфічні можливості:

Prometheus є єдиним інструментом, що підтримує JMX-моніторинг (для Java-додатків) та хмарний моніторинг "з коробки". Це робить його кращим вибором для організацій, що покладаються на хмарну інфраструктуру.

Zabbix, у свою чергу, є єдиним, що підтримує IPMI моніторинг (для серверного обладнання, як-от температура та швидкість вентиляторів), а також моніторинг Docker та Kubernetes, що робить його ідеальним для контейнеризованих середовищ.

Налаштування та користувацький досвід

Prometheus значно легше налаштовувати, ніж інші системи, і має велику спільноту для підтримки.

З точки зору візуалізації даних, Zabbix має явну перевагу. Він дозволяє створювати діаграми, карти та інформаційні панелі без необхідності сторонніх сервісів, таких як Grafana. Zabbix також має краще розроблений інтерфейс користувача, ефективніші рішення для управління користувачами та розширені можливості сповіщень.

Висновок

Кожна з розглянутих систем має свої сильні та слабкі сторони, що робить їх придатними для різних сценаріїв використання. Zabbix та Nagios зазвичай краще підходять для мереж малих та середніх підприємств. Натомість Prometheus є системою моніторингу, оптимізованою для великих корпоративних мереж та хмарних середовищ. Зокрема, Prometheus дозволяє створювати розподілені системи моніторингу з кількома повнофункціональними вузлами, що ідеально підходить для сучасних підприємств з географічно розподіленими офісами. Проте, важливою особливістю Prometheus є його складність.

У наступному розділі розглянемо побудову системи моніторингу, яка поєднує можливості розподіленого розгортання (як у Prometheus) зі спрощеністю налаштування, подібною до Zabbix та Nagios.

РОЗДІЛ 2
АРХІТЕКТУРНА РОЗРОБКА ВЛАСНОГО РІШЕННЯ ДЛЯ МОНІТОРИНГУ

Цей розділ присвячений детальному опису розробки системи моніторингу, яка відповідає трьом ключовим вимогам:

- Децентралізоване використання: Можливість ефективно працювати в розподілених мережах.
- Стійкість до збоїв: Система має бути толерантною до помилок.
- Низька складність: Рівень складності повинен бути порівнянним із Zabbix та Nagios.

Після ретельного аналізу існуючих технологій, підходів до розробки та готових компонентів для моніторингу мереж, було вирішено використовувати компоненти Zabbix як основу для створення системи з необхідними показниками моніторингу.

2.1 Розподілений моніторинг у Zabbix

Розподілений мережевий моніторинг (DMN) — це стратегія, яка дозволяє збирати дані про продуктивність мережі з кількох точок спостереження (агентів моніторингу або датчиків). Цей підхід допомагає ізолювати проблеми мережі до конкретної програми чи пристрою, а також визначити, чи проблема пов'язана з мережею, чи з самою програмою.

У розподіленій системі моніторингу, кілька серверів або агентів розгортаються у різних географічних локаціях чи сегментах мережі. Ці сервери збирають дані з різних пристроїв і систем, а потім надсилають їх на центральний сервер для обробки та аналізу.

У Zabbix розподілений моніторинг реалізується за допомогою Zabbix Proxy. Zabbix Proxy діє як посередник між хостами, що моніторяться, і центральним сервером Zabbix. Проксі-сервер збирає дані від кількох агентів Zabbix, кешує їх, а

потім періодично пересилає на основний сервер. Це значно зменшує мережевий трафік та навантаження на центральний сервер, дозволяючи йому ефективно обробляти велику кількість контрольованих хостів та елементів моніторингу. Як це працює:

1. Агент Zabbix на контрольованому хості надсилає дані моніторингу на Zabbix Proxy.
2. Проксі-сервер перевіряє свої кешовані дані. Якщо дані для цього хоста вже є, він оновлює їх новими значеннями. В іншому випадку, нові дані додаються до кешу.
3. Проксі-сервер періодично надсилає кешовані дані на центральний сервер Zabbix.
4. Сервер Zabbix зберігає отримані дані у базі даних, обробляє їх та використовує для створення звітів, сповіщень або тривог.

Використання Zabbix Proxy є найпростішим способом реалізації як централізованого, так і розподіленого моніторингу, де всі агенти та проксі-сервери звітують на один Zabbix Server, забезпечуючи централізований збір усіх даних. Zabbix Proxy ідеально підходить для:

• Віддаленого моніторингу філій або географічно розподілених локацій.

• Контролю об'єктів з ненадійним зв'язком.

• Зменшення навантаження на Zabbix Server при моніторингу тисяч пристроїв.

• Спрощення обслуговування розподілених систем моніторингу.

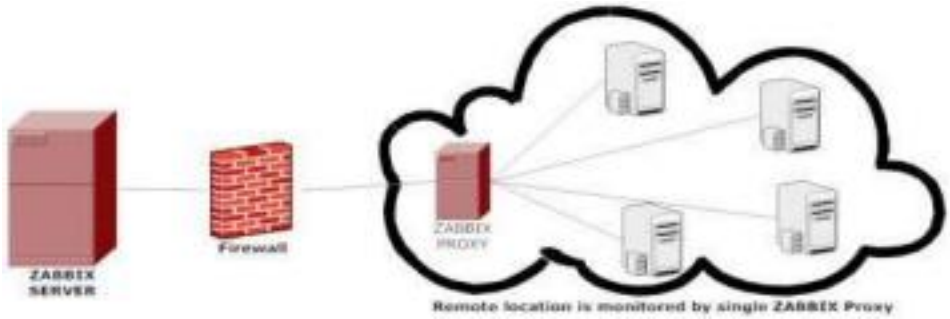


Рисунок 2.1.1 – Розподілена архітектура моніторингу з використанням агентів [5]

Простота налаштування та стійкість до збоїв Zabbix Proxy

Zabbix Proxy значно спрощує налаштування мережі. Для його роботи потрібно лише одне TCP-з'єднання з центральним сервером Zabbix. Це суттєво полегшує конфігурацію брандмауерів, оскільки замість численних правил достатньо налаштувати лише одне.

Важливою перевагою Zabbix Proxy є його стійкість до тимчасових збоїв зв'язку. Всі дані, зібрані проксі, зберігаються локально, перш ніж бути переданими на сервер. Це гарантує, що жодні дані не будуть втрачені, якщо виникнуть тимчасові проблеми з підключенням до сервера Zabbix. Тривалість локального зберігання даних визначається параметрами ProxyLocalBuffer та ProxyOfflineBuffer у конфігураційному файлі проксі-сервера.

Функціональні обов'язки Zabbix Proxy

Zabbix Proxy виконує роль збирача даних. Він не відповідає за обробку тригерів, подій або надсилання сповіщень. Ці функції залишаються за центральним сервером Zabbix.

Для кращого розуміння функціональності Zabbix Proxy, розгляньмо наступну таблицю :

Функція	**Підтримка аген**
- Вбудована мережа Моніторинг	Так
Значення попередньої обробки елемент	Так
Автоматична реєстрація агент	Так
Виявлення низького рівня Вразливості	Так
Дистанційне керування	Так
Тригер обчислення	Ні
Обробка подій	Ні
Кореляція подій	Ні
Надіслати сповіщення	Ні

Рисунок 2.1.2 – Функція проксі-сервера

Після відновлення роботи сервера Zabbix, особливо якщо він був офлайн

тривалий час, існує ризик його перевантаження через великий обсяг зібраних даних, що накопичилися в проксі-серверах. Це може призвести до того, що кеш історії буде заповнений на 95-100%, сповільнюючи обробку перевірок і загальну продуктивність системи.

