ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему «Система моніторингу серверів і додатків на основі технологій Prometeus та Grafana»

на здобуття освітнього ступеня бакалавра

зі спеціальності 126 Інформаційні системи та технології

освітньо-професійної програми Інформаційні системи та технології

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

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

Виконав: здобувач вищої освіти гр. ІСД-41
Владислав ДЕМ'ЯНЕНКО
Ім'я, ПРІЗВИЩЕ

Керівник:
науковий ступінь, вчене звання
Ірина СРІБНА, д. т. н., доцент
Ім'я, ПРІЗВИЩЕ

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

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти бакалавр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедрою ІСТ

_____Каміла СТОРЧАК

«____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дем'яненко Владислав Володимирович

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Система моніторингу серверів і додатків на основі технологій Prometeus та Grafana

керівник кваліфікаційної роботи Ірина СРІБНА, д. т. н., доцент,

(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» 02 2025р. № 56

2. Строк подання кваліфікаційної роботи «30» 05 2025 р.

3. Вихідні дані до кваліфікаційної роботи: офіційна документація Prometheus, Grafana, науково-технічна література, дані про існуючі системи моніторингу серверів, результати українських та закордонних наукових досліджень у сфері керування серверами.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз існуючих рішень систем моніторингу серверного обладнання
2. Технічне проєктування системи моніторингу
3. Розробка та інтеграція програмного забезпечення із системами Prometheus і Grafana

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «24» 02 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	01.03.2025 р.	
2	Дослідження технічних аспектів та ефективності існуючих рішень моніторингу серверів	15.03.2025 р.	
3	Вибір технологічного стеку, постановка технічного завдання на розробку	17.03.2025 р.	
4	Проєктування структури, архітектури та користувацького інтерфейсу	4.04.2025 р.	
5	Програмна реалізація та тестування роботи системи	16.05.2025 р.	
6	Розробка демонстраційних матеріалів	26.05.2025 р.	
7	Попередній захист дипломної роботи	27.05.2025 р.	
8	Подання роботи в деканат	30.05.2025 р.	

Здобувач(ка) вищої освіти _____ Владислав ДЕМ'ЯНЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Ірина СРІБНА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 63 стор., 45 рис., 2 табл., 20 джерел.

Мета роботи – розробити систему моніторингу серверів і додатків на основі технологій Prometheus і Grafana, яка забезпечить централізований збір, візуалізацію та аналіз метрик з можливістю генерації сповіщень, підвищуючи рівень контролю за станом IT-інфраструктури.

Об'єкт дослідження – процеси моніторингу серверного обладнання та програмних сервісів у сучасних інформаційних системах.

Предмет дослідження – методи та засоби збору, зберігання, обробки та візуалізації метрик у реальному часі з використанням відкритих систем моніторингу Prometheus і Grafana, а також їх інтеграційні можливості з іншими сервісами.

Короткий зміст роботи: У першому розділі виконано аналіз сучасних систем моніторингу серверного обладнання, детально розглянуто принципи роботи Prometheus, особливості візуалізації Grafana, а також альтернативні рішення на кшталт Zabbix і Nagios. У другому розділі сформульовано технічне завдання, обґрунтовано вибір технологічного стеку, спроєктовано архітектуру системи та визначено компоненти, що використовуються для розгортання (у т.ч. Docker, Exporters, Alertmanager, Loki, Promtail). Третій розділ присвячено реалізації збору й експорту метрик, створенню дашбордів у Grafana, налаштуванню алертингу та перевірці роботи системи в процесі тестування. Окрему увагу приділено результатам функціонального тестування та перспективам масштабування і розвитку рішення.

КЛЮЧОВІ СЛОВА: СИСТЕМА МОНІТОРИНГУ, PROMETHEUS, GRAFANA, METRICS EXPORTER, ДОКЕР, VISUALIZATION, ALERTMANAGER, LOKI, ІНФОРМАЦІЙНА ІНФРАСТРУКТУРА, ЗБІР ДАНИХ, АЛЕРТИНГ.

ABSTRACT

Textual part of the qualification work for obtaining a bachelor's degree: 63 pages, 45 figures, 2 tables, 20 sources.

The goal of the work is to develop a server and application monitoring system based on Prometheus and Grafana technologies, which will provide centralized collection, visualization and analysis of metrics with the ability to generate notifications, increasing the level of control over the state of the IT infrastructure.

The object of the study is the processes of monitoring server equipment and software services in modern information systems.

The subject of the study is methods and means of collecting, storing, processing, and visualizing metrics in real time using open monitoring systems Prometheus and Grafana, as well as their integration capabilities with other services.

Summary of the work: In the first section, the analysis of modern monitoring systems for server equipment is performed, the principles of Prometheus operation, the features of Grafana visualization, as well as alternative solutions such as Zabbix and Nagios are considered in detail. The second section formulates the terms of reference, substantiates the choice of the technology stack, designs the system architecture and defines the components used for deployment (including Docker, Exporters, Alertmanager, Loki, Promtail). The third section is devoted to the implementation of collecting and exporting metrics, creating dashboards in Grafana, setting up alerting, and checking the system during testing. Special attention is paid to the results of functional testing and the prospects for scaling and developing the solution.

KEYWORDS: MONITORING SYSTEM, PROMETHEUS, GRAFANA, METRICS EXPORTER, DOCKER, VISUALIZATION, ALERTMANAGER, LOKI, INFORMATION INFRASTRUCTURE, DATA COLLECTION, ALERTING.

ЗМІСТ

ВСТУП 9
РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ СИСТЕМ МОНІТОРИНГУ СЕРВЕРНОГО ОБЛАДНАННЯ 11
1.1 Огляд сучасних систем моніторингу та сфера їх застосування 11
1.2 Особливості збору, зберігання та аналізу даних у Prometheus 21
1.3 Потужність візуалізації Grafana та її інтеграційні можливості 28
РОЗДІЛ 2. ТЕХНІЧНЕ ПРОЄКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ 35
2.1 Постановка технічного завдання та вибір технологічного стеку 35
2.2 Проєктування архітектури системи моніторингу серверів 38
2.3 Розгортання та налаштування Prometheus і Grafana 41
РОЗДІЛ 3. РОЗРОБКА ТА ІНТЕГРАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІЗ СИСТЕМАМИ PROMETHEUS І GRAFANA 58
3.1 Реалізація збору та експорту даних 58
3.2 Налаштування візуалізації та оповіщень 64
3.3 Результати тестування та перспективи розвитку системи 68
ВИСНОВКИ 70
ПЕРЕЛІК ПОСИЛАНЬ 72
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 74

ВСТУП

У сучасних умовах стрімкого розвитку цифрових технологій та зростання обсягу інформації, що генерується IT-системами, особливого значення набувають системи моніторингу серверів та додатків. Забезпечення безперервного нагляду за станом обчислювальної інфраструктури дозволяє своєчасно виявляти критичні відхилення у роботі, знижувати ризики простоїв, запобігати втратам даних та підвищувати загальну ефективність функціонування систем. У цьому контексті технології Prometheus та Grafana виступають як провідні рішення з відкритим вихідним кодом, які дозволяють будувати масштабовані, адаптивні та інтерактивні інструменти моніторингу з гнучкими можливостями візуалізації та оповіщення.

Мета дослідження полягає у розробці та моделюванні інформаційної системи моніторингу серверів і сервісів на основі технологій Prometheus, Grafana та супутніх компонентів з урахуванням вимог до продуктивності, масштабованості, інтеграції з інфраструктурою та автоматизованого керування інцидентами.

Завдання дослідження включають:

- проведення аналітичного огляду сучасних систем моніторингу;
- вибір архітектурного підходу до побудови системи збору та візуалізації даних;
- обґрунтування вибору технологічного стеку;
- реалізацію модулів збору метрик, логів і створення алертів;
- розгортання середовища Prometheus+Grafana з використанням Docker;
- проведення функціонального тестування створеного рішення;
- визначення перспектив розширення системи.

Об’єкт дослідження — процеси моніторингу та оцінки працездатності серверного та прикладного середовища з використанням інструментів візуалізації.

Предмет дослідження — архітектура, компоненти та технічні рішення для збору, агрегації, зберігання, візуалізації метрик та логів у системах реального часу.

Методи дослідження: теоретичний аналіз наукових і технічних джерел, практичне проєктування інформаційних систем, моделювання інфраструктури

моніторингу в контейнеризованому середовищі, тестування працездатності системи.

Основні результати дослідження були апробовані на VII Міжнародній науково-технічній конференції «Сучасний стан та перспективи розвитку IoT» у вигляді тез доповіді на тему «Інтелектуальні інформаційні системи для моніторингу, управління та підтримки користувачів».

Наукова новизна полягає в інтеграції відкритих систем моніторингу в єдину платформу з підтримкою збору, візуалізації, оповіщення та логування у реальному часі на основі сучасного технологічного стеку.

Практична значущість полягає в можливості адаптації запропонованого рішення до широкого спектра прикладних IT-систем з мінімальними витратами на інфраструктуру, завдяки контейнеризації, автоматизації розгортання та підтримці масштабування.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ СИСТЕМ МОНІТОРИНГУ СЕРВЕРНОГО ОБЛАДНАННЯ

1.1 Огляд сучасних систем моніторингу та сфера їх застосування

У сучасних умовах стрімкої цифрової трансформації та переходу підприємств до гібридних і хмарних інфраструктур, забезпечення стабільної, безперервної та захищеної роботи серверного обладнання та прикладного програмного забезпечення набуває особливої актуальності. Зростання залежності бізнес-процесів, державних сервісів та повсякденного життя від інформаційних систем висуває жорсткі вимоги до їх надійності, швидкості реакції на інциденти та прогнозованості навантаження. Саме тому системи моніторингу серверів і додатків стали критично важливими елементами сучасної IT-інфраструктури.

Ринок моніторингових рішень демонструє сталу динаміку зростання, що зумовлено підвищеним попитом на інструменти оперативного контролю, аналізу та візуалізації технічних параметрів у реальному часі. Згідно з глобальним звітом The Business Research Company, розмір ринку моніторингу становив $70,75 млрд у 2024 році й очікується, що до 2029 року він сягне $87,22 млрд зі середньорічним темпом зростання (CAGR) на рівні 4,2% [1]. Такі показники свідчать про високу затребуваність у надійних інструментах, здатних забезпечити комплексну оцінку продуктивності й працездатності різнорідних IT-систем, що активно впроваджуються як у приватному, так і в державному секторах. Ілюстрація динаміки ринку представлена на рис. 1.1.

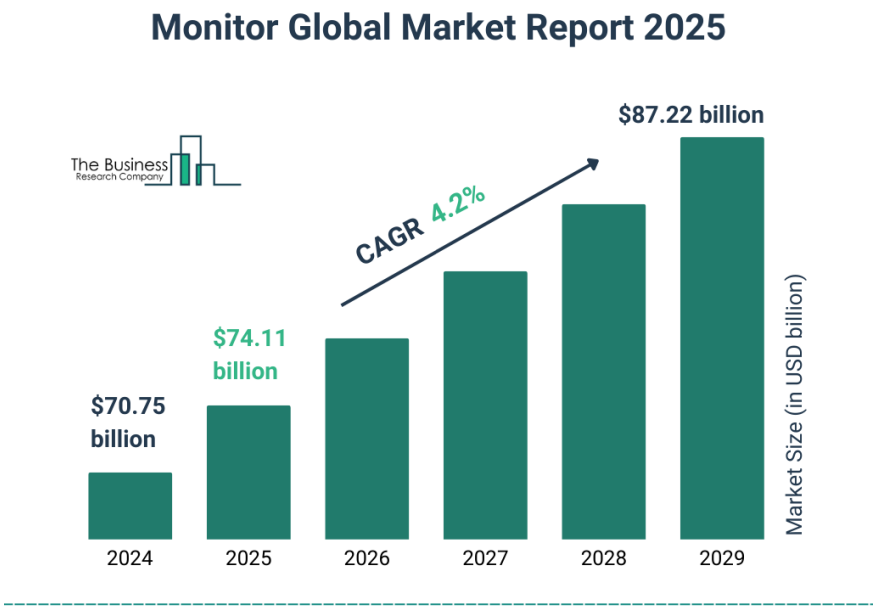


Рис. 1.1 Прогноз динаміки світового ринку систем моніторингу до 2029 року за даними The Business Research Company [1]

Системи моніторингу є багатокомпонентними програмно-апаратними рішеннями, призначеними для збору, обробки, зберігання та аналітики даних про стан технічних і програмних ресурсів. Вони дозволяють здійснювати постійне спостереження за ключовими показниками ефективності (KPI), зокрема завантаженням процесорів, споживанням пам’яті, активністю дискових підсистем, станом мережевих інтерфейсів, рівнем доступності вебсервісів тощо. Крім цього, розширені системи моніторингу підтримують централізований збір логів, трасування запитів, аналітику журналів безпеки та оповіщення у разі виявлення аномалій або збоїв.

До найпоширеніших платформ, які сьогодні використовуються в ролі систем моніторингу, належать Zabbix, Nagios, Datadog, Dynatrace, Splunk, Prometheus у поєднанні з Grafana, а також низка комерційних рішень із підтримкою машинного навчання та автоматичного реагування. Обираючи конкретну систему, IT-спеціалісти орієнтуються на такі критерії, як відкритість вихідного коду, підтримка масштабованості, можливість інтеграції з контейнеризованими середовищами

(Docker, Kubernetes), адаптивність до хмарних технологій і простота кастомізації інтерфейсів для візуалізації даних.

Особливої актуальності набувають рішення, здатні забезпечити високий рівень автономності, тобто з мінімальним втручанням оператора генерувати аналітичні висновки та своєчасно реагувати на інциденти. Наприклад, комбінація Prometheus як системи збору метрик з Grafana як платформи для їх гнучкої візуалізації дозволяє будувати інтерактивні дашборди, що надають повну картину стану IT-середовища. Ці системи підтримують інструменти агрегування, масштабування, розширення алертів і створення складних запитів на мові PromQL, що робить їх ефективними для середніх і великих підприємств.

Системи моніторингу в сучасному інформаційному середовищі є не просто інструментами діагностики, а стратегічними складовими забезпечення стійкості, безпеки та ефективності IT-інфраструктур. Їх актуальність невпинно зростає в умовах розвитку хмарних технологій, інтернету речей (IoT) та загальної тенденції до автоматизації процесів обслуговування серверних потужностей.

Одним із найбільш усталених та широко впроваджених рішень для моніторингу серверів і мережевого обладнання є система Zabbix. Це відкрите програмне забезпечення корпоративного рівня, яке забезпечує повний цикл спостереження за IT-інфраструктурою в режимі реального часу: від збору та зберігання метрик до візуалізації, аналізу й генерації повідомлень про аномальні події. Zabbix підтримує як централізоване, так і розподілене розгортання, надаючи гнучкість у масштабуванні як для малих компаній, так і для великих дата-центрів.

Архітектура Zabbix є модульною та складається з кількох основних компонентів: Zabbix Server, Agent, Proxy, а також вебінтерфейсу адміністратора. Сервер виконує роль центрального елементу системи — він зберігає всю інформацію про конфігурацію, об'єкти моніторингу, отримані дані та сповіщення. Саме сервер здійснює аналіз отриманих метрик, застосування тригерів (умов логічного реагування), формування подій та виконання дій у відповідь (надсилання повідомлень, виконання скриптів тощо) [2].

Zabbix Proxy є проміжною ланкою між агентами й сервером, що дає змогу збирати дані в розподіленому середовищі з подальшим їх буферизованим передаванням до центрального сервера. Це особливо корисно в умовах зниженого доступу до мережі або при моніторингу великих інфраструктур, де оптимізація навантаження на головний сервер є критично важливою [3].

Zabbix Agent — це спеціалізований демон, який запускається на цільових вузлах і здійснює збір статистики з операційної системи (завантаженість CPU, використання RAM, активність дисків, мережева активність) та з прикладного програмного забезпечення. Агент працює як у пасивному (очікування запитів від сервера), так і в активному режимі (ініціює передачу даних самостійно). Для спрощення підключення нових вузлів і скорочення часу конфігурації Zabbix підтримує автоматичне виявлення обладнання та сервісів, а також використовує шаблони для типових служб (наприклад, HTTP, SSH, FTP, DBMS) [2].

Уся інформація представлена в зручному вебінтерфейсі, який забезпечує доступ до даних про поточний стан системи, історії подій, журналів безпеки, налаштувань сповіщень тощо. Інтерфейс підтримує візуалізацію у вигляді графіків, таблиць, дашбордів, а також дозволяє гнучко фільтрувати інформацію за сервісами, рівнями загроз і часовими рамками. Візуальний вигляд архітектури системи наведено на рис. 1.2.

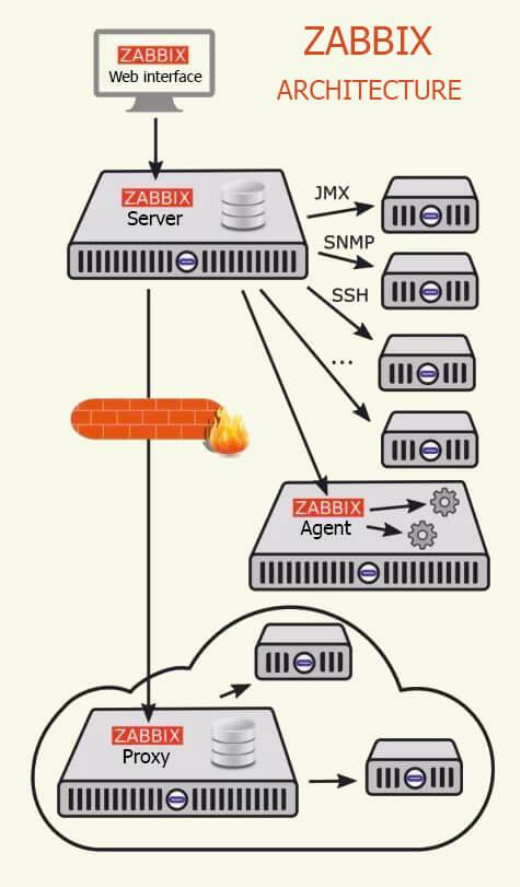


Рис. 1.2 Архітектура системи моніторингу Zabbix [3]

Механізм логічного реагування у Zabbix реалізовано через тригери — це вирази, що порівнюють значення метрик з граничними параметрами. Якщо умова тригера виконується, генерується подія відповідного рівня критичності, а далі — виконується визначена дія (action): наприклад, надсилання email, Slack-повідомлення, запуск зовнішньої команди чи створення тікету. Рівні загроз мають колірне кодування: від “Not classified” до “Disaster” [3].

Zabbix також підтримує збір логів, SNMP-моніторинг мережевих пристроїв, опитування через JMX, Telnet, IPMI та інші протоколи, що робить його універсальним рішенням для змішаних середовищ. Крім цього, через API та webhook можлива інтеграція з зовнішніми інструментами, системами обліку, сервісами хмарної інфраструктури тощо.

У технічному плані Zabbix Server і Proxy працюють під Linux, тоді як агент доступний для більшості платформ, включно з Windows. Сервіс підтримує бази даних MySQL, PostgreSQL, Oracle, а вебінтерфейс реалізований на PHP. Завдяки

відкритому коду, підтримці масштабування, регулярним оновленням та наявності LTS-релізів (наприклад, Zabbix 5.0), Zabbix широко використовується в телекомунікаціях, банківському секторі, державному управлінні та промисловості [2], [3].

Система Nagios є однією з перших і найпоширеніших платформ відкритого коду для моніторингу серверного обладнання, мережевих ресурсів, додатків і служб. Завдяки своїй надійності, широким можливостям кастомізації та підтримці великої кількості плагінів, вона зберігає актуальність навіть у сучасних умовах кластеризованих і хмарних обчислень. Nagios використовує агентно-орієнтовану архітектуру клієнт–сервер, де центральний сервер обробляє інформацію, яку надсилають плагіни з підлеглих вузлів [4].

Ключовими компонентами системи є: основний процес (Nagios core logic), файл статусу (Status File), файл зовнішніх команд (External Command File), вебінтерфейс, а також набір плагінів для виконання перевірок. Основна логіка системи забезпечує координацію між конфігураційними файлами, отриманими метриками та операціями, що виконуються у відповідь на інциденти. Архітектура Nagios дає змогу точно визначити стан хостів або служб через плагіни, які ініціюються як за розкладом, так і за подіями [4]. Візуальна структура архітектури подана на рис. 1.3.

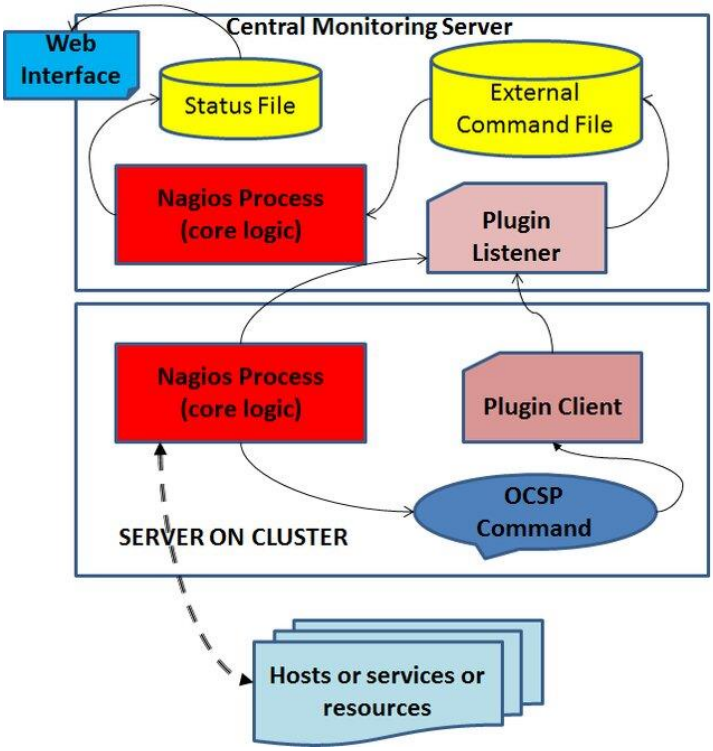


Рис. 1.3 Архітектура системи моніторингу Nagios у кластеризованому середовищі [5]

Особливістю Nagios є її підтримка широкого спектра плагінів, що дозволяє здійснювати моніторинг системних ресурсів (CPU, RAM, диск), служб (HTTP, FTP, SMTP, SSH) та спеціалізованих протоколів (SNMP, NRPE, NSClient++, WMI). Плагіни працюють як незалежні клієнтські утиліти, що виконують конкретні команди перевірки й повертають результати до основного процесу. Такий підхід забезпечує високу гнучкість і модульність у налаштуванні перевірок [4].

Файл статусу використовується як точка синхронізації між вебінтерфейсом і ядром системи. Саме в ньому зберігається поточна інформація про стан усіх перевірених об’єктів. Через зовнішній командний файл адміністратор або автоматизована система може взаємодіяти з ядром, надсилаючи команди для перезапуску перевірок, зміни конфігурацій або ініціювання подій. Цей механізм дозволяє будувати сценарії активного реагування на інциденти або адаптивного моніторингу [4].

У статті Prasad A. [5] наведено вдосконалену концепцію використання Nagios у хмарних середовищах у рамках архітектури CMaaS (Centralized Monitoring as a

Service). Запропонована модель передбачає використання Nagios як централізованого ядра моніторингу, здатного обробляти не лише стандартні метрики якості обслуговування (QoS), а й деталізовані журнали доступу користувачів до хмарних ресурсів. Така розширена функціональність актуальна в умовах, коли хмарні сервіси надаються багатьом внутрішнім користувачам, і вимагається аудит дій кожного з них.

Згідно з CMaaS, плагіни Nagios працюють у двох режимах: часовому (time-step), де метрики надходять періодично, і подійному (event-driven), де ініціація передачі даних відбувається за фактом доступу до ресурсу. Така подвійна модель забезпечує глибоке логування та контроль не лише над технічними характеристиками, а й над поведінкою користувачів у системі [5].

З технічного боку, Nagios підтримує масштабування в кластерах і може бути інтегрований із іншими системами через API або плагін-інтерфейси. Адміністратор може створити власні сценарії перевірок, підключити зовнішні команди (OCSP — Out-of-band Command Scripts), реалізувати керовані реакції на події, а також синхронізувати дані журналів із зовнішніми системами моніторингу або аудитними платформами.

Завдяки відкритому коду, широкій підтримці спільноти, простоті інтеграції в середовища з підвищеними вимогами до нагляду та контролю, Nagios залишається одним з найбільш гнучких інструментів для побудови кастомізованих систем моніторингу, включно з критичними хмарними й гібридними середовищами.

Zabbix та Nagios є одними з найпопулярніших рішень з відкритим вихідним кодом у сфері контролю за станом серверного обладнання, мережевих пристроїв та прикладних сервісів. Обидві системи мають потужну функціональність, однак реалізовані за різними архітектурними підходами та мають відмінності у сфері конфігурації, масштабування, візуалізації даних та інтеграції в складні IT-інфраструктури [6].

Zabbix є централізованою системою моніторингу, побудованою на принципах інтеграції всіх компонентів у єдину екосистему. Вона підтримує власну базу даних, API для конфігураційних операцій, високорівневу абстракцію візуалізації

(дашборди, карти, графіки) та механізми автоматичного виявлення хостів. Архітектура Zabbix передбачає наявність серверного ядра, проксі-компонентів для розподіленого моніторингу, агентів на вузлах та вебінтерфейсу для адміністрування. Така модель дозволяє забезпечити гнучку масштабованість без необхідності складної взаємодії між численними конфігураційними файлами. Крім того, Zabbix підтримує інтеграцію з Prometheus, SNMP, IPMI, JMX та іншими протоколами збору даних, а також має повноцінний механізм створення тригерів і алертів на основі логічних умов.

На відміну від цього, Nagios реалізує класичну агентно-орієнтовану модель із розділенням ядра, плагінів, конфігураційних файлів і зовнішніх компонентів візуалізації. Основний процес Nagios Core виконує планування перевірок, опрацювання результатів та генерацію подій. Плагіни, що запускаються відповідно до визначеного графіку, здійснюють перевірку доступності та продуктивності сервісів, після чого результати передаються до ядра системи. Конфігурація Nagios виконується вручну через текстові файли, що ускладнює її розгортання в середовищах із великою кількістю хостів. Хоча для спрощення адміністрування існують інтерфейсні надбудови (наприклад, Nagios XI), вони, як правило, є комерційними і мають обмежену функціональність у безкоштовних версіях.

У технічному порівнянні варто відзначити, що Zabbix підтримує вищий ступінь автоматизації завдяки вбудованим шаблонам та механізмам автоматичного виявлення, тоді як у Nagios автоматизація потребує сторонніх скриптів або модулів. Важливою перевагою Zabbix є наявність єдиного інтерфейсу для конфігурації, візуалізації та аналітики, що реалізовано у вигляді інтерактивного вебінтерфейсу з можливістю побудови кастомізованих панелей спостереження. Натомість, у Nagios для досягнення аналогічного рівня інтерактивності часто використовуються сторонні рішення, такі як Grafana або Nconf, що підвищує складність розгортання.

У питаннях масштабованості Zabbix надає готові рішення для централізованого збору метрик у розподілених системах, використовуючи проксі-сервери, які зменшують навантаження на центральне ядро та забезпечують буферизацію в умовах нестабільного з'єднання. Nagios, у свою чергу, дозволяє

реалізовувати масштабування через використання модулів NSCA або реалізацію кластерних конфігурацій, що вимагає детального опрацювання з боку адміністратора та має більший рівень технічної складності.

Щодо механізмів оповіщення, Zabbix дозволяє створювати складні правила ескалації інцидентів із вбудованими інструментами для сповіщення через email, SMS, webhook або інші зовнішні сервіси. Всі алерти формуються на основі логічних умов тригерів, із можливістю параметризації часу, частоти та рівня критичності. Nagios також підтримує потужні сценарії алертів, однак їхня реалізація вимагає конфігурування численних опцій вручну через скрипти та зовнішні плагіни.

Порівняльний аналіз ключових параметрів систем представлено у табл. 1.1.

Таблиця 1.1

Порівняння вимірюваних технічних характеристик Zabbix і Nagios

Показник	**Zabbix**	**Nagios Core**
Середній час встановлення (год)	1,5	4,0
Кількість шаблонів у базовій версії	100+	~20
Навантаження на CPU при моніторингу 1000 хостів (%)	15–25	30–50
Затримка доставки алерту (сек)	1–3	5–10
Середній час конфігурації одного вузла (хв)	2–3	8–10
Підтримка пристроїв «з коробки»	5000+	1000+ (через плагіни)
Інтерфейс аналітики (оцінка 1–10)	9	5
Рівень автоматизації (оцінка 1–10)	9	4
Витрати на підтримку (людино-год/міс)	6–10	15–20
Кількість активних оновлень/рік	10+	2–3

Аналіз таблиці 1.1 демонструє перевагу системи Zabbix за більшістю критично важливих технічних характеристик. Зокрема, Zabbix потребує значно менше часу на встановлення (1,5 години проти 4 годин у Nagios Core) та конфігурацію вузлів, а також показує нижче навантаження на процесор при масштабованому моніторингу (15–25% проти 30–50%). Крім того, затримка доставки алертів у Zabbix не перевищує 3 секунд, тоді як у Nagios вона може сягати 10 секунд, що суттєво впливає на швидкість реагування системи.