Коли кеш історії сервера Zabbix переповнюється, він припиняє доступ до нових записів, фактично зупиняючи збір даних. Найчастіше це відбувається, коли проксі-сервери намагаються одночасно завантажити всі накопичені дані після простою сервера. Щоб запобігти такому перевантаженню, в Zabbix реалізовано механізм дроселювання (обмеження), який наразі не можна вимкнути. Як працює механізм дроселювання:

1. Порогове значення 80%: Коли використання кешу історії досягає 80%, сервер Zabbix припиняє приймати дані від проксі-серверів.

2. Список обмеженого доступу: Проксі-сервери, що намагаються відправити дані, додаються до спеціального списку обмеженого доступу. 3. Зниження до 60%: Сервер відновлює прийом даних лише тоді, коли використання кешу знижується до 60%.

4. Послідовна обробка: Дані від проксі-серверів зі списку обмеженого доступу приймаються послідовно, у порядку їх надходження. Це означає, що перший проксі, який спробував завантажити дані під час періоду дроселювання, буде обслужений першим, і дані від інших проксі не будуть прийматися, доки операція з першим не завершиться.

5. Завершення дроселювання: Цей режим триватиме, поки використання кешу знову не досягне 80%, не знизиться до 20%, або список обмеженого доступу не стане порожнім. У першому випадку сервер знову припинить приймати дані. В інших двох випадках сервер повернеться до нормального режиму роботи, приймаючи дані від усіх агентів.

Цей механізм забезпечує стабільність роботи сервера Zabbix, запобігаючи його перевантаженню та втраті даних у періоди пікових навантажень.

2.2 Висока доступність (HA) у Zabbix

Висока доступність (HA) є критично важливим аспектом будь-якої системи моніторингу, включно з Zabbix. HA гарантує безперервну роботу системи без збоїв протягом певного періоду часу, мінімізуючи вплив на бізнес-операції.

Для досягнення високої доступності в Zabbix розгортається кластерна архітектура, де кілька серверів Zabbix працюють разом. Кожен сервер у кластері може контролювати певний набір хостів або пристроїв.

Принцип роботи HA-кластера Zabbix:

• Активний та резервний вузли: У кластері завжди є один активний сервер Zabbix, який виконує всі операції моніторингу. Інші сервери перебувають у режимі очікування (резервні вузли), готовими негайно взяти на себе функції активного сервера в разі його збою.

• Спільна база даних: Усі сервери в кластері використовують спільну базу даних для зберігання всіх даних моніторингу. Це забезпечує єдність і цілісність даних незалежно від того, який вузол є активним.

• Відстеження стану вузлів: Стан усіх вузлів кластера відстежується і зберігається в таблиці ha_node бази даних.

• Оновлення стану: Кожен вузол кластера повідомляє про свій стан кожні 5 секунд, оновлюючи відповідний запис у таблиці ha_node. • Моніторинг резервним вузлом: Резервний вузол постійно відстежує час останнього доступу активного вузла в таблиці ha_node. • Автоматичне перемикання при відмові: Якщо різниця між часом останнього доступу активного вузла та поточним часом досягає визначеної "затримки відновлення", кластер автоматично перемикається на один з резервних вузлів. Ця затримка може бути налаштована користувачем. • Журналювання подій: Усі операції резервного перемикання (failover) записуються в журналі сервера Zabbix.

• Автоматичне перемикання інтерфейсу: Zabbix Frontend (веб інтерфейс) автоматично переключається на активний вузол сервера Zabbix,

забезпечуючи безперебійний доступ користувачів.

Кластер серверів високої доступності Zabbix підтримує необмежену кількість вузлів, що дозволяє будувати дуже надійні та стійкі системи моніторингу.

Рисунок 2.2.1 – Архітектура HA в Zabbix [7]

У архітектурі високої доступності Zabbix, системи поділяються на два ключові типи серверів:

- Активний сервер: Цей сервер є основним і відповідальним за безпосередній моніторинг хостів та збір даних. Він активно виконує всі операції моніторингу.
- Пасивний сервер: Діє як резервний, перебуваючи в режимі очікування. Він готовий негайно перейняти на себе завдання моніторингу, якщо активний сервер вийде з ладу.

Кожен вузол (сервер) в HA-кластері має такі характеристики:

- Єдина конфігурація: Усі вузли мають ідентичну конфігурацію, що спрощує управління.
- Спільна база даних: Всі вузли використовують одну й ту саму базу даних для зберігання моніторингових даних.
- Кілька режимів роботи: Вузол може перебувати в одному з режимів: активний (active), очікування (standby), недоступний (unavailable) або зупинений (stopped).

Одночасно лише один вузол може бути активним. Резервний вузол виконує лише одну програму — менеджер високої доступності. Він не збирає та не обробляє дані, а також не бере участі в інших звичайних серверних операціях (не прослуховує порти та має мінімальні підключення до бази даних).

Переваги високої доступності:

- Безперервність роботи: HA гарантує, що служба моніторингу залишається працездатною та функціональною навіть при збоях обладнання чи програмного забезпечення. Це запобігає простоям і забезпечує постійний доступ до даних моніторингу.
- Покращення продуктивності: Конфігурація високої доступності допомагає розподілити навантаження між кількома серверами, тим самим покращуючи загальну продуктивність і час відгуку системи.

- Просте масштабування: HA дозволяє легко масштабувати інфраструктуру моніторингу в міру зростання організації. Додавання нових серверів до кластера високої доступності є відносно простим і не вимагає переривання роботи служб моніторингу.

Загалом, впровадження високої доступності в Zabbix допомагає забезпечити безперервну доступність та оптимальну продуктивність служб моніторингу, що сприяє покращенню бізнес-операцій та зменшенню витрат.

2.3 Методи управління базою даних у Zabbix

Масштабованість, доступність і продуктивність будь-якої системи моніторингу, включно з Zabbix, значною мірою залежать від архітектури її бази даних. Zabbix пропонує два основні підходи до управління базами даних: база даних для кожного сервера та спільна база даних. Розглянемо їхні переваги та недоліки, щоб допомогти вибрати оптимальне рішення.

База даних на сервер:

Ця архітектура передбачає налаштування окремої бази даних для кожного сервера Zabbix. Кожна база даних зберігає дані моніторингу, зібрані відповідним

сервером.

Переваги:

- Масштабованість: До системи можна легко додавати додаткові сервери з власними базами даних.
- Висока відмовостійкість: Оскільки кожен сервер має власну базу даних, збій одного сервера не вплине на роботу інших.
- Покращення продуктивності: Розподіл збору та обробки даних між кількома серверами зменшує навантаження на кожен окремий сервер Zabbix.

Недоліки:

- Складність: Управління численними окремими базами даних може бути складним і трудомістким.

- Збільшення вартості: Цей підхід вимагає більше апаратних ресурсів, оскільки кожен сервер потребує власної бази даних.
- Синхронізація даних: Забезпечення узгодженості даних між окремими базами може бути складним, що потенційно призводить до розбіжностей.
- Відсутність централізації: Оскільки кожен сервер має власну базу даних, важко отримати єдине, консолідоване уявлення про мережу. Спільна база даних:

У цьому підході кілька серверів Zabbix використовують один і той самий сервер бази даних. Кожен сервер Zabbix відповідає за збір даних моніторингу зі своїх призначених пристроїв та їх зберігання в спільній базі даних. Переваги:

- Економічність: Цей метод потребує лише одного сервера бази даних, що знижує витрати на обладнання та ліцензування.
- Централізований вигляд: Використання єдиної бази даних кількома серверами забезпечує більш централізоване та цілісне уявлення про мережу.

Недоліки:

• Низька масштабованість та доступність: Спільна база даних може стати єдиною точкою відмови та вузьким місцем продуктивності при великих навантаженнях.

• Ризик пошкодження та невідповідності даних: Одночасні оновлення від різних служб можуть збільшити ймовірність пошкодження або невідповідності даних.

• Знижена автономія та гнучкість: Усі сервери використовують одні й ті самі ресурси бази даних та дозволи, що обмежує їхню автономність. • Складність: Управління кількома серверами Zabbix, які працюють на спільному сервері бази даних, може бути складним і трудомістким.

2.4 Кластер бази даних

У будь-якій системі база даних часто є однією з головних точок збою або зниження продуктивності, тому її архітектура є надзвичайно важливою. Кластеризація баз даних є чудовим рішенням для забезпечення відмовостійкості та високої продуктивності у розроблюваному рішенні. Кластеризація баз даних — це процес об'єднання кількох серверів баз даних для спільної роботи як єдина система. Це допомагає розвантажити систему, розподілити навантаження та виконувати технічне обслуговування без зупинки системи.

Кластер даних зазвичай складається з двох або більше вузлів (серверів). Ці сервери розподіляють робоче навантаження, забезпечують резервування та покращують доступність. Це досягається шляхом реплікації даних на кількох серверах. Кожен вузол кластера працює синхронно, тобто він містить ті самі дані, що й усі інші вузли. Таким чином, якщо один сервер вийде з ладу, дані залишаються доступними на іншому сервері.

Існує два основних типи кластерних архітектур: shared-nothing (без спільного використання нічого) та shared-disk (зі спільним диском). Архітектура Shared-Nothing (Без спільного використання нічого) У цій архітектурі кожен вузол/сервер бази даних є повністю незалежним. Жоден окремий сервер бази даних не є "головним" (master) сервером, і немає центрального вузла бази даних,

який би відстежував і контролював доступ до даних у системі. Архітектура "без спільного використання нічого" забезпечує відмінну горизонтальну масштабованість, оскільки між вузлами чи серверами баз даних не використовуються спільні ресурси, що усуває потенційні вузькі місця.

Рисунок 2.4.1 – Архітектура «Спільне нічого»

Архітектура Shared-Disk (Зі спільним диском)

На противагу архітектурі "shared-nothing", в архітектурі "shared-disk" усі вузли мають спільний доступ до всіх доступних серверів баз даних. Це означає, що будь-який вузол може отримати доступ до всіх даних у системі. На відміну від "shared-nothing", тут рівень мережевого з'єднання розташований між центральним процесором і сервером бази даних, що дозволяє отримувати доступ до кількох серверів баз даних.

Варто зазначити, що кластери зі спільним диском зазвичай не пропонують такої значної масштабованості, як архітектури без спільного використання. Це пов'язано з тим, що для ефективного контролю потоку даних у системі (особливо коли всі вузли мають спільний доступ до одних і тих самих даних) потрібен вузол керування. Проблема виникає, коли кількість підлеглих вузлів перевищує певну межу: головний вузол стає нездатним ефективно контролювати та керувати ними.

Зближення архітектур: Мережеві обчислення та розподілене кешування Різниця між цими двома архітектурами останнім часом дещо розмилася завдяки появі

мережевих обчислень та розподіленого кешування. У такій конфігурації дані все ще централізовано керуються, але цей контроль здійснюється

потужним "віртуальним сервером", який фактично складається з багатьох фізичних серверів, що працюють разом як єдине ціле.

Рисунок 2.4.2 – Архітектура спільного диска [8]

Переваги кластерів баз даних

Впровадження кластерів баз даних у систему моніторингу, таку як Zabbix, надає низку значних переваг:

Масштабованість: Кластерні бази даних розроблені для обробки великих обсягів даних і легко масштабуються відповідно до їх зростання. У міру розширення потреб моніторингу мережі, кластер може легко розширюватися разом з ними, дозволяючи ефективно збирати та аналізувати дані.

Висока доступність: Розподілений мережевий моніторинг вимагає високого рівня доступності для забезпечення безперервності даних та постійного моніторингу. Кластерні бази даних гарантують безперервний доступ до даних навіть у разі збоїв апаратного чи програмного забезпечення.

Відмовостійкість: Кластерні бази даних також забезпечують відмовостійкість, що означає, що вони продовжують функціонувати навіть у разі збою мережі або сервера. Це гарантує цілісність ваших даних моніторингу навіть за непередбачених обставин.

Балансування навантаження: Кластерні бази даних можна налаштувати для автоматичного розподілу навантаження між кількома вузлами, гарантуючи, що жоден вузол не буде перевантажений даними. Це допомагає підтримувати стабільну продуктивність та забезпечує своєчасний збір і аналіз даних моніторингу.

Запобігання простоям: Кластеризація може допомогти запобігти тривалим простоям через збої обладнання. Ресурси програми можуть бути швидко переміщені на інший вузол кластера, часто навіть без втрати клієнтського з'єднання.

Спрощена модернізація: Традиційні оновлення серверів часто вимагають значного часу простою та складних міграцій. Використання кластерів значно спрощує ці процеси з мінімальним часом простою. Додайте нові вузли до кластера, встановіть усі необхідні оновлення, а потім, за допомогою процедури відмовостійкості, перемістіть сервер на новий вузол і видаліть старий з кластера.

2.5 Кластер Galera

Для забезпечення кластеризації баз даних у цьому рішенні буде використано кластер Galera. Galera Cluster — це синхронний кластер баз даних з кількома головними серверами, що базується на синхронній реплікації MySQL та InnoDB.

Ключова перевага Galera Cluster полягає в тому, що операції зчитування та запису з бази даних можна спрямовувати на будь-який вузол. Це означає, що кожен окремий вузол може бути втрачений без простою системи або складних процедур відновлення після відмови.

Внутрішня архітектура кластера Galera складається з таких чотирьох компонентів:

Система керування базами даних (СУБД): Це центральний блок кластера. Кожен вузол запускає відповідний сервер бази даних. Кластер Galera підтримує популярні СУБД, такі як MariaDB, MySQL та Percona XtraDB.

API wsrep: Цей API визначає та реалізує інтерфейс і обов'язки для доступу до підключеного сервера бази даних, а також регулює реплікацію даних. Він складається з двох основних елементів:

Перехоплювачі wsrep: Посилаються на сервер бази даних для реплікації даних.

dlopen(): Надає функції для зв'язку з перехоплювачами wsrep.

Плагін реплікації Galera: Цей плагін реалізує API wsrep. Він забезпечує рівень автентифікації, рівень реплікації (включаючи протоколи реплікації) та структуру групового зв'язку.

Плагіни для групового спілкування: Galera Cluster має кілька плагінів для впровадження систем групового спілкування, таких як інструментарій Поширення та gcomm. Архітектура цих плагінів забезпечується платформою групового спілкування.

Різниця між цими двома архітектурами нещодавно розмилася з появою мережевих обчислень або розподіленого кешування. У цій конфігурації дані все ще централізовано керуються, але контролюються потужним «віртуальним сервером», що складається з багатьох серверів, що працюють разом як єдине ціле.

Рисунок 2.5.1 – Архітектура кластера Galera [10]

Кластер Galera функціонує на основі Galera Replication Plugin, який розширює стандартний API реплікації MySQL (зокрема, API реплікації наборів запису, або API wsrep). Це дозволяє забезпечити синхронну реплікацію на кількох головних серверах.

Механізм реплікації:

1. Сертифікація транзакцій: Galera Cluster використовує реплікацію на основі сертифікатів. Кожна транзакція, що реплікується (набір записів), містить не лише дані, які потрібно реплікувати, але й інформацію про всі блокування, що утримуються базою даних під час транзакції.

2. Перевірка та застосування: Кожен вузол кластера перевіряє реплікований набір записів на наявність конфліктів з іншими наборами записів, що очікують на застосування.

3. Фіксація: Якщо конфліктуючих блокувань немає, набір записів вважається завершеним (сертифікованим), і кожен вузол негайно застосовує його до свого табличного простору. Це гарантує, що всі вузли кластера завжди мають ідентичний вигляд бази даних.