Також слід відзначити вищий рівень автоматизації та аналітики у Zabbix, що отримав оцінку 9 із 10 проти 4–5 у Nagios. Значна кількість вбудованих шаблонів (понад 100 у Zabbix) спрощує інтеграцію нових пристроїв і сервісів, тоді як Nagios вимагає ручної роботи або сторонніх плагінів. Сукупно ці показники свідчать про ефективнішу експлуатаційну придатність Zabbix у сучасних динамічних середовищах.

Zabbix є більш інтегрованим, масштабованим і зручним для впровадження рішенням, орієнтованим на централізовану обробку даних і автоматизацію процесів моніторингу. Натомість Nagios зберігає перевагу в гнучкості налаштувань та можливості використання в особливо специфічних середовищах, де необхідний повний контроль над усіма процесами конфігурації. Вибір між цими платформами має ґрунтуватися на балансі між необхідним рівнем кастомізації, ресурсами для адміністрування та вимогами до масштабованості і швидкості розгортання.

1.2 Особливості збору, зберігання та аналізу даних у Prometheus

Prometheus є потужною системою моніторингу та зберігання часових рядів (time series), розробленою спільнотою SoundCloud і нині підтримуваною під егідою Cloud Native Computing Foundation. Архітектура Prometheus орієнтована на високоефективний, масштабований збір метрик з інфраструктур різного рівня складності — від окремих серверів до хмарних і контейнеризованих середовищ, таких як Kubernetes. Його основною особливістю є використання моделі pull — тобто Prometheus самостійно опитує визначені джерела (експортери) з певною

періодичністю, замість того, щоб покладатися на push-модель з боку агента, як у класичних рішеннях типу Nagios або Zabbix [7].

Ключовим компонентом системи є Prometheus Server, що здійснює опитування метрик за HTTP-протоколом, використовуючи вбудований планувальник задач. Результати зберігаються у власному високопродуктивному TSDB (time-series database), спеціально оптимізованому для роботи з часовими мітками, мітками (labels) та ключами, що забезпечують багатовимірне моделювання даних. Кожна метрика в Prometheus описується парою ключ–значення з додатковими label-атрибутами, що дозволяє ефективно фільтрувати, агрегувати та комбінувати дані з різних джерел. Завдяки цій структурі Prometheus підтримує понад мільйон записів у секунду навіть на стандартному обладнанні, зберігаючи прийнятний рівень навантаження на CPU та диск.

Архітектура системи Prometheus побудована за принципом модульності та сервісної децентралізації, що забезпечує гнучкість, масштабованість і стійкість у розподілених IT-інфраструктурах. На рисунку 1.4 представлено логічну структуру взаємодії основних компонентів Prometheus із джерелами метрик, підсистемами обробки та візуалізації даних [7].

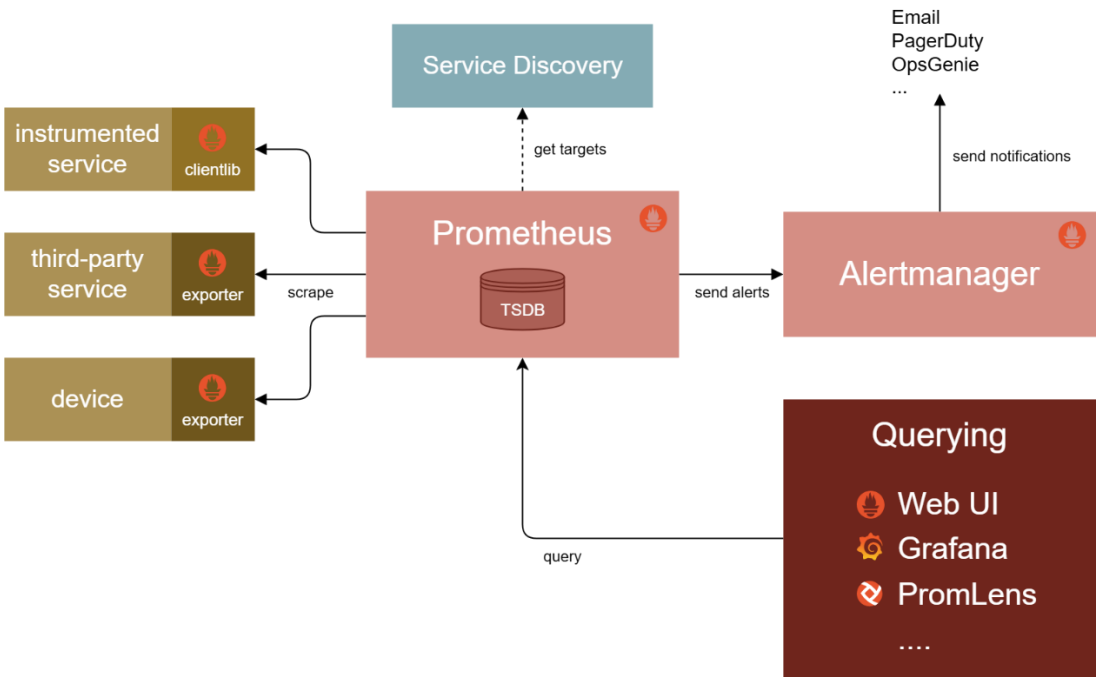


Рис. 1.4 Архітектура системи моніторингу на основі Prometheus [7]

Центральним компонентом виступає Prometheus Server, який виконує роль планувальника опитувань джерел метрик, зберігача даних і виконавця запитів. Його ядро містить вбудовану базу часових рядів TSDB (Time Series Database), що оптимізована для високошвидкісного запису та агрегації великого обсягу метричних даних. Prometheus працює за моделлю «pull», періодично надсилаючи HTTP-запити до визначених endpoint-ів, що підтримуються експортерами або безпосередньо інструментованими сервісами.

На боці джерел даних виділяються три типи постачальників метрик:

- інструментовані сервіси — програми, які інтегрують клієнтську бібліотеку Prometheus і самостійно надають метрики у необхідному форматі;
- сторонні сервіси — зовнішні системи, що не мають вбудованої підтримки Prometheus, але моніторяться за допомогою спеціалізованих експортерів (наприклад, node_exporter, blackbox_exporter тощо);
- фізичні пристрої — мережеве або серверне обладнання, що також опитується через експортери або SNMP-шлюзи.

Важливим елементом архітектури є Service Discovery — підсистема автоматичного виявлення нових вузлів, сервісів або контейнерів, що дозволяє динамічно оновлювати цілі моніторингу без ручного втручання. Підтримується інтеграція з Kubernetes, Consul, EC2, Docker Swarm, а також статичне конфігурування через файли.

Після збору метрик вони зберігаються у базі TSDB, де кожен часовий ряд визначається унікальною комбінацією назви метрики та набору label-пар. Prometheus підтримує ротацію, компресію та збереження даних у вигляді сегментованих блоків із вказаними часовими індексами, що оптимізує доступ до великих обсягів історичної інформації.

Окремим підкомпонентом є Alertmanager — сервіс для обробки та маршрутизації алертів, які формуються на основі логічних виразів у PromQL. Alertmanager підтримує згортання дубльованих подій (grouping), глушіння повідомлень (silencing), визначення політик ескалації, шаблонізацію повідомлень і

інтеграцію з зовнішніми каналами комунікації (Slack, email, Telegram, Opsgenie тощо).

Компонент Querying відповідає за взаємодію з кінцевим користувачем через мови запитів і інструменти візуалізації. Сюди входять: вбудований вебінтерфейс Prometheus (PromUI), сторонні панелі візуалізації (наприклад, Grafana), а також PromLens — потужний редактор PromQL-запитів з валідацією та підсвічуванням синтаксису. Кожен з цих інструментів працює безпосередньо з API Prometheus, отримуючи агрегаційні або фільтровані метрики з TSDB у форматі JSON або text/plain.

Така архітектура дозволяє Prometheus ефективно масштабуватися як у горизонтальному, так і у вертикальному напрямках, підтримуючи при цьому високу щільність збору метрик (до мільйонів точок у хвилину), гнучке розмежування відповідальностей між компонентами та відмовостійкість при короткочасних збоїв у мережі чи вузлах системи.

Для збору даних Prometheus покладається на концепцію експортерів — невеликі HTTP-сервіси, що розгортаються на цільових вузлах і надають структуровані метрики у форматі тексту. Серед найпопулярніших експортерів — Node Exporter (для збору метрик операційної системи), cAdvisor (для Docker-контейнерів), mysqld_exporter (для моніторингу MySQL), а також чорні box-експортери для перевірки доступності HTTP, DNS або TCP. Ці експортери не потребують складної конфігурації й підтримують багатопоточне обслуговування запитів від Prometheus, що суттєво спрощує розгортання великомасштабних систем моніторингу.

Особливістю Prometheus є мова запитів PromQL (Prometheus Query Language), яка дозволяє здійснювати складні обчислення та агрегацію метрик у реальному часі. PromQL підтримує операції над часовими рядами, фільтрацію за мітками, математичні та логічні вирази, а також функції агрегування за часовими інтервалами. Наприклад, за допомогою PromQL можна побудувати запит на середнє навантаження CPU за останні 5 хвилин для певного кластеру серверів або виявити зростання споживання пам’яті протягом доби з детальною розбивкою по сервісах.

Такий підхід дозволяє не лише аналізувати поточний стан системи, а й реалізувати автоматичні правила алертів, що реагують на аномальні патерни поведінки [7].

Візуалізація даних в Prometheus здійснюється або через вбудований вебінтерфейс, або в комбінації з Grafana, яка забезпечує гнучке й інтуїтивне налаштування панелей моніторингу. Через стандартну інтеграцію Grafana з Prometheus можна швидко створювати дашборди для моніторингу навантаження на сервери, затримок мережі, часу відповіді сервісів, використання диску тощо. Крім того, Prometheus має вбудований механізм алертів (Alertmanager), який підтримує маршрутизацію повідомлень до зовнішніх систем, зокрема Slack, Telegram, Email, PagerDuty, Opsgenie, або виконання HTTP-запитів (webhook).

На рисунку 1.5 представлено детальніший варіант архітектури Prometheus, що ілюструє повний життєвий цикл метрик — від їх збору на рівні додатків, баз даних, контейнерів або хостів до подальшої обробки, зберігання, оповіщення й візуалізації [8].

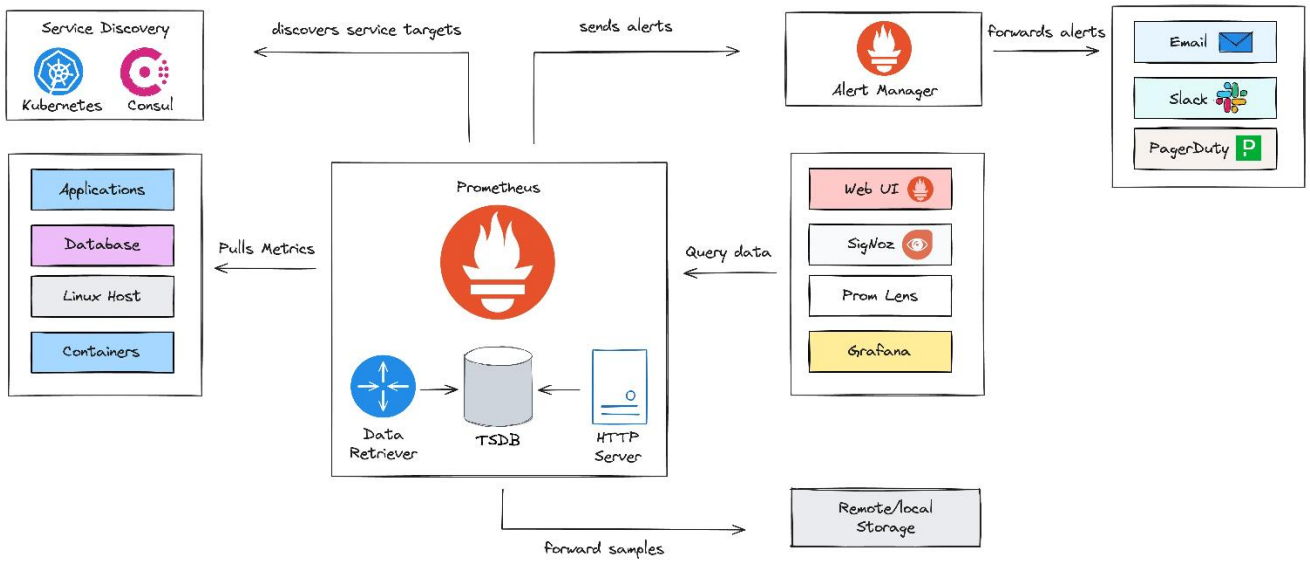


Рис. 1.5 Життєвий цикл метрик у Prometheus: від збору до оповіщення [8]

Prometheus періодично виконує HTTP-запити до визначених джерел (експортерів), після чого отримані метрики надходять до компоненту Data Retriever, який відповідає за зчитування, індексацію й попередню фільтрацію даних перед збереженням у TSDB. Зібрані метрики можуть зберігатися локально або

експортуватися до віддалених систем зберігання для довготривалої архівації через механізм remote write.

Завдяки інтеграції з компонентами Service Discovery (зокрема Kubernetes та Consul), Prometheus може автоматично виявляти нові цілі моніторингу без ручного конфігурування. Після зберігання даних користувачі можуть взаємодіяти з ними через інтерфейси запитів — вбудовану Web UI, редактори PromLens і SigNoz, а також панелі Grafana. Окремий модуль Alert Manager відповідає за формування алертів на основі PromQL-виразів і надсилає повідомлення до таких систем, як Slack, Email чи PagerDuty, відповідно до заданої політики маршрутизації подій. Таким чином, архітектура Prometheus забезпечує повний цикл: «збір → зберігання → аналітика → сповіщення» у межах одного технічного стека [8].

Архітектурна гнучкість Prometheus дозволяє реалізовувати як централізовані, так і федеративні моделі моніторингу, коли кілька екземплярів Prometheus можуть бути об'єднані у єдину систему через механізм зовнішніх джерел (remote_read/remote_write). Це забезпечує масштабування як у горизонтальному, так і у вертикальному напрямках без істотних втрат продуктивності. В умовах хмарних середовищ Prometheus успішно використовується як основа для створення власних сервісів моніторингу завдяки підтримці auto-discovery (виявлення нових цільових вузлів через Kubernetes API, Consul або файлові конфігурації).

В основі архітектури Prometheus лежить концепція зберігання та обробки часових рядів — наборів пар значення та часу, які генеруються у результаті моніторингу метрик із численних джерел. Кожна метрика у Prometheus ідентифікується не лише за назвою, але й за набором label-атрибутів, які утворюють багатовимірний простір даних. Завдяки цьому забезпечується гнучка класифікація та агрегація інформації за такими параметрами, як тип сервісу, IP-адреса вузла, метод запиту або статус відповіді.