Для стабільної роботи мінімальний кластер Galera повинен складатися з 3 вузлів, і взагалі рекомендується працювати з непарною кількістю вузлів. Це забезпечує кворум (більшість). Якщо один вузол вийде з ладу (наприклад, через проблеми з мережею або відмову машини), два інші вузли все ще матимуть кворум і зможуть продовжувати обробку транзакцій без перерви.

Ключові особливості Galera Cluster:

• Реплікація на основі сертифікатів: На відміну від традиційної реплікації MySQL (де головна база даних передає двійкові журнали підлеглим), Galera використовує сертифікацію кожної транзакції кожним вузлом перед її фіксацією. Це запобігає затримкам і невідповідностям даних, забезпечуючи повну узгодженість між усіма вузлами.

• Автоматична ініціалізація та відновлення вузлів: При додаванні нового вузла до кластера він автоматично синхронізується з існуючими

вузлами та стає повноцінним членом. У разі збою вузла, решта кластера продовжує працювати, а вузол, що вийшов з ладу, може бути замінений новим, який також автоматично синхронізується.

• Балансування навантаження та маршрутизація запитів: Будь-який вузол у кластері може обробляти запити на читання та запис. Балансувальник навантаження може розподіляти запити по кластеру, забезпечуючи рівномірний розподіл робочого навантаження. Маршрутизація запитів гарантує, що запити на запис надсилаються до правильного вузла на основі розділу даних (якщо такий є).

Переваги Galera Cluster:

• Відсутність затримки реплікації: Завдяки синхронній реплікації всі вузли миттєво синхронізуються, гарантуючи постійну актуальність та узгодженість даних.

• Відсутність втрачених транзакцій: Galera гарантує, що всі транзакції будуть зафіксовані на всіх вузлах кластера, перш ніж вважатися завершеними, що виключає втрату даних через збої вузлів.

• Масштабованість читання: Кластер забезпечує масштабованість читання, розподіляючи трафік читання між усіма вузлами за допомогою балансування навантаження.

• Менша затримка клієнта: Завдяки реплікації з кількома головними вузлами, запити клієнтів обробляються швидше, оскільки вони можуть підключатися до найближчого доступного вузла, зменшуючи затримку.

• Відмінна підтримка хмарних технологій: Galera ідеально підходить для гнучкого розширення ресурсів бази даних у хмарі та спрощує розподіл даних між різними центрами обробки даних, оскільки кожна транзакція надсилається до кожного центру лише один раз.

2.6 Структурна діаграма: Розроблене рішення для моніторингу Після

аналізу різних технічних варіантів було розроблено індивідуальне рішення для моніторингу, що відповідає таким ключовим вимогам: • Відмовостійкість: Система повинна бути здатною витримувати навантаження та продовжувати функціонувати навіть у разі відмови окремих компонентів, забезпечуючи безперервний доступ до інформації. • Розподіленість: Система має бути розподіленою, щоб ефективно обробляти інформацію з будь-яких регіональних офісів, скорочуючи час реагування.

Для забезпечення цих вимог (відмовостійкості та розподіленості) розроблено таку архітектуру корпоративного рішення для моніторингу:

Рисунок 2.6.1 – Блок-схема рішення для моніторингу

Ефективний моніторинг мережі організації з головним офісом і кількома регіональними відділеннями вимагає комплексного підходу та застосування

різноманітних технологій. Наша стратегія зосереджена на побудові високонадійної інфраструктури, що забезпечує безперервність моніторингу та доступність даних. Першим кроком до створення надійної мережевої інфраструктури є розгортання

відмовостійкої конфігурації в головному офісі. Це досягається завдяки використанню двох серверів Zabbix у резервній структурі. Таке рішення гарантує, що у разі збою одного сервера, інший миттєво перебере його функції, забезпечуючи безперебійну роботу системи моніторингу та, відповідно, мережевої інфраструктури в цілому.

База даних є критично важливим компонентом будь-якої корпоративної мережі. Збій бази даних може призвести до втрати метрик, що матиме катастрофічні наслідки для бізнесу. Для забезпечення безперервного доступу до інформації буде налаштовано кластер MariaDB Galera. Три сервери баз даних утворять кластер, працюючи разом для забезпечення високої доступності. Ця технологія заснована на синхронній реплікації даних між усіма вузлами кластера, що гарантує відсутність втрат даних у разі збою будь-якого компонента.

Взаємодія користувачів із додатками відбувається через інтерфейси, тому збої в них можуть призвести до втрати з'єднання та негативно вплинути на користувацький досвід. Для вирішення цієї проблеми ми створюємо відмовостійку конфігурацію інтерфейсу, використовуючи два фронтенди. Вони працюватимуть спільно, забезпечуючи постійний доступ користувачів до необхідних даних. Якщо один інтерфейс вийде з ладу, інший зможе взяти на себе обслуговування користувачів без перерви.

Для забезпечення безперервного доступу до фронтенду, перед ним буде розміщено балансувальник навантаження. Коли користувач надсилає DNS-запит, DNS-сервер розподіляє доменне ім'я на дві різні IP-адреси – по одній для кожного балансувальника навантаження. DNS-сервер відповідає однією з цих IP-адрес за циклічним принципом (round-robin), рівномірно розподіляючи запити між двома балансувальниками. У свою чергу, балансувальник навантаження також рівномірно розподіляє запити користувачів між двома інтерфейсами, запобігаючи

перевантаженню будь-якого з них. Крім того, у разі збою одного інтерфейсу, балансувальник навантаження забезпечує відмовостійкість, перенаправляючи запити на робочий інтерфейс.

Розподілений моніторинг регіональних офісів

Для мережевої інфраструктури з кількома регіональними офісами розподілений моніторинг є критично важливим, оскільки він дозволяє завчасно виявляти проблеми та контролювати кожен офіс окремо. Для моніторингу регіональних офісів буде використано проксі-сервер як шлюз між головним офісом та регіональними відділеннями. Агенти в регіональних офісах збиратимуть дані та надсилатимуть їх до головного офісу через проксі-сервер для подальшого аналізу. Це дозволить мережевим адміністраторам контролювати стан мережі кожного регіонального офісу в режимі реального часу та оперативно виявляти потенційні проблеми.

РОЗДІЛ 3
ЗБІРКА ПРОГРАМНИХ КОМПОНЕНТІВ СИСТЕМИ

У цьому розділі ми детально опишемо покроковий процес налаштування тестової конфігурації системи моніторингу. Основна мета цього компонента — продемонструвати функціональність різних типів моніторингу: як агентного, так і безагентного.

3.1 Розгортання системи моніторингу

Для запуску локальної системи моніторингу Zabbix ми використовуємо контейнеризацію Docker. Головна перевага контейнерів полягає у їх здатності створювати узгоджене та ізольоване середовище для вашої системи. Це забезпечує легке розгортання, масштабування та керування вашим середовищем Zabbix, а також дозволяє запускати його на будь-якій платформі, що підтримує Docker.

Ми розгорнули наступні сервіси за допомогою технології Docker Compose: • MariaDB Galera Cluster: Забезпечує відмовостійке та масштабоване сховище даних для Zabbix.

• Zabbix Server: Основний компонент системи моніторингу.

• Zabbix Agent: Відповідає за збір даних на моніторингових хостах. • Zabbix Web: Надає веб-інтерфейс для керування та візуалізації. Всі ці служби

взаємодіють між собою через приватну мережу-міст, названу "zabbix-net".

Функціональні ролі компонентів:

- Zabbix Agent збирає дані про різні показники стану, такі як використання процесора, пам'яті, диска, мережі тощо, і надсилає їх на Zabbix Server.
- Zabbix Server є центральним елементом системи. Він отримує, обробляє та зберігає ці дані в MariaDB для подальшого використання. Zabbix Server покладається на MariaDB для зберігання всієї історії метрик, подій та

конфігураційних даних, а MariaDB забезпечує надійне зберігання та швидкий доступ до цих даних.

- Zabbix Web надає користувацький інтерфейс, через який можна переглядати зібрані дані моніторингу, керувати системою та створювати власні показники.

На Рисунку 3.1.1 показано визначення контейнера "zabbix-server", включаючи налаштування підключення до бази даних, параметри порту та мережі, що використовується контейнером.

Конфігурація контейнера "zabbix-agent" (див. Рисунок 3.1.2) визначає параметри для його підключення до сервера Zabbix та забезпечення взаємодії між ними. Повний код файлу `docker-compose.yml` наведено у Додатку А.

Рисунок 3.1.1 – Визначення контейнерів для сервера Zabbix у файлі dockercompose.yml

Рисунок 3.1.2 – Визначення контейнера для агента Zabbix у файлі dockercompose.yml

Після виконання команди docker compose up -d, усі сервіси, визначені у файлі docker-compose.yml, розгортаються та з'єднуються між собою, формуючи повноцінну та функціональну мережу моніторингу.

Як результат, агент Zabbix починає збір показників про стан системи, які потім відображаються у веб-інтерфейсі Zabbix. (Дивіться Рисунок 3.1.3 для

візуалізації цих даних).

40

Рисунок 3.1.3 – Панель керування показниками збору даних

3.2 Створення програми для автентифікації користувачів

Застосунок для автентифікації користувачів RESTful було написано з використанням мови програмування Python та фреймворку FastAPI. Він має три кінцеві точки реєстраціїу. Вхід та вихід користувача (Рисунок 3.2.1).

Під час реєстрації (Рисунок 3.2.2.) до застосунку надсилається запит, що містить дані користувача. Далі скористайтеся SQL-запитом, щоб створити новий запис користувача в базі даних. Якщо запит успішний, функціяПовертає об'єкт моделі User як JSON-документ.

41

Кінець	**HTTP методи**	**Вебсайт**	**план**

Зареєструватися	Поштова служба	/Користувачі/Реєстрація	{ "Повне ім'я": "Рядок", "Електронна адреса": " user@example.com ", "пароль": "рядок", "Роль": "Користувач", "розташування": "Європа"}
вхід	Поштова служба	/Користувачі/Вхід	{ "Електронна адреса": " user@example.com ", "пароль": "рядок" }
Вийти	Поштова служба	/користувач/вихід	{ "Електронна адреса": " user@example.com ", "пароль": "рядок" }

Рисунок 3.2.1 – Авторизовані кінцеві точки застосунків)

Рисунок 3.2.2 – Функція реєстрації користувача

Коли користувач входить до системи (Рисунок 3.2.3), використовується SQL-запит для перевірки наявності користувача в базі даних. Якщо користувача знайдено, його статус зміниться з Неактивний на Активний (Рисунок 3.2.4.). Вихід з системи такий самий, як і вхід, але статус повертається до неактивного.

Рисунок 3.2.3 – Функція входу користувача

Рисунок 3.2.4. – Функція зміни статусу користувача

Для зберігання інформації про користувачів розгортається контейнер

"mysql" (Рисунок 3.2.6), який містить базу даних MySQL. У базі даних створено таблицю користувачів з такими полями:

Місце	Тип даних
прове Ім'я	varchar(20)
Елект	varchar(40)
ронна пароль	varchar(20)
Роль	varchar(10)
Розташув	varchar(15)
ання Активний	tinyint(1), значення

замовчуванням =

0 Рисунок (3.2.6

_ Схема таблиці

3.2.5. (користувач)

Зберігайте інформацію про місцезнаходження та роль користувача Використайте клас Enumeration, який містить колекцію доступних значень (Додаток В).

Рисунок 3.2.6 – Визначення контейнера mysql

3.3 Моделювання трафіку

Для імітації мережевого трафіку в системі було розроблено Python-скрипт. Цей скрипт з певною періодичністю випадково додає та видаляє користувачів із системи, створюючи динамічне навантаження.

Деталі імітації трафіку:

Скрипт містить функцію mock_traffic (Додаток Е), яка є центральним елементом для генерації трафіку. Вона циклічно викликає функції login_users та logout_users, що імітують активність користувачів.

- Функція login_users симулює вхід заданої кількості користувачів. Вона використовує допоміжну функцію get_active_users для отримання списку неактивних користувачів з бази даних. Для кожного користувача створюється об'єкт UserLogin з електронною поштою та паролем, після чого виконується POST-запит до кінцевої точки системи /user/login. Запит перевіряє код стану відповіді для визначення успішності входу.
- Функція logout_users працює аналогічно, намагаючись деактивувати певну кількість користувачів через кінцеву точку /user/logout. Код використовує випадкові числа, згенеровані з заданого діапазону, для додавання до визначеної кількості користувачів під час операцій входу та виходу. Це дозволяє імітувати різні рівні навантаження та різноманітність трафіку в системі.

3.4 Впровадження моніторингу

Для збору та надсилання метрик на сервер Zabbix було розроблено окремий Python-скрипт (Додаток D).

Через веб-інтерфейс Zabbix були створені спеціальні показники (метрики) для збору та відстеження даних, пов'язаних з активністю користувачів (Рисунок 3.4.1). Прикладами таких метрик є: кількість активних клієнтів системи, активні користувачі з певною роллю та кількість клієнтів з певного регіону.

Рисунок 3.4.1 – Перелік показників

Механізм збору та відправки метрик:

- Скрипт взаємодіє з базою даних, отримуючи необхідні показники за допомогою SQL-запитів.
- Він також містить клас ZabbixMonitoring, який інкапсулює налаштування для моніторингу через Zabbix. Конструктор цього класу встановлює необхідні значення імені хоста для сервера Zabbix та локального агента.
- Метод send_metric у класі ZabbixMonitoring відповідає за відправку метрик на сервер Zabbix. Він використовує клас ZabbixSender з бібліотеки pyzabbix для встановлення з'єднання з агентом Zabbix.
- Цей метод створює об'єкт ZabbixMetric, що містить ім'я хоста Zabbix, ім'я метрики (значення зі списку метрик, див. Додаток B) та саме значення метрики.
- Дані надсилаються на сервер Zabbix із заданою частотою для

кожного показника.

3.5 Результати

В результаті розробки було створено системний компонент, який ефективно демонструє різні можливості моніторингу: як через стандартний агент Zabbix, так і за допомогою модуля pyzabbix.

Для цього система моніторингу Zabbix була розгорнута локально. Стандартний агент Zabbix використовується для збору інформації про поточний стан хоста та її передачі на сервер.

Додатково, розроблено спеціалізований додаток (Рисунок 3.5.1), який збирає показники активності користувачів на основі певних атрибутів та надсилає їх на сервер.

Рисунок 3.5.1 – Схема застосування

Цей додаток складається з трьох основних компонентів: автентифікація користувачів, моделювання трафіку та моніторинг. Компонент моніторингу активується одночасно з автентифікацією користувача (Рисунок 3.5.2).

Рисунок 3.5.2. – Активування авторизованої програми

Для кожного показника встановлено метод у категорії моніторингу

(Рисунок 3.5.3). Кожен визначений метод виконується в окремому потоці, що дозволяє одночасно контролювати кілька частин вашої програми.

Рисунок 3.5.3 – Активуйте авторизовану програму та одночасно ввімкніть моніторинг

Під час моніторингу значення індикаторів передаються на сервер Zabbix та зберігаються в базі даних MariaDB. Результати моніторингу були перевіреніВізуалізуйте графічно за допомогою інтерфейсу користувача Zabbix сітка (Рисунок 3.5.4).

Рисунок 3.5.4 – Результати моніторингу

ВИСНОВКИ

Під час виконання цієї кваліфікаційної роботи було успішно досягнуто поставлених цілей та комплексно виконано низку важливих завдань, спрямованих на розробку та демонстрацію ефективної системи моніторингу корпоративної мережі.