На рисунку 1.6 представлено приклад метрики http_requests_total, що містить набір label-пар (наприклад, job="api-server", instance="10.0.0.1:443", method="GET"), а також відповідні вибірки значень, прив’язаних до часових міток.

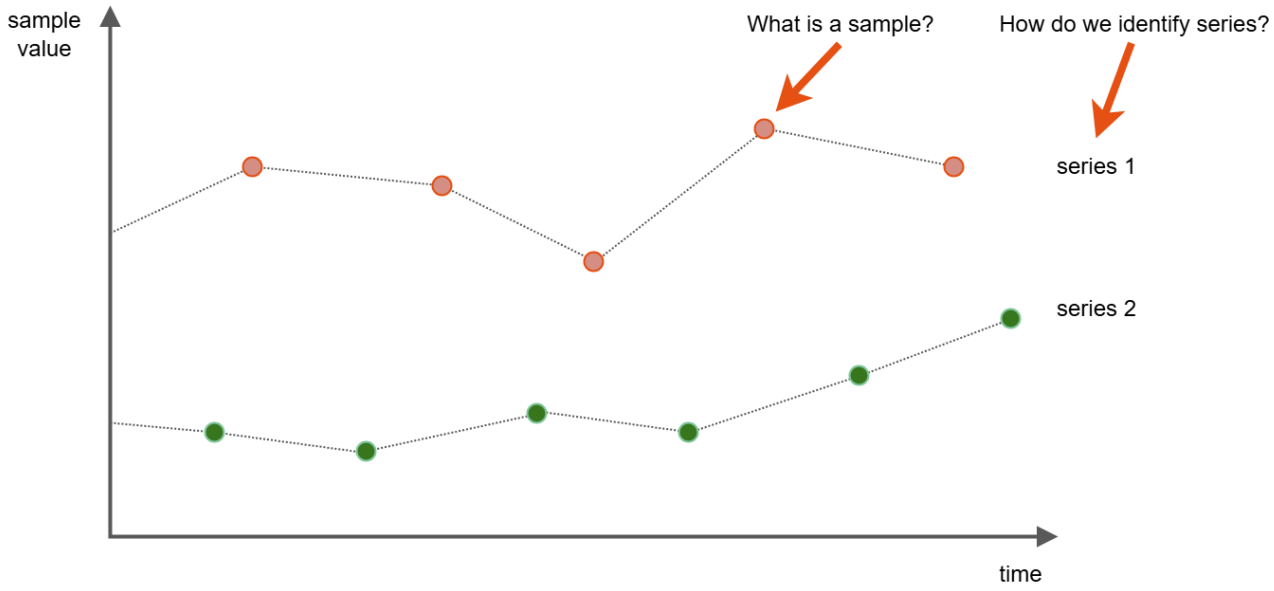


Рис. 1.6 Структура метрики Prometheus: назва, лейбли та часові вибірки [9]

Ця структура дозволяє Prometheus обробляти мільйони часових рядів одночасно з мінімальним споживанням ресурсів, зберігаючи як поточний стан системи, так і історичну інформацію з високою точністю [9].

Завдяки такій моделі даних, Prometheus дозволяє реалізувати складні аналітичні сценарії через запити на мові PromQL. Наприклад, можна виконати агрегацію значень HTTP-запитів для певного сервісу за останні 10 хвилин, фільтрувати метрики за значенням лейблів або побудувати прогноз тренду. Запити PromQL можуть використовуватись не лише для побудови графіків, але й для створення логіки алертів.

На рисунку 1.7 наочно продемонстровано, як Prometheus за допомогою логіки обробки часових рядів виявляє аномалії або порушення заданих умов.

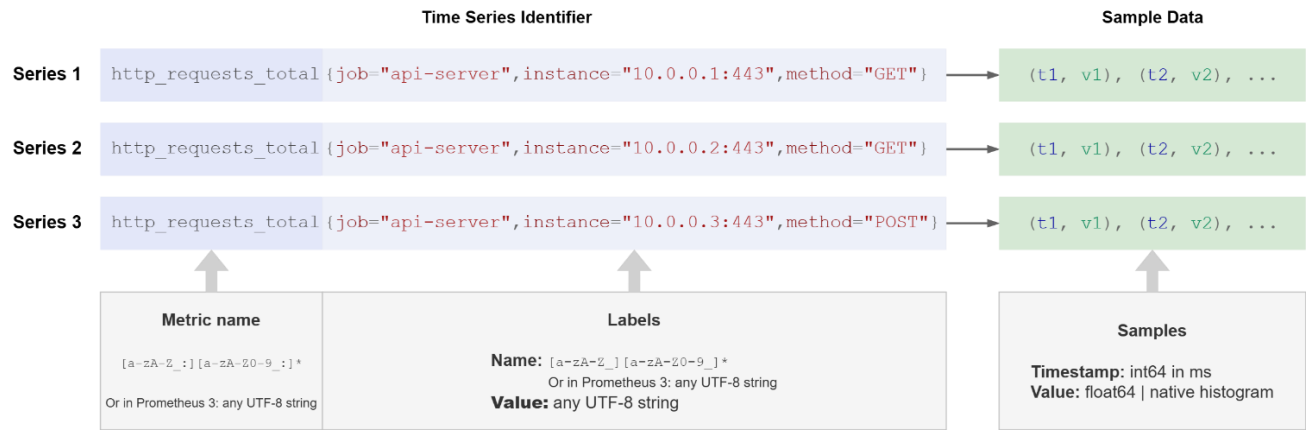


Рис. 1.7 Виявлення аномалій на основі часових рядів у Prometheus [9]

Наприклад, якщо значення певної метрики виходить за допустимий інтервал або демонструє нетипову динаміку (наприклад, різке зростання), система формує подію, яка потім передається до Alertmanager. Таким чином, механізм спостереження стає не просто пасивним, а реактивним — здатним динамічно реагувати на зміни у поведінці IT-сервісів [9].

Таке поєднання структурованого зберігання, семантики метрик і потужної запитної мови створює універсальне середовище для побудови масштабованих, гнучких і аналітично насичених систем моніторингу, які можуть ефективно реагувати на широкий спектр подій у реальному часі.

Prometheus є ефективним і масштабованим інструментом для збору, зберігання й аналізу технічних метрик у сучасних системах моніторингу серверного обладнання. Його архітектурна простота, багатовимірна модель даних, власна оптимізована база, потужна мова запитів і широка підтримка інтеграцій роблять його одним з найкращих рішень у середовищах з високими вимогами до продуктивності, гнучкості й автономності обробки.

1.3 Потужність візуалізації Grafana та її інтеграційні можливості

Grafana є високофункціональною платформою з відкритим вихідним кодом, призначеною для побудови інтерактивних і гнучко налаштовуваних візуалізацій часових рядів, агрегованих метрик та подій, що надходять із систем моніторингу.

Вона функціонує як вебзастосунок із серверною частиною (backend), написаною на Go, та клієнтською частиною, реалізованою за допомогою TypeScript/React, що забезпечує сучасний, інтуїтивно зрозумілий інтерфейс користувача. В архітектурному плані Grafana працює як проміжний рівень між джерелами метрик (наприклад, Prometheus, InfluxDB, Loki, Elasticsearch) і користувачем, виконуючи запити до баз даних, обробку результатів і відображення їх у вигляді панелей.

Grafana реалізує підтримку pull-моделі взаємодії, при якій запити до джерела даних ініціюються безпосередньо під час візуалізації або оновлення панелі. Для цього кожен дашборд (dashboard) складається з набору панелей (panels), кожна з яких виконує незалежний запит до одного або кількох джерел даних. Завдяки цьому забезпечується можливість одночасного відображення інформації з різних систем моніторингу. Запити до Prometheus формуються мовою PromQL і можуть виконуватись у режимі інтерактивного оновлення (live view), з налаштуванням частоти опитування, агрегування та проміжків часу. Це дозволяє отримувати метрики в реальному часі з мінімальною затримкою, а також будувати складні агрегати, що охоплюють десятки або сотні вузлів системи.

Крім класичної візуалізації, Grafana забезпечує глибоку інтеграцію з аналітичними механізмами. Користувач може використовувати шаблонізовані змінні ($variable) для створення адаптивних запитів, параметризувати віджети, а також використовувати трансформації даних безпосередньо в інтерфейсі — наприклад, для обчислення відносних значень, нормалізації, побудови трендів або визначення середніх/граничних величин у часових інтервалах. Завдяки плагінній архітектурі, Grafana також підтримує підключення кастомних панелей, джерел даних і бекендів для обробки запитів, що дозволяє розширювати функціональність без зміни ядра системи.

Grafana не лише є візуальним інструментом, а й виконує роль повноцінного клієнтського інтерфейсу аналітики даних, який дозволяє не лише спостерігати, але й формулювати аналітичні висновки, будувати гіпотези та керувати оповіщеннями. Її роль у моніторингових системах — забезпечення доступності, інформативності

та інтерпретованості метрик, що є критично важливими для інженерів DevOps, адміністраторів інфраструктури та розробників [10].

З технічної точки зору, Grafana є вебзастосунком, що дозволяє підключати різноманітні джерела даних (data sources), серед яких Prometheus, InfluxDB, Graphite, Elasticsearch, MySQL, PostgreSQL та інші. Основна мета Grafana — створення кастомізованих графічних панелей, які можуть включати графіки, гістограми, табличні уявлення, теплові карти, гейджі, сигнали стану й інші типи віджетів. Інтерфейс побудований за принципом drag-and-drop з підтримкою шаблонів, змінних і багаторівневої навігації між візуальними елементами.

Інтеграція Grafana з Prometheus відбувається через додавання Prometheus як джерела даних. Після встановлення Grafana (наприклад, через Docker або пакетну інсталяцію) адміністратор заходить у вебінтерфейс, де на головній панелі необхідно натиснути «Add data source» (рис. 1.8).

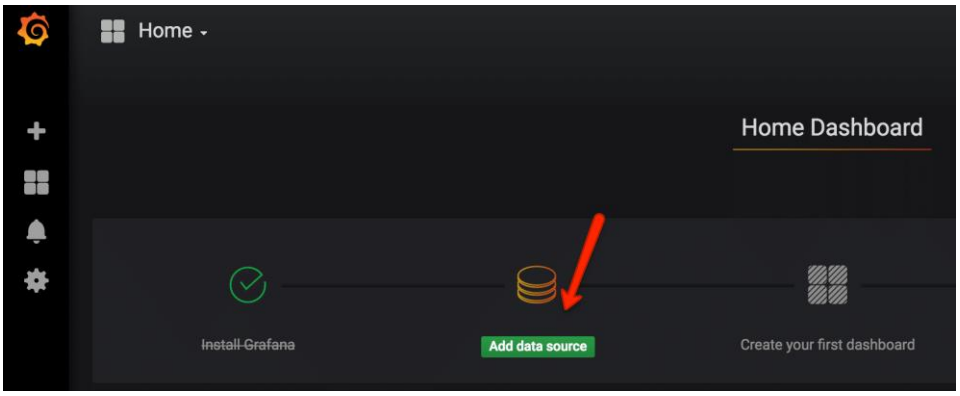


Рис. 1.8 Інтерфейс початкового налаштування Grafana з додаванням джерела даних [11]

Далі у полі налаштувань обирається тип Prometheus, вказується URL сервера Prometheus (наприклад, http://localhost:9090) і зберігається конфігурація (рис. 1.9).

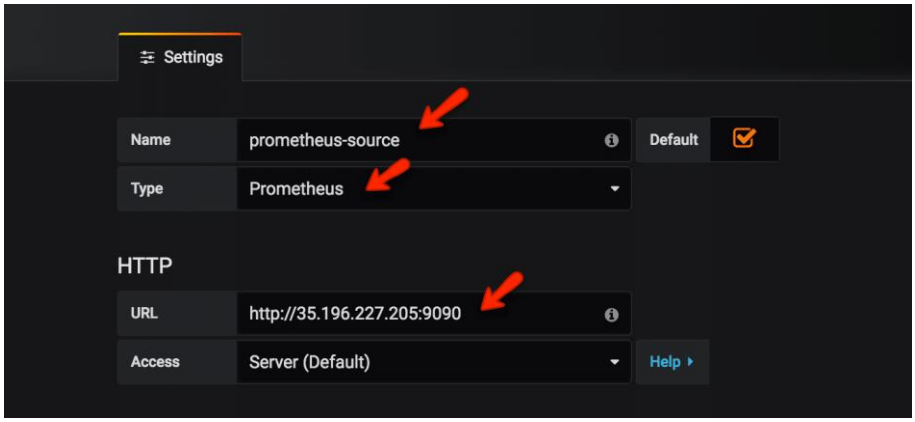


Рис. 1.9 Конфігурація Prometheus як джерела даних у Grafana [11]

Після успішного підключення користувач створює нову панель через меню «Create → Dashboard» (рис. 1.10), після чого відкривається інтерфейс вибору типу візуалізації: графік, таблиця, теплокарта, текстовий блок тощо (рис. 1.11).

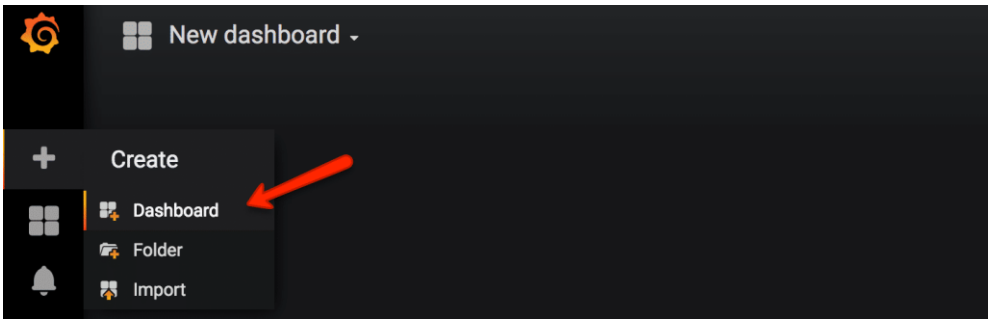


Рис. 1.10 Створення нового дашборду в меню Grafana [11]

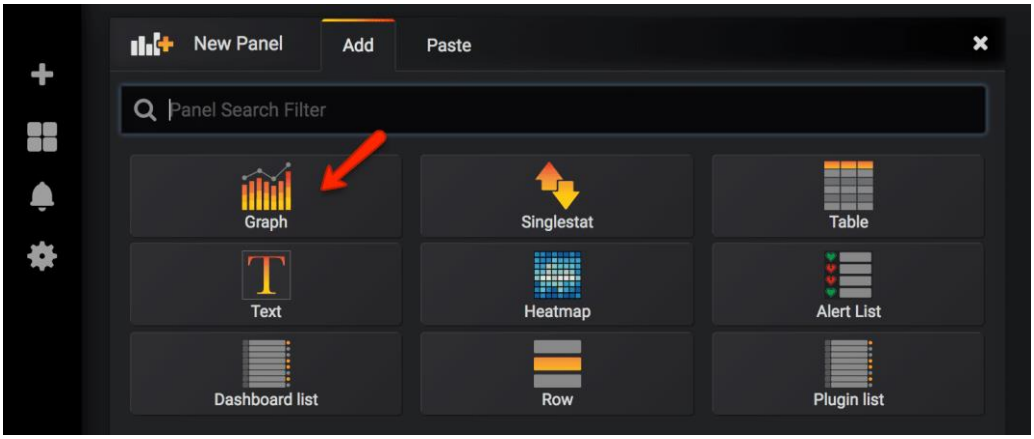


Рис. 1.11 Вибір типу панелі візуалізації в редакторі Grafana [11]

Обраний тип панелі відкривається у режимі редагування, де можна задати назву, відображення осей, легенду, стиль ліній, кольори й інші параметри (рис. 1.12).