На початковому етапі, після ретельного аналізу вимог до моніторингу сучасної корпоративної мережі, яка включає як центральний кампус, так і географічно розподілені офіси, було сформовано детальне технічне завдання на розробку архітектури системи моніторингу. Ключовою вимогою до цієї архітектури була висока доступність, що гарантує безперебійну роботу системи моніторингу навіть у разі відмови окремих її компонентів. Це забезпечує постійний контроль над станом мережі та мінімізує ризики втрати даних моніторингу.

Для визначення найбільш оптимального рішення було проведено глибокий аналіз і порівняння існуючих прототипів систем моніторингу, таких як Zabbix, Nagios та Prometheus. Кожен із цих інструментів був детально вивчений з точки зору функціональності, масштабованості, складності впровадження та підтримки. За результатами цього всебічного аналізу, компоненти Zabbix були обрані як основна платформа для побудови власної системи моніторингу, завдяки їхній гнучкості, широкому функціоналу та активній спільноті розробників.

На підставі детального аналізу технічних рішень, що дозволяють реалізувати сформульовані вимоги, була розроблена структурна схема власного рішення для моніторингу. Враховуючи децентралізовану структуру сучасних підприємств, що складаються зі штаб-квартири та численних регіональних офісів, особливу увагу було приділено забезпеченню відмовостійкості на рівні штаб-квартири. Тут була реалізована конфігурація резервного копіювання, де два сервери працюють у режимі активного резервування (active-active), забезпечуючи безперервність обробки даних. База даних також функціонує як кластер, що значно підвищує її надійність. Для розподілу навантаження та подальшого підвищення

відмовостійкості, система включає два фронтенди, доступ до яких забезпечується

через балансувальник навантаження. Моніторинг ж регіональних відділень здійснюється за допомогою спеціально налаштованих агентів, що дозволяє збирати дані локально та ефективно передавати їх до центральної системи.

Для практичної демонстрації функціональності агентного моніторингу була розроблена та розгорнута конфігурація тестової системи. Завдяки використанню технології контейнеризації Docker, вдалося швидко та ефективно розгорнути необхідний стек сервісів, включаючи:

MariaDB Galera Cluster: Для високонадійного та масштабованого зберігання даних моніторингу.

Zabbix Server: Центральний компонент, що відповідає за прийом, обробку та зберігання даних.

Zabbix Web: Веб-інтерфейс для візуалізації, управління та аналізу зібраних даних.

Zabbix Agent: Безпосередньо збирає метрики з об'єктів моніторингу та надсилає їх на Zabbix Server, демонструючи механізм агентного збору даних. Таким чином, було чітко продемонстровано повний цикл збору, обробки, зберігання та відображення даних моніторингу за допомогою агентів Zabbix. Окремим важливим етапом була демонстрація можливостей безагентного моніторингу. Для цього було розроблено спеціалізований додаток для автентифікації користувачів, використовуючи сучасну мову програмування Python та швидкий асинхронний фреймворк FastAPI. Цей додаток мав три ключові кінцеві точки: для реєстрації, входу та виходу користувачів із системи, а їхні дані зберігалися в базі даних MySQL. Для імітації реального навантаження було розроблено спеціальний скрипт, що генерував трафік до цих кінцевих точок шляхом надсилання HTTP-запитів на локальний сервер. Паралельно з цим, було створено другий скрипт на Python, який не тільки моніторив та збирав метрики щодо цього згенерованого трафіку (наприклад, час відповіді, кількість успішних/невдалих запитів), але й використовував бібліотеку pyzabbix для безпосередньої відправки

зібраних метрик до Zabbix. Ця частина роботи наочно підтвердила здатність системи здійснювати ефективний моніторинг без необхідності встановлення

агентів на кожному об'єкті, що значно розширює її універсальність та застосовність.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zabbix- A Simpler way of Monitoring [Електронний ресурс] – Режим доступу до ресурсу:

https://medium.com/@shashankrock24/zabbix-a-simpler-way-of monitoringf233f3206c42

2. Nagios Documentation [Електронний ресурс] – Режим доступу до ресурсу:

https://www.nagios.org/documentation/

3. Nagios Tutorial [Електронний ресурс] – Режим доступу до ресурсу:

https://www.javatpoint.com/nagios

4. What is Prometheus? [Електронний ресурс] – Режим доступу до ресурсу:

https://prometheus.io/docs/introduction/overview/

5. Proxies [Електронний ресурс] – Режим доступу до ресурсу:

https://www.zabbix.com/documentation/current/en/manual/distributed_monitori n g/prox ies

6. High availability [Електронний ресурс] – Режим доступу до ресурсу: https://www.zabbix.com/documentation/current/en/manual/concepts/server/ha 7. WHAT'S NEW IN ZABBIX 6.0 LTS BY ARTŪRS LONTONS / ZABBIX SUMMIT ONLINE 2021 [Електронний ресурс] – Режим доступу до ресурсу:

https://noise.getoto.net/2021/12/10/whats-new-in-zabbix-6-0-lts-by-arturs lontonszabbix-summit-online-2021/

8. What is Database Clustering? [Електронний ресурс] – Режим доступу до ресурсу:

https://www.harperdb.io/post/what-is-database-clustering

9. Overview of Galera Cluster [Електронний ресурс] – Режим доступу до ресурсу:

https://galeracluster.com/library/documentation/overview.html

10. Galera Cluster for MariaDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.ionos.com/digitalguide/hosting/technical-matters/mariadb galera-clusters/

ДОДАТКИ

Додаток А

Docker compose файл для розгортання системи моніторингу.

```
Version: '2.1'


#This Docker file is used to set up a Zabbix monitoring environment
#consisting of multiple services: MariaDB, Zabbix Server, Zabbix Agent,  #and Zabbix
Web. The services are connected via a bridge network called 'zabbix-net'  # Define a bridge
network for the containers to communicate with each other  networks: 53abbix-net:
 driver: bridge
 services:
# MariaDB container for database storage mariadb:
 # Connect the container to the 53abbix-net network networks:
  - zabbix-net
 # Use the mariadb-galera image from Docker Hub image:
docker.io/bitnami/mariadb-galera:10.11 # Map container ports to the host machine
ports: - '3306:3306'
  - '4444:4444'
  - '4567:4567'
  - '4568:4568'
 # Persist data in the mariadb_galera_data volume volumes:
  - mariadb_galera_data:/bitnami/mariadb
 environment:
  - MARIADB_ROOT_PASSWORD=12345
  - MARIADB_GALERA_MARIABACKUP_PASSWORD=12345
  - MARIADB_GALERA_CLUSTER_ADDRESS=gcomm://
  - MARIADB_CHARACTER_SET=utf8mb4
  - MARIADB_COLLATE=utf8mb4_bin
 healthcheck:
 test: ['CMD', '/opt/bitnami/scripts/mariadb-galera/healthcheck.sh']  interval: 15s
# how often to perform the health check timeout: 5s # how long to wait  for the health
check to complete retries: 6 # how many times to retry the health  check if it fails



 # Zabbix server container for monitoring 53abbix-server:
 image: 53abbix/53abbix-server-mysql:ubuntu-6.4-latest networks: - 53abbix-
net container_name: 53abbix-server environment:
 # Use "mariadb" as the database server host
 DB_SERVER_HOST: 'mariadb'
 MYSQL_USER: 'root'
 MYSQL_PASSWORD: '12345'
 ENABLE_TIMESCALEDB: true
 ZBX_STARTREPORTWRITERS: 2
 # Map container ports to the host machine ports:
  - «10051:10051»
 # Link the container to the mariadb container links:
  - mariadb
          # Depend on the mariadb container to start up first depends_on: - mariadb
```

```
        # Zabbix agent container for monitoring host metrics 53abbix-agent: image:
53abbix/53abbix-agent:ubuntu-6.4-latest user: root networks:
```



```
          - zabbix-net
         # Link the container to the 54abbix-server container links:
          - zabbix-server
         # Set privileged access mode for allowing resource access privileged: true
environment:
         ZBX_HOSTNAME: Zabbix server ports:
          - «10050:10050»


         # Zabbix web interface container for viewing monitoring data 54abbix-web:
image: 54abbix/54abbix-web-apache-mysql:ubuntu-6.4-latest ports: - "80:8080"
          - «443:8443»
         networks: - 54abbix-net container_name: 54abbix-web environment:
        DB_SERVER_HOST: 'mariadb'
         MYSQL_USER: 'root'
         MYSQL_PASSWORD: '12345'
         ZBX_SERVER_HOST: 'zabbix-server'
         PHP_TZ: Europe/Kiev
         # Set up links between the container and the mariadb and 54abbix-server
containers links:
          - mariadb - 54abbix-server depends_on: - mariadb
          - zabbix-server
         mysql:
         image: mysql:5.7 environment:
         MYSQL_DATABASE: 'user'
         MYSQL_USER: 'mysql'
         MYSQL_PASSWORD: 'root' MYSQL_ROOT_PASSWORD: 'root' ports:  -
          '3307:3306' expose: - '3306' volumes:
          - mysql_data_container:/var/lib/mysql
         volumes: mariadb_galera_data: mysql_data_container:
```