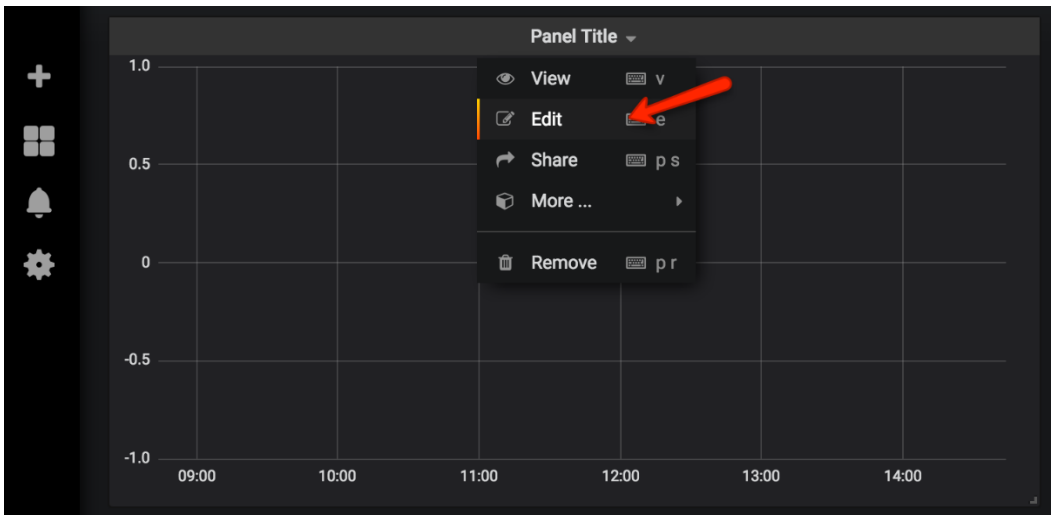


Рис. 1.12 Редактор панелі з параметрами графічного представлення [11]

Ключовим елементом є вкладка "Metrics", де користувач задає PromQL-запити до підключеного джерела даних. Наприклад, запит rate(node_cpu_seconds_total{mode="system"}[1m]) формує графік навантаження CPU за останню хвилину (рис. 1.13).

Рис. 1.13 Формування запиту PromQL та вивід метрик у реальному часі [11]

У цьому ж інтерфейсі можна попередньо переглянути результат візуалізації, налаштувати інтервали агрегації, одиниці виміру та інші параметри відображення. Після збереження панелі її можна перемістити до певної директорії, що зручно при організації великої кількості дашбордів (рис. 1.14).

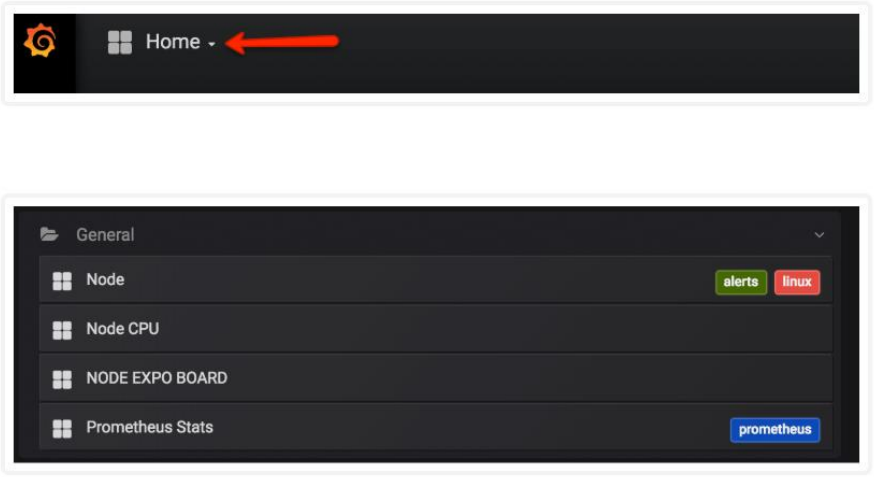


Рис. 1.14 Створення папки для впорядкування дашбордів [11]

Побудовані панелі відображають дані в реальному часі, з можливістю автоматичного оновлення, масштабування по осях часу та інтерактивної взаємодії з елементами. На рис. 1.15 показано приклад типового дашборду, який містить декілька графіків та гейджів для аналізу навантаження, використання пам’яті, активності запитів та стану мережі.

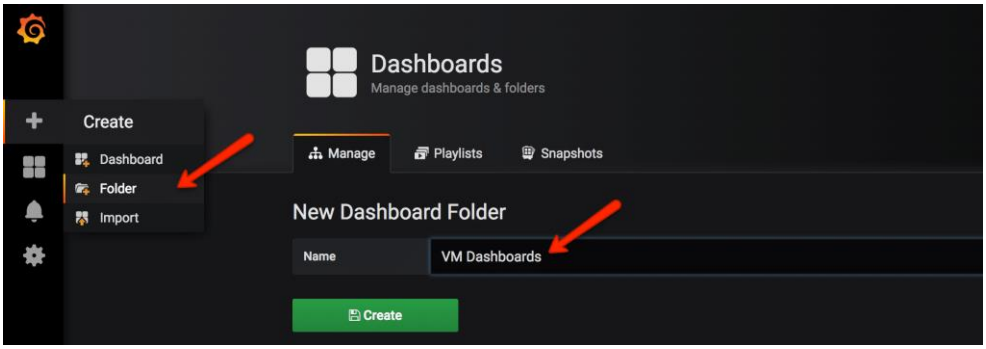


Рис. 1.15 Типовий дашборд Grafana з графіками, гейджами і показниками продуктивності [11]

Структура та логіка збереження дашбордів у Grafana дозволяє створювати ієрархії, ділити доступ між користувачами, експортувати конфігурації JSON для резервного копіювання або версіонування. Крім того, Grafana підтримує додаткові модулі алертингу, які дозволяють задавати порогові значення метрик і генерувати сповіщення через email, Slack або інші сервіси (рис. 1.16).

Рис. 1.16 Інтерфейс налаштування порогових значень і оповіщень в панелі Grafana [11]

Ця функціональність часто використовується у поєднанні з Alertmanager із Prometheus для централізованого керування подіями.

Grafana виступає як універсальний інтерфейс візуалізації для систем моніторингу, забезпечуючи як аналітичні можливості, так і гнучкість в адмініструванні. Її інтеграція з Prometheus дозволяє реалізувати повноцінний технологічний стек для безперервного моніторингу стану серверів, мережевих сервісів, контейнеризованих середовищ та бізнес-метрик на одному екрані.

РОЗДІЛ 2. ТЕХНІЧНЕ ПРОЄКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ

2.1 Постановка технічного завдання та вибір технологічного стеку

Сучасні IT-системи характеризуються зростаючою складністю, динамікою навантаження, а також критичними вимогами до доступності, продуктивності та безпеки. У таких умовах ключового значення набуває впровадження інструментів оперативного моніторингу, здатних не лише відображати поточний стан інфраструктури, а й попереджати про можливі збої, дозволяючи приймати превентивні дії. З огляду на це, формулюється потреба в розробці універсальної, масштабованої системи моніторингу, орієнтованої на серверні обчислювальні ресурси та прикладні сервіси.

Метою технічного проєкту є побудова системи, що дозволить здійснювати автоматизований моніторинг широкого спектру показників — включно з метриками навантаження на процесор, пам’ять, файлові системи, мережеві інтерфейси, а також стан контейнеризованих середовищ, доступність віддалених ресурсів і загальний стан інфраструктури. Крім цього, система повинна забезпечувати можливість збору, аналізу та зберігання логів у контексті подій, а також реалізовувати функціональність сповіщення у разі порушення заздалегідь визначених умов.

До функціональних вимог до системи належать: підтримка збору метрик із різнорідних джерел (Linux/Windows-сервери, Docker, веб-сервіси), побудова гнучких та масштабованих дашбордів з можливістю глибокої інтерактивної аналітики, підтримка запитів на основі часових рядів, налаштування умов алертингу та маршрутизації сповіщень, збереження логів і можливість їх перегляду через централізований інтерфейс. Важливою умовою є підтримка відкритих стандартів, спрощена інтеграція зі сторонніми сервісами та незалежність від конкретної операційної системи чи платформи.

Серед нефункціональних вимог передбачається висока доступність системи, мінімальний вплив на продуктивність цільових серверів, гнучкість у розгортанні (зокрема, за допомогою контейнеризації), зручність адміністрування, а також

модульність та розширюваність архітектури. Система має масштабуватись як вертикально (за рахунок потужності обладнання), так і горизонтально (через розділення збору, зберігання, візуалізації та оповіщення на окремі компоненти). Також важливо забезпечити безпеку доступу, керування ролями користувачів і підтримку аудиту дій.

Технічне завдання, сформульоване з урахуванням зазначених вимог, включає розробку модульної архітектури з використанням спеціалізованих агентів збору даних, системи зберігання метрик з підтримкою мови запитів PromQL, візуального інтерфейсу для побудови дашбордів та засобів конфігурації алертів. Вибір технологічного стеку має базуватись на принципах відкритості, продуктивності, активної підтримки спільноти та сумісності з DevOps-інструментарієм.

Вибір технологічного стеку для реалізації системи моніторингу був обґрунтований вимогами до масштабованості, відкритості, гнучкої інтеграції, а також наявністю широкої спільноти підтримки й активної екосистеми інструментів. До складу обраного рішення увійшли компоненти, які охоплюють усі етапи моніторингу — від збору даних до візуалізації та генерації сповіщень.

На рівні збору та передачі даних використовуються спеціалізовані агенти: Node Exporter для Linux [12] та Windows Exporter для ОС Windows [13]. Вони забезпечують низькорівневий збір ключових системних метрик, таких як завантаження процесора, обсяг використаної пам’яті, навантаження на дискову підсистему, мережеву активність тощо. Для моніторингу контейнеризованих середовищ застосовується cAdvisor [14], що дозволяє отримувати показники щодо використання ресурсів окремими Docker-контейнерами, зокрема CPU, пам’яті, I/O і тривалості життя процесів [15]. Збір логів забезпечує Promtail — агент, який транслює логи з локальної файлової системи [16] у систему Grafana Loki з додаванням метаданих (міток), що дозволяє будувати гнучкі запити та групувати записи [17].

Збереженням та обробкою часових рядів займається Prometheus — високооптимізована time-series база даних із власним механізмом збирання (pull-модель), форматом зберігання та мовою запитів PromQL [18]. Система автоматично

опитує зареєстровані джерела даних із вказаною періодичністю та зберігає результати у локальному сховищі. У свою чергу, для логів використовується Grafana Loki — рішення, побудоване за принципами Prometheus, але адаптоване для неструктурованих текстових даних. Loki не індексує вміст логів, що дозволяє досягти високої продуктивності при масштабуванні.

Система сповіщень реалізована за допомогою Alertmanager, що інтегрується з Prometheus і відповідає за обробку спрацювань алертів, маршрутизацію повідомлень, подавлення повторень (silencing) і агрегацію подій. Alertmanager підтримує вивід до зовнішніх каналів: електронна пошта, Slack, Telegram, PagerDuty тощо. Крім того, у проєкті застосовується Blackbox Exporter — інструмент для зовнішнього моніторингу, який дозволяє перевіряти доступність віддалених сервісів шляхом виконання HTTP(S), TCP або ICMP (ping) запитів [19, 20].

Останньою, але критично важливою ланкою є візуалізація та аналітика. Для цього використовується Grafana — потужна open source платформа, яка дозволяє створювати адаптивні інформаційні панелі з виводом метрик і логів у вигляді графіків, гейджів, теплокарт, таблиць та інших візуальних компонентів. Grafana надає можливість створення інтерактивних запитів до Prometheus або Loki, підтримує шаблонізовані змінні, алерти, ролі доступу, історію версій дашбордів та інтеграцію з системами авторизації.

Рис. 2.1 ілюструє повний ланцюг передачі даних в контексті використання обраних інструментів: від агентів збору до систем зберігання, оповіщення і візуалізації.

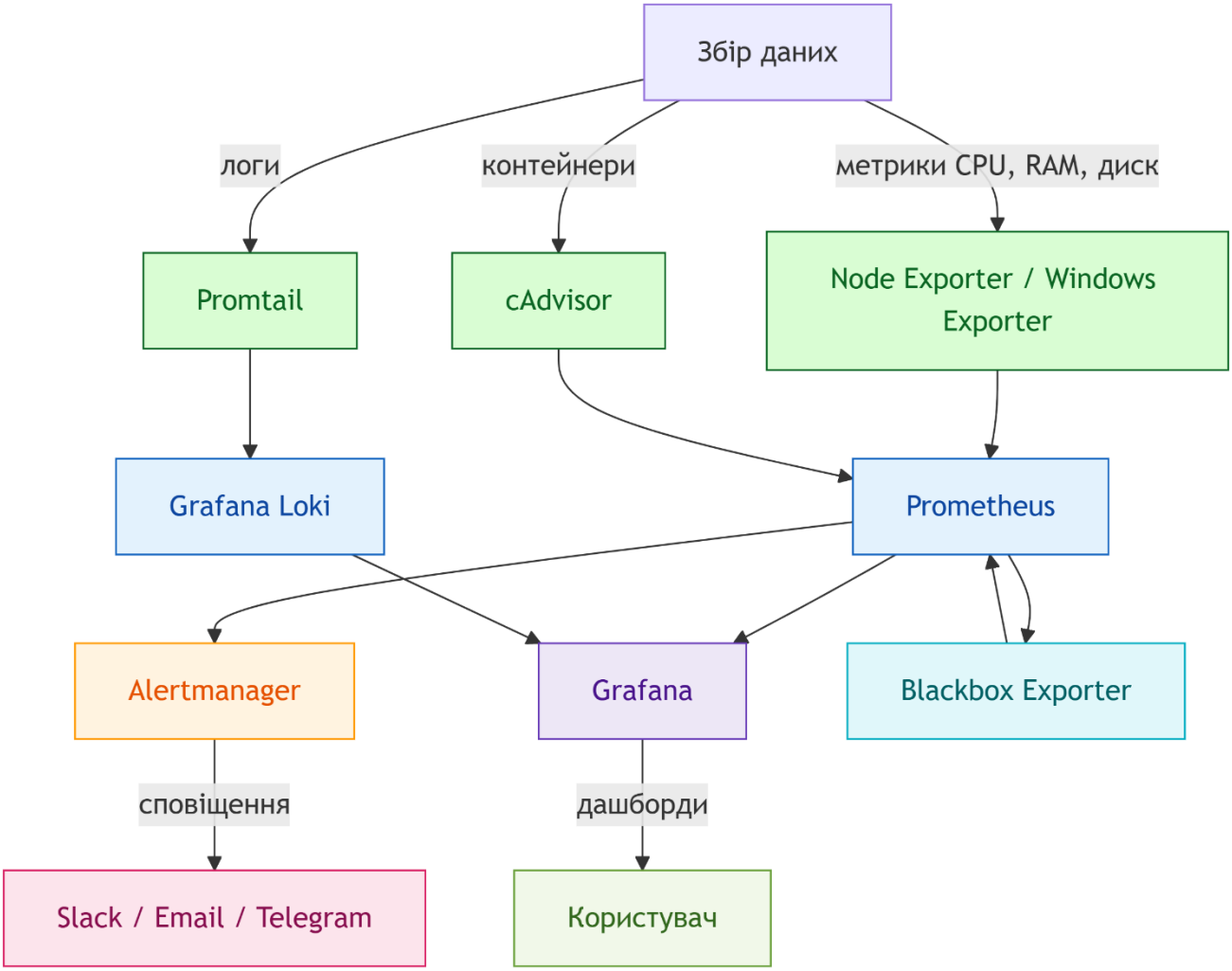


Рис. 2.1 Структура технологічного стеку системи моніторингу

Обраний стек технологій забезпечує повноцінний замкнутий цикл моніторингу: від збору сирих даних — до візуального аналізу й автоматичних дій у разі виявлення відхилень. Його модульність, підтримка контейнеризації та відкрита архітектура дозволяють ефективно масштабувати систему під вимоги підприємств будь-якого рівня — від невеликих серверних ферм до комплексних кластерних інфраструктур.

2.2 Проєктування архітектури системи моніторингу серверів

Під час проєктування системи моніторингу серверного обладнання в умовах багатокомпонентної інфраструктури постає необхідність розробки такої

архітектури, яка забезпечувала б централізований, масштабований та надійний збір, зберігання, обробку, візуалізацію та оповіщення про критичні події. Враховуючи сучасні вимоги до доступності сервісів, обрано дворівневу архітектуру, що передбачає розділення компонентів на рівень Application Host (вузли об'єктів моніторингу) та Monitoring Host (централізований вузол збору, обробки та представлення даних). Загальна схема взаємодії компонентів наведена на рисунку 2.2, який ілюструє потоки даних між агентами збору метрик, Prometheus, Alertmanager, Grafana та системою логування Loki. Вказано також канали доставки повідомлень (наприклад, Slack) і зовнішній моніторинг через Blackbox Exporter.

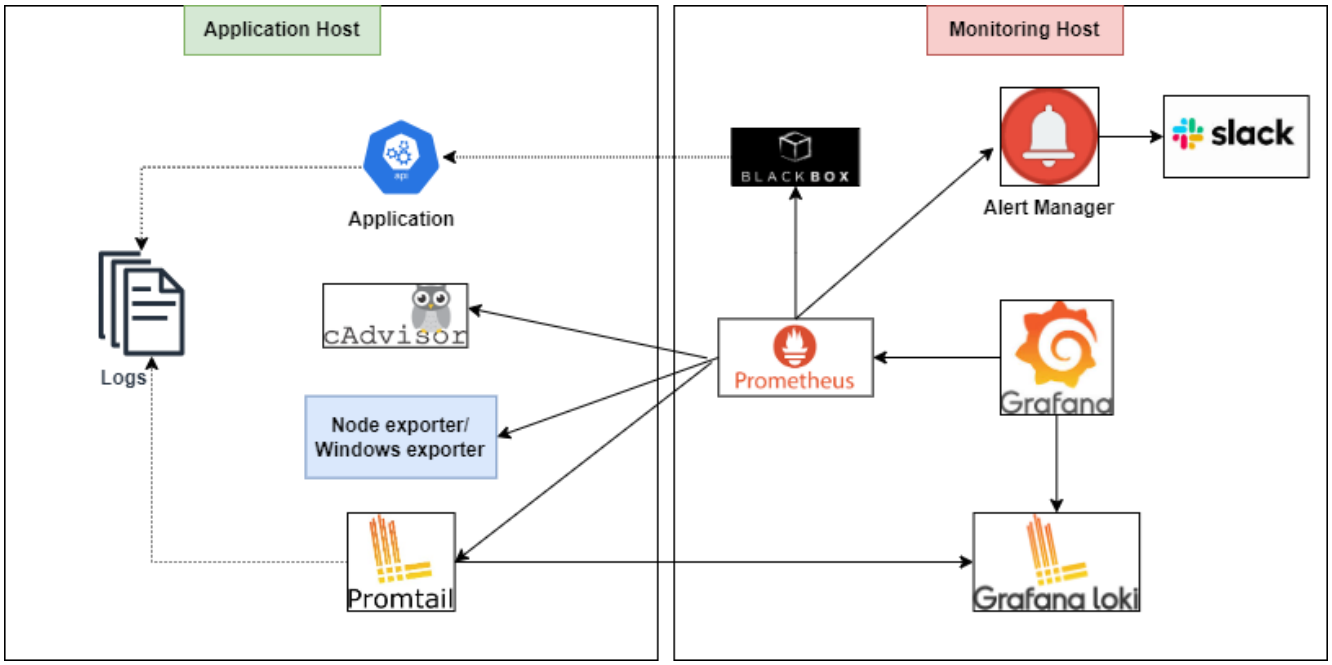


Рис. 2.2 Архітектурна схема проєктованої системи моніторингу серверів

На рівні вузлів застосунків передбачається розміщення агентів збору операційних метрик (Node Exporter або Windows Exporter), які відповідають за експонування показників навантаження на CPU, використання пам’яті, мережевої активності, I/O-дисків тощо. У випадку, коли застосунки розгорнуті в Docker-середовищі, доцільним є використання cAdvisor — спеціалізованого агента, розробленого Google, що збирає дані про ресурси контейнерів, включаючи використання CPU, пам’яті, файлової системи, а також інформацію про тривалість

життя та кількість екземплярів контейнерів. Для збору логів із застосунків передбачено застосування Promtail, який агрегує журнали з файлової системи та передає їх у систему централізованого логування Loki.

На рівні централізованого моніторингового вузла (Monitoring Host) проєктується встановлення основного ядра системи — Prometheus, який виконує періодичне опитування (scraping) відповідних метрик із агентів, розміщених на вузлах застосунків, і зберігає їх у власній time-series базі даних (TSDB). Додатково, до Prometheus підключається Blackbox Exporter, який забезпечує зовнішній моніторинг мережевих сервісів (HTTP, TCP, ICMP), здійснюючи пінгування та HTTP-запити до кінцевих точок REST або інших сервісів.

Система оповіщення проектується на основі Alertmanager, що приймає події про порушення стану з Prometheus, обробляє їх відповідно до конфігураційних правил (групування, глушіння, маршрутизація) та спрямовує сповіщення до відповідних каналів — Slack, Email, Telegram тощо. Важливо передбачити інтеграцію Alertmanager із шаблонами повідомлень та механізмами фільтрації, що дозволяє уникнути надлишкових або дублікатних алертів при масштабному навантаженні.

Компонентом візуалізації проєктується Grafana — вебзастосунок для побудови інтерактивних дашбордів. Grafana отримує доступ до Prometheus як джерела даних, дозволяє візуалізувати метрики в реальному часі, створювати агреговані звіти, гістограми, гейджі, а також інтегруватися з Loki для відображення логів із Promtail. Передбачено використання шаблонізованих змінних ($job, $instance) для побудови адаптивних панелей, зручних у масштабованому середовищі.

Усі компоненти мають бути ізольовані в окремих Docker-контейнерах, що забезпечить гнучке управління залежностями, контроль доступу через обмеження мережевих портів (наприклад, 9090 — Prometheus, 9093 — Alertmanager, 3100 — Loki, 3000 — Grafana) та підтримку горизонтального масштабування. Внутрішній зв'язок між контейнерами проектується через спільну Docker-мережу, що спрощує

маршрутизацію запитів між сервісами без необхідності використання DNS чи зовнішніх проксі.

Для забезпечення ізоляції моніторингового середовища передбачається розміщення Monitoring Host на окремій віртуальній машині, що дозволить відокремити аналітичне навантаження від продуктивного. Такий підхід відповідає принципам безпеки та стабільності роботи, а також полегшує масштабування — як шляхом додавання нових Application Host, так і через балансування збору метрик.

2.3 Розгортання та налаштування Prometheus і Grafana

Проєктована система моніторингу реалізується як комплекс взаємопов’язаних контейнеризованих сервісів, об’єднаних у єдине середовище за допомогою засобів оркестрації Docker Compose. З метою досягнення модульності, повторного використання та зручності керування конфігураціями, усі компоненти системи розгортаються на основі заздалегідь структурованих YAML-файлів, що визначають параметри запуску, мережеву взаємодію, зовнішні точки доступу та налаштування сервісів моніторингу, візуалізації та логування.

До складу конфігураційних файлів, які формують архітектурну і функціональну основу платформи моніторингу, входять такі ключові елементи:

- docker-compose.yml — визначає структуру контейнерів, їхні залежності, змінні середовища, порти та спільні мережі;
- prometheus.yml — основний файл конфігурації Prometheus, що містить параметри опитування метрик та перелік цільових вузлів;
- datasource.yml і dashboard.yml — частина автоматичного provisioning механізму Grafana, який дозволяє одразу при запуску системи підключити відповідні джерела даних і завантажити попередньо створені дашборди;
- alertmanager.yml — конфігурація служби Alertmanager, яка відповідає за обробку та маршрутизацію оповіщень;

- loki.yaml та promtail.yml — описують логіку збирання й обробки логів для системи Loki, включаючи шляхи до лог-файлів, шаблони парсингу та механізм передачі;
- blackbox.yml — конфігурація зовнішнього експортера Blackbox, що дозволяє виконувати мережеві перевірки доступності за HTTP, TCP, ICMP;
- допоміжні Bash-скрипти generate_skeleton.sh і utils_helper.sh — виконують роль інструментів генерації базової структури проєкту або налаштування середовища.

Ці файли не тільки забезпечують автоматизоване розгортання компонентів, а й уможливлюють централізоване адміністрування, зменшення ручної конфігурації та підвищення повторюваності розгортання в інших середовищах.

Файл docker-compose.yml виконує роль основного маніфесту оркестрації, що визначає конфігурацію запуску взаємопов'язаних контейнерів, які формують цілісну моніторингову платформу (рис. 2.3).

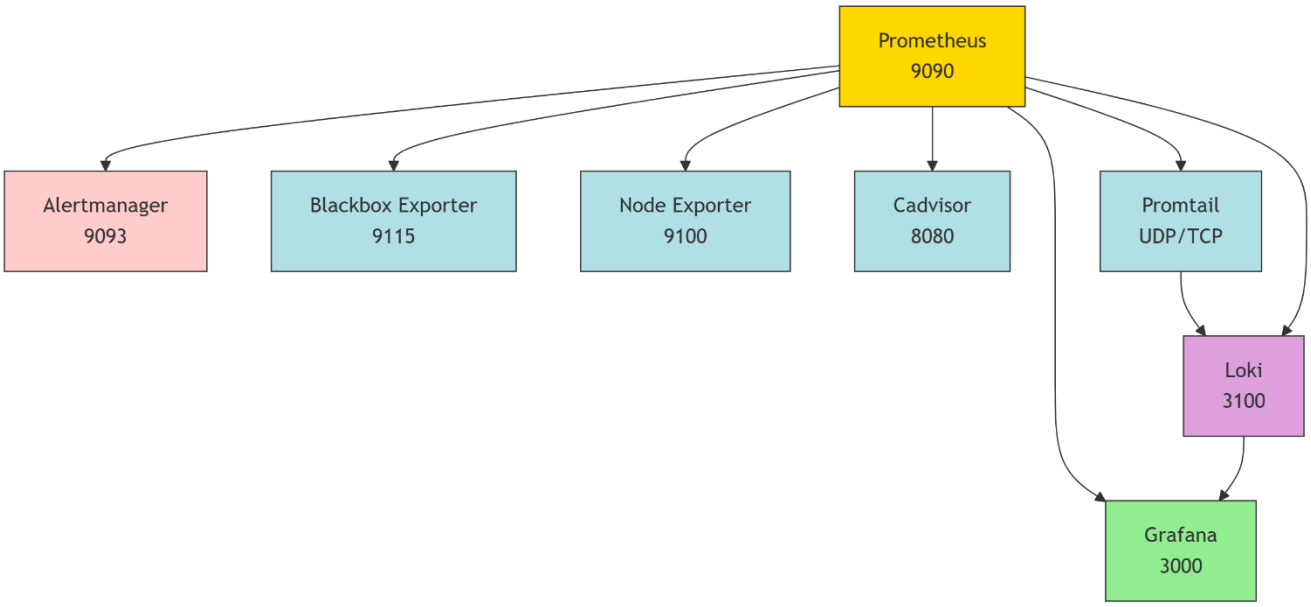


Рис. 2.3 Архітектура контейнерів системи моніторингу на основі docker-compose.yml

Він базується на специфікації третьої версії Docker Compose (version: '3') і описує сервіси, необхідні для збору, обробки, візуалізації та маршрутизації метрик,

логів і оповіщень. Кожен сервіс вказано в секції services з деталізацією таких параметрів, як образ (image), ім’я контейнера (container_name), змонтовані томи (volumes), відкриті порти (ports), команди запуску (command) та залежності (depends_on).

Зокрема, сервіс prometheus використовує офіційний образ prom/prometheus:latest та монтує конфігураційний каталог і каталог зберігання даних метрик. Аргументи командного запуску включають посилання на файл prometheus.yml та встановлення терміну зберігання часових рядів на один рік (--storage.tsdb.retention.time=1y). Сервіс alertmanager працює в парі з Prometheus, отримуючи від нього правила сповіщень, і вимагає доступу до файлу alertmanager.yml. Його запуск також регламентується відповідними аргументами, а залежність від Prometheus забезпечується директивою depends_on.