Додаток Б

Python код, що визначає RESTful API з трьома кінцевими точками для обробки запитів на реєстрацію, вхід і вихід користувачів із системи.

```
        """
        This code defines a RESTful API with three routes to handle user signup, login, and
logout requests.
        """ from fastapi import FastAPI, Body, HTTPException, status from
fastapi.openapi.models import Response from app.schema import User, UserLogin from app.utils
import run_query, check_user, set_active
         app = FastAPI()


        # Define a POST route for user signup
        @app.post("/user/signup", tags=["user"], status_code=status.HTTP_200_OK) def
user_sighup(user: User = Body(default=None)): try:
         # SQL query to insert a new user into the database
         sql = """INSERT INTO users (fullname, email, password, role, location)  VALUES
(%s, %s, %s, %s, %s)""" val = (user.fullname, user.email, user.password,
user.role.value, user.location.value) run_query(sql, val, is_update=False)  return
user.json() except Exception as error:
         # If an error occurs during the signup process, return a JSON response with  an
error  message  and  status  code  500  return  Response(content={"error":  str(error)},
media_type="application/json",
status_code=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

```
        # Define a POST route for user login
        @app.post("/user/login", tags=["user"], status_code=status.HTTP_200_OK) def
user_login(user: UserLogin = Body(default=None)):
         # check if the user exists in the system if check_user(user):
         # set the user's status as active set_active(user, True) return  user.json() else:
         # If the user credentials are invalid, raise an HTTPException with a 403
Forbidden status code and error message raise
HTTPException(status_code=status.HTTP_403_FORBIDDEN, detail={"error": "Invalid login
details!"})


        # Define a POST route for user logout
        @app.post("/user/logout", tags=["user"], status_code=status.HTTP_200_OK) def
user_logout(user: UserLogin = Body(default=None)):
           if check_user(user):
         # set the user's status as inactive set_active(user, False)  return
user.json() else: raise
HTTPException(status_code=status.HTTP_403_FORBIDDEN, detail={"error": "Logout failed"})
```


Додаток В

Цей код визначає три окремі класи enum (Role,Location та Metrics) які використовуються для визначення та забезпечення виконання наборів констант із фіксованим набором можливих значень.

```
        """
        This code defines three separate enum classes (Role, Location, and Metrics) which
are used to define and enforce sets of constants with a fixed set of possible values.
""" import enum


        # define Role enum with two possible values class Role(enum.Enum): USER = "user"
        ADMIN = "admin"
        # define Location enum with three possible values class Location(enum.Enum):
        EUROPE = "europe"
         ASIA = "asia"
         NORTH_AMERICA = "north america"
        # define Metrics enum with different metrics that can be used to evaluate the system
class Metrics(enum.Enum):
         TOTAL_NUMBER_OF_ACTIVE_CLIENTS = "total_number_of_active_clients"
         TOTAL_NUMBER_OF_ACTIVE_USERS = "total_number_of_active_users"
         TOTAL_NUMBER_OF_ACTIVE_ADMINS = "total_number_of_active_admins"
         NUMBER_OF_CLIENTS_FROM_EUROPE = "number_of_clients_from_europe"
         NUMBER_OF_CLIENTS_FROM_ASIA = "number_of_clients_from_asia"
           NUMBER_OF_CLIENTS_FROM_NORTH_AMERICA = "number_of_clients_from_north_america"
```


Додаток Г

Цей файл визначає дві моделі Pydantic, User і UserLogin.

Дані моделі використовуються для визначення схеми даних для входу, виходу та реєстрації користувачів.

```
        """
        This file defines two Pydantic models, User and UserLogin. The User and UserLogin
models are used to define the data schema for user information and user login information,
respectively.
        """ from pydantic import BaseModel, Field, EmailStr from app.enums import Location,
Role
```

```
        # define User model to represent the user data class User(BaseModel):
fullname: str = Field(default=None) email: EmailStr = Field(default=None)  password:
str = Field(default=None)
        # role and location fields use the Role and Location enum defined in enums.py
role: Role = Field(default=None) location: Location = Field(default=None)
```

```
        # define UserLogin model to represent the user login data class
UserLogin(BaseModel):
         email: EmailStr = Field(default=None) password: str = Field(default=None)
```


Додаток Г

Даний код імітує трафік у системі авторизації користувачів шляхом виконання дій входу та виходу.

```
        """
        The code provided simulates traffic in a system by performing login and logout
actions for a specified duration.
        """ import requests import datetime from app.schema import UserLogin import time
import random from app.utils import run_query
```

```
        # utility function to get active users def get_active_users(is_active):
query = "SELECT fullname, email, password, role, location FROM users WHERE
is_active=%s" return run_query(query, (is_active,), False)
        # function to simulate traffic def mock_traffic():
         start_time = datetime.datetime.now() end_time = start_time +
datetime.timedelta(minutes=5) count = 1 while datetime.datetime.now() < end_time:
print("...") login_users(count) logout_users(count)  count += random.randint(0, 4)
time.sleep(5)
        # function to simulate login actions for a specified number of users def
login_users(count):
         user_list = get_active_users(is_active=0) if user_list: for i in  range(count +
random.randint(0, 10)): _, email, password, _, _ = user_list[i]  user =
UserLogin(email=email, password=password) response =
requests.post("http://localhost:8001/user/login", user.json()) if  response.status_code
== 200:
         print("login") else:
         print("login error")
```

```
        # function to simulate logout actions for a specified number of users def
logout_users(count):
         user_list = get_active_users(is_active=1)
           if user_list:
         for i in range(count + random.randint(0, 7)): _, email, password,  _, _ =
user_list[i] user = UserLogin(email=email, password=password)  response =
requests.post("http://localhost:8001/user/logout", user.json()) if  response.status_code ==
200:
         print("logout") else:
         print("logout error")
         if __name__ == '__main__':
         mock_traffic()
```


Додаток Д

Сценарій Python, який налаштовує моніторинг для системи з користувачами, розташованими в різних регіонах. Сценарій використовує інструмент моніторингу Zabbix для збору метрик про активних клієнтів, активних користувачів, активних адміністраторів і кількість клієнтів з окремих регіонів. """

```
        This code is a Python script that sets up monitoring for a system with users located
in different regions.
        The script uses the Zabbix monitoring tool to collect metrics about active clients,
active users, active admins, and the number of clients from specific regions.  """ import
datetime import time from pyzabbix import ZabbixSender, ZabbixMetric from  app.utils import
run_query from app.enums import Role, Location, Metrics