Додатково, blackbox — це контейнер експортера blackbox-exporter, що дозволяє здійснювати активний моніторинг доступності зовнішніх систем за допомогою HTTP, TCP або ICMP. Він конфігурується файлом blackbox.yml і відкриває порт 9115 для отримання запитів від Prometheus. Сервіс grafana орієнтований на візуалізацію даних і запускається з образом grafana/grafana:8.4.11, збереженням даних у постійному томі grafana-storage і використанням налаштувань з grafana.ini. Окрім цього, Grafana автоматично імпортує дашборди та джерела даних через механізм provisioning (папка provisioning).

Файл prometheus.yml є основним конфігураційним елементом моніторингового рушія Prometheus, який задає загальні глобальні параметри, визначає джерела збору метрик (scrape targets), механізми алертингу та підключення до сервісу Alertmanager. Структура цього YAML-файлу побудована відповідно до офіційної специфікації Prometheus і містить три ключові блоки: global, rule_files, alerting та scrape_configs (рис. 2.4).

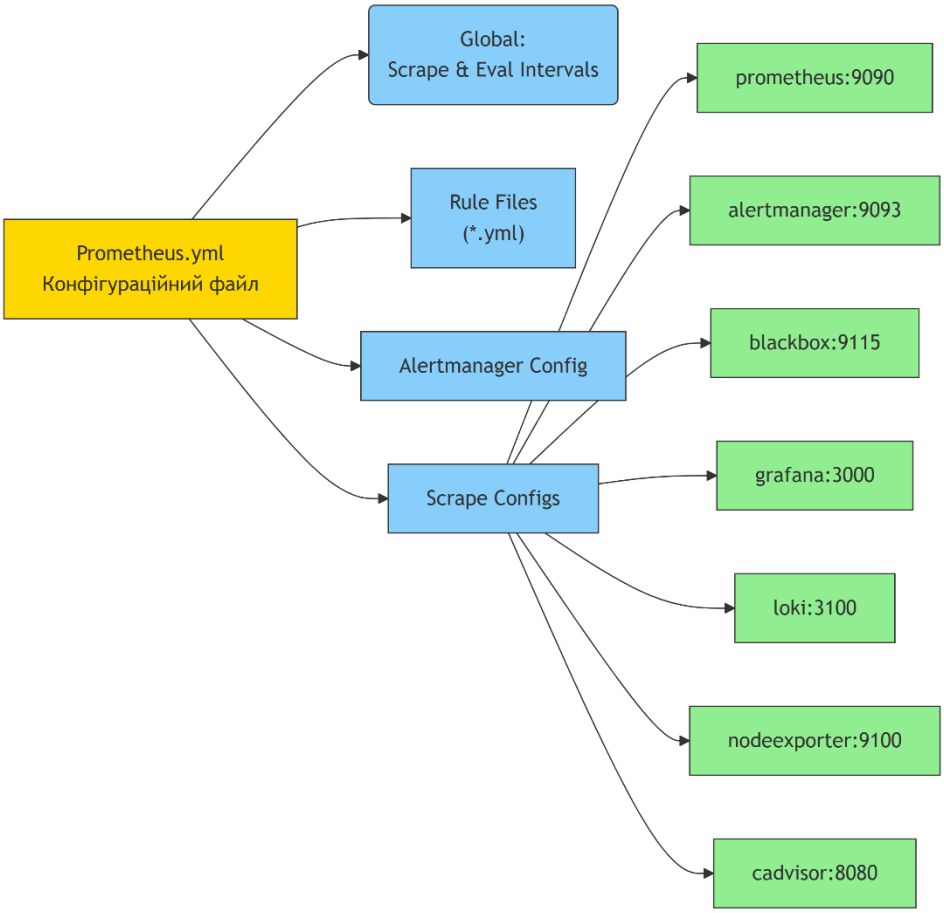


Рис. 2.4 Модель конфігурації Prometheus

У блоці global задано базові інтервали: scrape_interval: 30s — інтервал опитування джерел метрик, evaluation_interval: 40s — інтервал обробки правил алертів, а також визначено мітки external_labels, які будуть прикріплюватися до всіх метрик, що передаються до зовнішніх систем, наприклад, у federated Prometheus.

Блок rule_files містить перелік зовнішніх файлів правил алертингу (*.yml), серед яких: self_monitoring_alert_rules.yml, node_exporter_alert_rules.yml, blackbox_alert_rules.yml тощо. Це дозволяє централізовано зберігати та версіонувати логіку оповіщення для кожного типу експортера або сервісу.

Секція alerting визначає параметри підключення до сервісу Alertmanager. Конфігурація включає список Alertmanager-інстансів з фіксованими адресами (static_configs), до яких Prometheus надсилає згенеровані алерти. Кожна ціль маркується додатковими змінними середовища (target_env, target_resource_name),

що дозволяє здійснювати гнучку маршрутизацію повідомлень на рівні Alertmanager.

Найбільшу функціональну вагу має блок scrape_configs, який визначає джерела збору метрик. У поточному конфігураційному файлі він включає job "self_monitoring_job", у межах якого перелічено цілу низку контейнерів: prometheus, alertmanager, blackbox, grafana, loki, nodeexporter, cadvisor тощо. Кожен з них має адресу у форматі host:port та пов’язаний з набором міток, які полегшують ідентифікацію метрик у багатокомпонентному середовищі. Механізм static_configs гарантує, що ці адреси опитуються незалежно від зовнішніх систем реєстрації, таких як Consul або Kubernetes.

Файл datasource.yml є складовою механізму автоматичного provisioning у Grafana і призначений для конфігурації джерел даних, які будуть доступні у вебінтерфейсі після запуску сервісу (рис. 2.5).

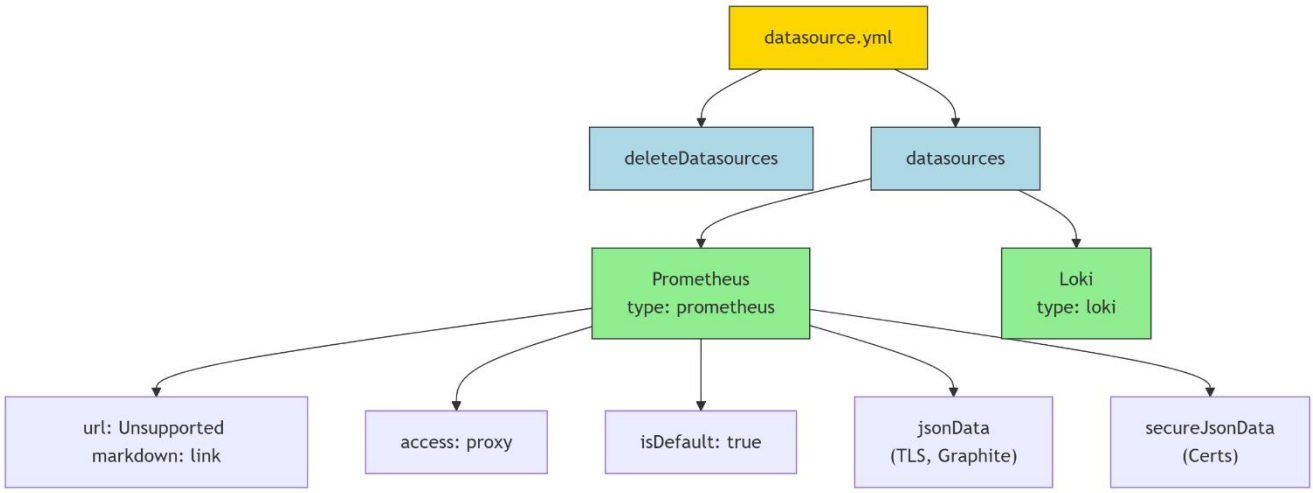


Рис. 2.5 Структура конфігурації datasource.yml

Використання подібних конфігурацій дозволяє централізовано задати параметри підключення до сервісів збору метрик або логів, уникаючи необхідності ручного введення даних через графічний інтерфейс.

Конфігураційна структура визначена через apiVersion: 1, що відповідає першій версії API provisioning у Grafana. У секції deleteDatasources зазначено перелік джерел даних, які мають бути видалені перед створенням нових — зокрема,

"Prometheus" та "Loki", що гарантує відсутність конфліктів при повторному розгортанні.

Основна частина файлу описана в секції datasources, де для кожного джерела даних вказуються тип (type: prometheus), режим доступу (access: proxy), URL-адреса сервісу (url: http://prometheus:9090), організаційний ідентифікатор (orgId: 1) та інші параметри, які контролюють автентифікацію, SSL-сертифікацію та політику доступу. Параметр isDefault: true встановлює Prometheus як джерело даних за замовчуванням, що автоматично використовується при створенні нових панелей у дашбордах.

Крім основних налаштувань, у структурі передбачено секції jsonData та secureJsonData, які містять параметри для тонкого налаштування TLS-автентифікації, роботи з Graphite (у випадку мультиджерельної архітектури) та шифровані поля, які зберігаються безпосередньо у внутрішній базі Grafana. Завдяки такому підходу, адміністратор має змогу відтворити середовище Grafana з нуля, маючи лише конфігураційні файли provisioning.

Файл dashboard.yml є складовою механізму автоматичного provisioning у Grafana і відіграє роль конфігураційного мосту, який визначає правила завантаження попередньо підготовлених дашбордів у систему. Він не містить самих JSON-описів панелей, але задає, де саме вони зберігаються, як інтерпретуються та яку поведінку мають у середовищі Grafana (рис. 2.6).

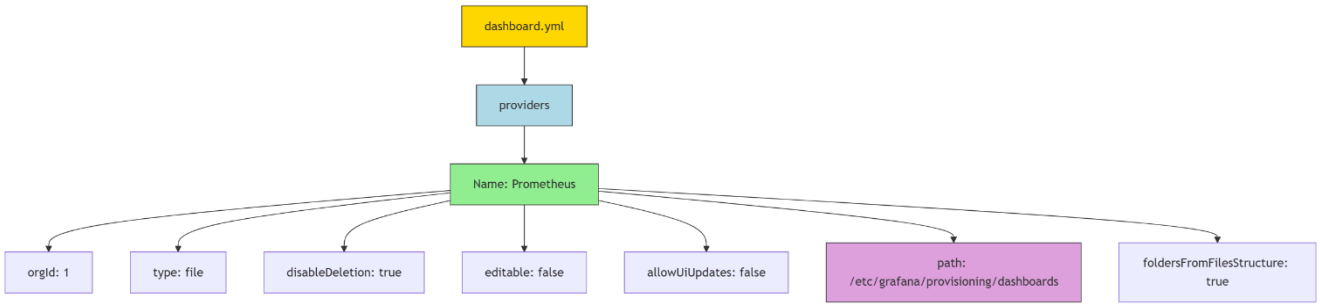


Рис. 2.6 Схема конфігурації провайдерів дашбордів Grafana

Основу файлу складає блок providers, який визначає список джерел (провайдерів) дашбордів. У наведеній конфігурації описано одного провайдера з

назвою "Prometheus", що застосовується до організації з ідентифікатором orgId: 1. Провайдер реалізується через type: file, тобто файли JSON-дешбордів зчитуються з файлової системи, а не створюються або модифікуються в інтерфейсі Grafana.

Ключовим є параметр options.path, який вказує шлях до каталогу з дашбордами — у даному випадку, /etc/grafana/provisioning/dashboards. Завдяки цьому підходу забезпечується повна автоматизація створення інтерфейсів моніторингу після запуску Grafana, без участі користувача.

Параметри disableDeletion: true, editable: false та allowUiUpdates: false накладають обмеження на користувача в інтерфейсі — створені дашборди не можуть бути змінені або видалені вручну. Це особливо актуально для середовищ із суворими вимогами до цілісності візуалізації, наприклад, у промислових або високонавантажених інфраструктурах, де моніторинг має бути ідентичним на всіх інстансах Grafana.

Файл alertmanager.yml є конфігураційним ядром компонента Alertmanager — сервісу, який працює спільно з Prometheus для обробки, маршрутизації, групування та доставки сповіщень про події. Даний файл описує логіку обробки алертів, джерела їх доставки (receivers), а також специфічні маршрути залежно від параметрів алерту (рис. 2.7).

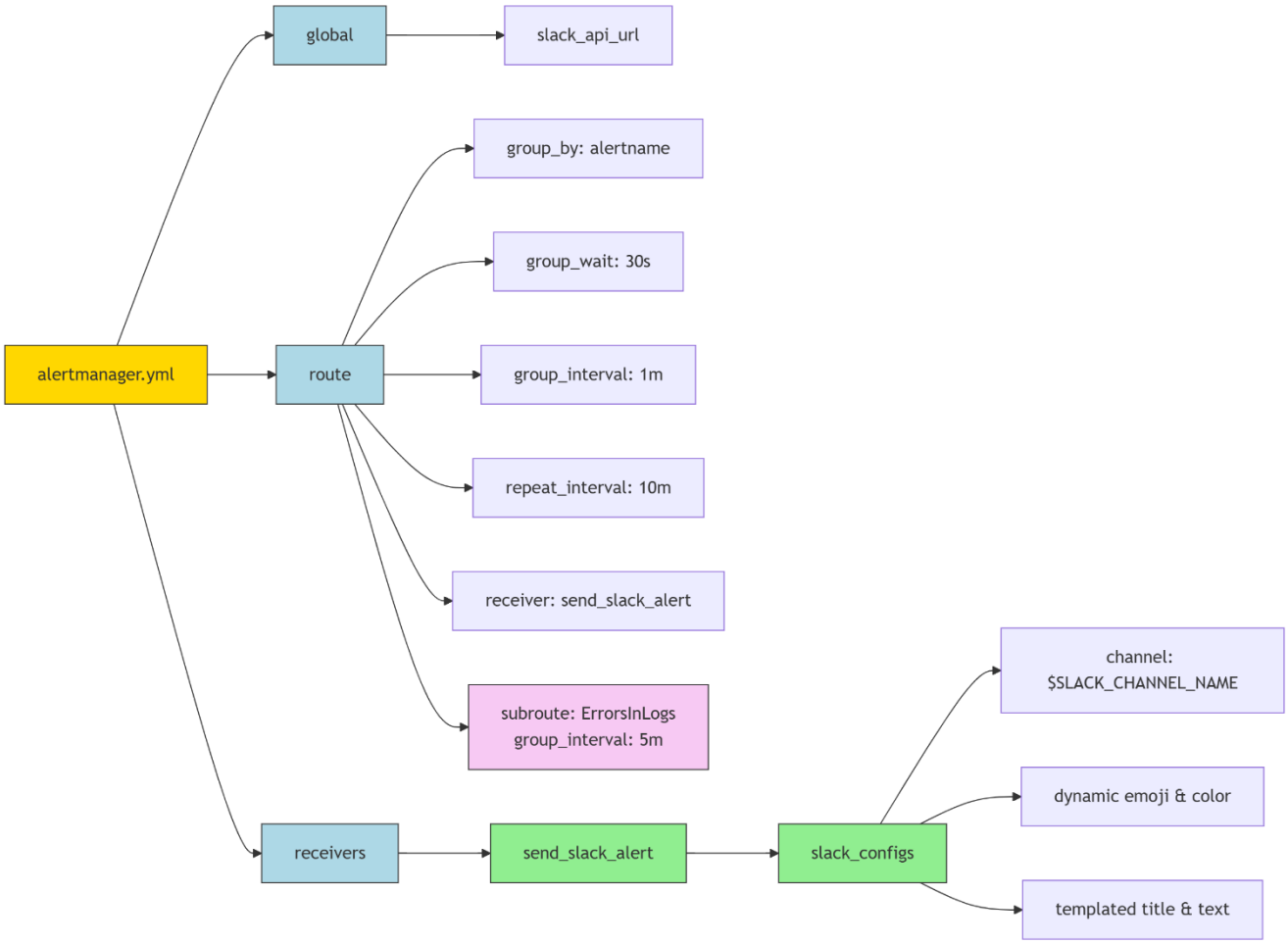


Рис. 2.7 Логіка конфігурації Alertmanager

У секції global задається глобальний параметр slack_api_url, значення якого передається через змінну середовища ${SLACK_WEB_HOOK_URL}. Це забезпечує гнучкість при розгортанні системи в різних середовищах та уникнення жорстко прописаних URL.