        # the total number of clients who  are active def get_login_clients():
                    SELECT COUNT(*) FROM users WHERE
            is_active = 1
         query = ""
         return run_query(query, [], False)
        # the number of clients who are active and have a specific role def

                    SELECT COUNT(*) FROM users WHERE role=%s AND
            is_active = 1
get_login_clients_by_role(role):
         query = ""
         return run_query(query, (role,), False)
      # the number of clients who are active and located in a specific region def
  get_login_clients_by_location(location): query = "SELECT COUNT(*) FROM users WHERE
      location=%s AND is_active = 1" return run_query(query, (location,), False)


        # Define a class for Zabbix monitoring class ZabbixMonitoring:
         # Constructor function to initialize the class with the necessary hostname values
def __init__(self):
         self.zabbix_hostname = 'Zabbix server' self.agent_hostname =  'localhost'


         # Function to send a metric to Zabbix server def send_metric(self,
metric_name: Metrics, metric_value):
         zbx = ZabbixSender(self.agent_hostname) metric =
 [ZabbixMetric(self.zabbix_hostname, metric_name.value, metric_value), ]
print(f"{metric_name.value} {zbx.send(metric)}")


         # Function to collect the total number of active clients and send it as a metric  to
Zabbix server def collect_total_clients(self): # Set the start and end time for  monitoring
start_time = datetime.datetime.now() end_time = start_time +  datetime.timedelta(minutes=6)
while datetime.datetime.now() < end_time:
         print("Monitoring clients...") num_list = get_login_clients()  num_active_clients =
num_list[0][0]
self.send_metric(Metrics.TOTAL_NUMBER_OF_ACTIVE_CLIENTS, num_active_clients)    # Wait for
        20 seconds before monitoring again time.sleep(20)


         # Function to collect the total number of active users and send it as a metric to
Zabbix server def collect_total_users(self):
         start_time = datetime.datetime.now() end_time = start_time +
datetime.timedelta(minutes=6) while datetime.datetime.now() < end_time:
```



```
        print("Monitoring users...") num_list =
get_login_clients_by_role(Role.USER.value) num_active_users = num_list[0][0]
self.send_metric(Metrics.TOTAL_NUMBER_OF_ACTIVE_USERS, num_active_users)
time.sleep(15)
```

```
        # Function to collect the total number of active admins and send it as a metric
to Zabbix server def collect_total_admins(self):
            start_time = datetime.datetime.now() end_time = start_time +
      datetime.timedelta(minutes=6) while datetime.datetime.now() < end_time:
                    print("Monitoring admins...") num_list =
get_login_clients_by_role(Role.ADMIN.value) num_active_admins = num_list[0][0]
self.send_metric(Metrics.TOTAL_NUMBER_OF_ACTIVE_ADMINS, num_active_admins)
time.sleep(15)


        # Function to collect the total number of active clients from Europe and send it
as a metric to Zabbix server def collect_total_clients_from_europe(self):   start_time =
datetime.datetime.now() end_time = start_time +  datetime.timedelta(minutes=6) while
datetime.datetime.now() < end_time:   print("Monitoring europe...") num_list =
get_login_clients_by_location(Location.EUROPE.value) num_clients_from_europe =
num_list[0][0] self.send_metric(Metrics.NUMBER_OF_CLIENTS_FROM_EUROPE,
num_clients_from_europe) time.sleep(15)


        # Function to collect the total number of active clients from Asia and send it as
a metric to Zabbix server def collect_total_clients_from_asia(self): start_time =
datetime.datetime.now() end_time = start_time + datetime.timedelta(minutes=6)  while
datetime.datetime.now() < end_time:
        print("Monitoring asia...") num_list =
get_login_clients_by_location(Location.ASIA.value) num_clients_from_asia =
num_list[0][0] self.send_metric(Metrics.NUMBER_OF_CLIENTS_FROM_ASIA,
num_clients_from_asia) time.sleep(15)


        # Function to collect the total number of active clients from North America and
send it as a metric to Zabbix server def collect_total_clients_from_north_america(self):
start_time = datetime.datetime.now() end_time = start_time +  datetime.timedelta(minutes=6)
while datetime.datetime.now() < end_time:  print("Monitoring north america...") num_list =
get_login_clients_by_location(Location.NORTH_AMERICA.value)
num_clients_from_north_america = num_list[0][0]
self.send_metric(Metrics.NUMBER_OF_CLIENTS_FROM_NORTH_AMERICA,
num_clients_from_north_america) time.sleep(15)


        # Create a list of monitoring methods to be executed monitoring_methods = [
        ZabbixMonitoring().collect_total_clients,
         ZabbixMonitoring().collect_total_users,
         ZabbixMonitoring().collect_total_admins,
         ZabbixMonitoring().collect_total_clients_from_europe,
         ZabbixMonitoring().collect_total_clients_from_asia,
         ZabbixMonitoring().collect_total_clients_from_north_america
        ]
```


Додаток Е

Додаткові функції для застосунку.

```
        import mysql.connector
        from app.schema import UserLogin


        # function to execute a database query and returns the result def run_query(query,
param, is_update):
         cnx = mysql.connector.connect(user='mysql', password='root', host='localhost',
port=3307, database='user') cursor = cnx.cursor() cursor.execute(query, param) if
is_update: cnx.commit() return return cursor.fetchall()


        # function to check if a user exists in the database def check_user(data:
UserLogin):
         query = """SELECT email, password FROM users where email=%s and password=%s"""  val
```

```
= (data.email, data.password) result = run_query(query, val, is_update=False) if  not
result: return False return True


        # function to set the active status of a user in the database def set_active(user:
UserLogin, value: bool):
         query = """UPDATE users SET is_active = %s where email=%s and  password=%s"""
val = (int(value), user.email, user.password) run_query(query, val,  is_update=True)
```

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
Державного некомерційного підприємства
«Державний університет «Київський авіаційний інститут»

ВІДГУК
керівника кваліфікаційної роботи

випускника спеціальності 123 Комп'ютерна інженерія
факультет/відділення «Комп'ютерної і програмна інженерія»
Любов СЕРГАЧОВА
(Ім'я, ПРІЗВИЩЕ)

1. *Кваліфікаційна робота на тему «Структурна схема рішення для моніторингу мережі».*
2. *Метою кваліфікаційої роботи є створення надійної системи моніторингу корпоративної мережі, що охоплює центральний кампус і географічно розподілені філії. Основний акцент було зроблено на забезпеченні високої доступності та відмовостійкості всіх компонентів..*
3. *Кваліфікаційна робота відповідає темі, затвердженій наказом начальника коледжу.*
4. *Кваліфікаційна робота виконана курсантом самостійно.*
5. *Здобувач освіти показав високі вміння працювати з літературними джерелами, аналізувати теоретичний та практичний матеріал, приймати обґрунтовані наукові рішення, застосовувати сучасні комп'ютерні інформаційні технології.*
6. *Любов СЕРГАЧОВА показала достатній рівень дотримання вимог ДСТУ при виконанні дипломної роботи в цілому та оформленні пояснювальної записки.*
7. *Рівень виконаної кваліфікаційної роботи задовільний, відповідає набутих знань, умінь та навичок, вимогам освітньо-кваліфікаційної характеристики фахівця і можливість присвоєння йому кваліфікації фахівця освітнього ступеня «фаховий молодший бакалавр» за спеціальністю 123 «Комп'ютерні системи і мережі» відповідно до Національного класифікатора України «Класифікатор професій ДК 005:2005» та заслуговує на оцінку «відмінно».*

Керівник кваліфікаційної роботи викладач, ктн, доцент
(науковий ступінь, посада)
«___»_____20__ р. _____ *Ірина ВДОВИЧЕНКО*
(підпис) (Ім'я, ПРІЗВИЩЕ)