Блок route описує загальну логіку маршрутизації алертів. Встановлено групування за міткою alertname, що дозволяє агрегувати алерти одного типу у спільні повідомлення. Параметри group_wait, group_interval і repeat_interval регламентують час очікування перед надсиланням першого повідомлення, інтервал між повторними сповіщеннями для вже існуючих алертів і паузу між повторними надсиланнями. Це особливо важливо для уникнення спаму при великій кількості подій.

У полі receiver вказано ім’я основного отримувача "send_slack_alert", який буде використано за замовчуванням. У масиві routes зазначено спеціалізований

маршрут для алертів із міткою alertname: "ErrorsInLogs", що має окремий group_interval у 5 хвилин.

Нарешті, секція receivers містить визначення логіки доставки для кожного каналу. У нашому випадку це Slack. Налаштування для slack_configs включає динамічне формування вмісту повідомлення за допомогою шаблонів Go Template — наприклад, колір алерту залежить від статусу (firing, resolved) і рівня важливості (warning, critical). Повідомлення формуються з полями title, text, footer, де використовується форматований текст, підставлення змінних із .CommonLabels та .Annotations.

Файл loki.yaml є основним конфігураційним документом для налаштування лог-агрегатора Loki, розробленого компанією Grafana Labs. Loki — це система збирання, індексації та зберігання логів, що оптимізована для горизонтального масштабування та інтеграції з Grafana. Конфігураційна структура забезпечує повний контроль над процесами інгестування, зберігання, обробки логів і правилами алертингу (рис. 2.8).

У заголовку конфігурації встановлено auth_enabled: false, що вимикає автентифікацію між компонентами системи — підхід, допустимий для тестових або захищених середовищ. У секції server визначено порт прослуховування HTTP-сервера Loki: http_listen_port: 3100.

Блок ingester містить параметри для керування процесом прийому логів. Зокрема, активовано ведення WAL (write-ahead log), з директорією /loki/wal, що підвищує надійність зберігання даних. Параметри chunk_idle_period, chunk_retain_period і max_transfer_retries регулюють життєвий цикл блоків даних.

Секція schema_config визначає механізм індексації та зберігання логів: використовується схема v11, з об'єктним сховищем типу filesystem, індексація виконується за префіксом index_ з періодом 168 годин. Це забезпечує ефективне поєднання з локальним зберіганням (boltdb і filesystem), що конфігурується у блоці storage_config.

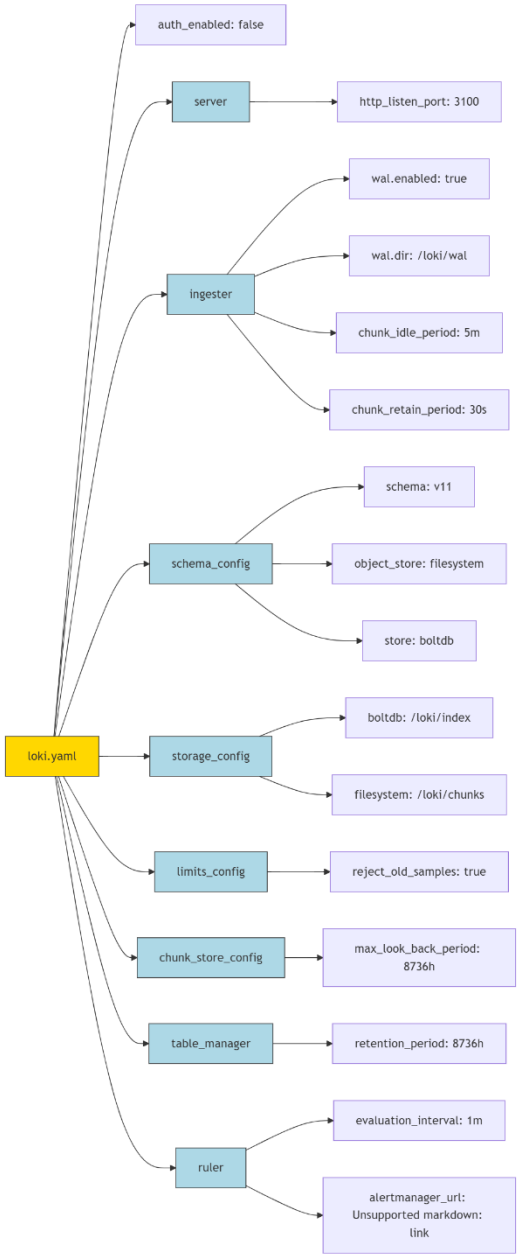


Рис. 2.8 Конфігураційна структура Loki

Розділ limits_config включає параметри обмежень: reject_old_samples активує відхилення застарілих записів логів, віком понад 168 годин. Такі обмеження дозволяють контролювати обсяг та актуальність даних у системі.

Секція chunk_store_config визначає максимальний інтервал, в межах якого доступні історичні дані: max_look_back_period: 8736h (тобто один рік). Також активовано механізм очищення даних (retention_deletes_enabled: true) з періодом зберігання 8736 годин, що задається у table_manager.

Крім того, файл містить конфігурацію ruler, яка відповідає за автоматичну оцінку правил (rules) і генерацію алертів. Використовується локальне сховище (storage: type: local) та шляхи до правил. Алертинг інтегрується з компонентом Alertmanager через URL http://alertmanager:9093.

Файл promtail.yml визначає конфігурацію для запуску Promtail — агента збору логів, який встановлюється на вузлах інфраструктури та відповідає за читання локальних лог-файлів, їхню попередню обробку й передачу до системи зберігання Loki через HTTP API (рис. 2.9).

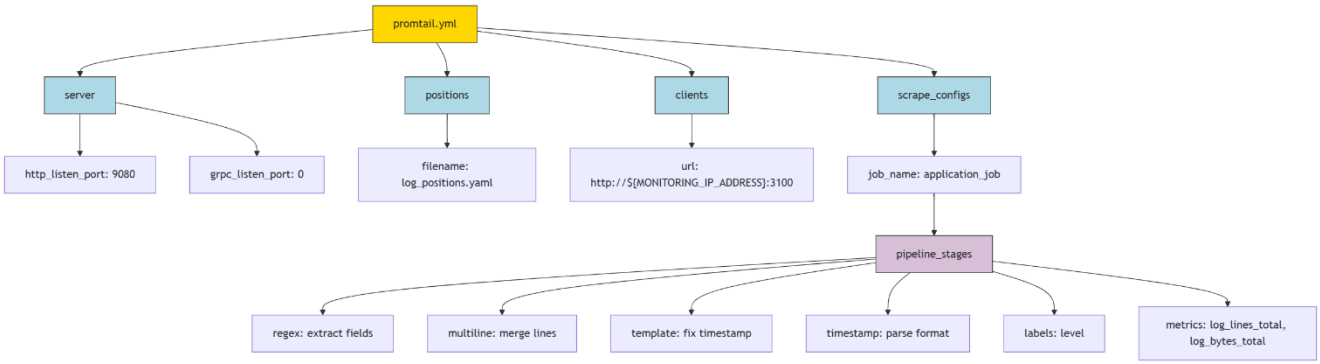


Рис. 2.9 Конфігурація агента збору логів Promtail

У секції server вказано порти, на яких запускається HTTP-інтерфейс Promtail (http_listen_port: 9080) та вимкнено gRPC (grpc_listen_port: 0), що вказує на монокомпонентну локальну інсталяцію без кластеризації. Це дозволяє спростити налаштування у випадку простих архітектур.

Секція positions є критично важливою для відстеження позиції зчитування файлів. Файл log_positions.yaml, шлях до якого задається змінною середовища ${MY_APP_PATH}, зберігає інформацію про останню прочитану позицію в кожному лог-файлі. Завдяки цьому Promtail не дублює лог-записи при перезапуску.

У блоці clients зазначено URL, куди Promtail надсилає оброблені логи — це інтерфейс API Loki: http://${MONITORING_IP_ADDRESS}:3100/loki/api/v1/push. Змінна середовища дозволяє зробити конфігурацію адаптивною для різних середовищ розгортання.

Основна функціональність визначена в секції scrape_configs, яка описує джерела логів і етапи їх обробки у вигляді pipeline_stages. Зокрема:

- regex — витягує з кожного рядка лог-файлу поля timestamp, threadName, level, component, content на основі регулярного виразу;
- multiline — групує багаторядкові повідомлення в одне лог-повідомлення, якщо воно починається з часу (timestamp);
- template — форматування часу із заміною ком на крапки для уніфікації;
- timestamp — нормалізація часових значень у форматі Go ("2006-01-02 15:04:05.000");
- labels — автоматично присвоює метку level до кожного логу, що дозволяє фільтрацію;
- metrics — два кастомні лічильники: log_lines_total (кількість лог-рядків) і log_bytes_total (об’єм у байтах), які інкрементуються відповідно до налаштувань. Вони дозволяють метрикувати не лише самі повідомлення, а й обсяг трафіку.

Ця конфігурація дозволяє Promtail ефективно працювати в середовищі, де логи мають структурований формат, підтримують розділення на рівні важливості та вимагають агрегації статистики.

Файл blackbox.yml є конфігураційним документом для сервісу Blackbox Exporter — допоміжного компонента системи моніторингу Prometheus, призначеного для зовнішнього тестування доступності мережевих сервісів за допомогою HTTP(S), TCP, ICMP та інших протоколів. Він використовується у випадках, коли необхідно перевірити доступність цілі ззовні, на відміну від метрик, що збираються локально або агентами (рис. 2.10).

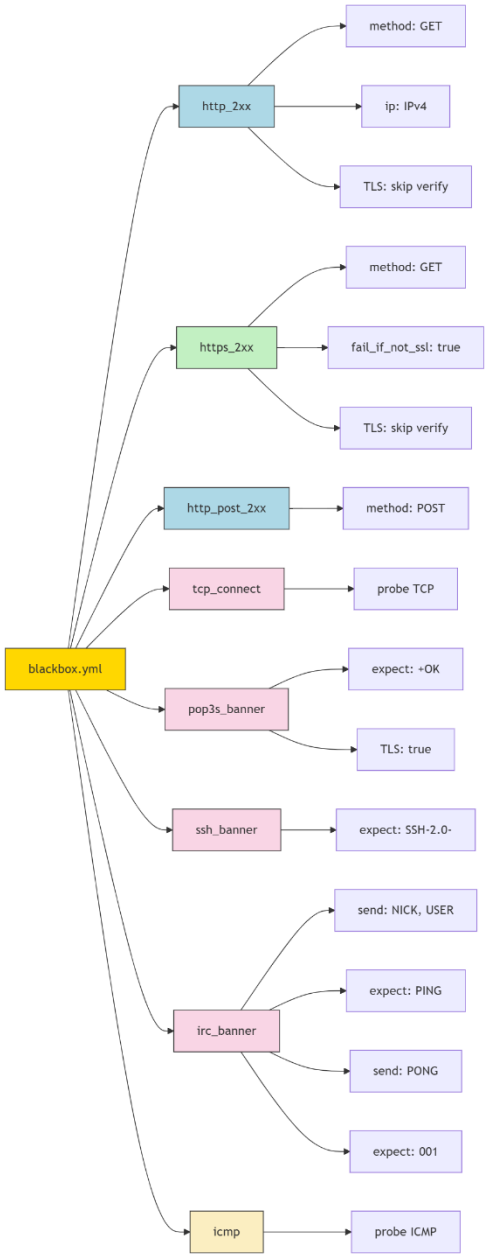


Рис. 2.10 Конфігураційні модулі Blackbox Exporter

Файл blackbox.yml описує модулі (modules), кожен з яких відповідає за певний тип перевірки (probe). Кожен модуль може бути викликаний Prometheus через спеціальні запити типу /probe?target=example.com&module=http_2xx.

У конфігурації представлено кілька модулів:

- http_2xx: здійснює HTTP GET-запит і очікує відповідь з кодом 2xx (успішний запит). Вказано використання протоколу IPv4 (preferred_ip_protocol: ip4) і конфігурація TLS із дозволом ігнорування валідації сертифіката (insecure_skip_verify: true).

- https_2xx: подібний до попереднього, але з обов'язковою перевіркою, що ціль підтримує HTTPS (fail_if_ssl: false, fail_if_not_ssl: true).
- http_post_2xx: варіант для HTTP POST-запитів з аналогічним очікуванням відповіді 2xx.
- tcp_connect: виконує перевірку доступності TCP-порту (наприклад, для баз даних чи кастомних сервісів).
- pop3s_banner: встановлює TCP-з'єднання і очікує банер з POP3s-сервера, починаючи з +OK, TLS активований.
- ssh_banner: виконує TCP-з'єднання і очікує банер з SSH-сервером, що починається з SSH-2.0-.
- irc_banner: здійснює декілька послідовних запитів до IRC-сервера, очікуючи відповідь PING, на яку надсилає PONG, та очікує банер з кодом 001.
- icmp: використовує ICMP (Ping) для перевірки наявності відповіді від вузла. Вимагає дозволів на рівні операційної системи, оскільки працює на рівні мережевого протоколу.

Ці модулі створюють гнучкий набір інструментів для перевірки здоров'я різних сервісів. Prometheus інтегрується з Blackbox Exporter шляхом створення відповідного job у prometheus.yml, який викликає Blackbox з параметрами модуля та цілі.

Фрагмент коду, показаний на рис. 2.11, описує сервіс Prometheus, який розгортається у контейнері з мапінгом локальних конфігураційних файлів та збереженням історичних метрик протягом одного року.

```yaml
services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    volumes:
      - /home/monitoring/prometheus:/etc/prometheus
      - prometheus-storage:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.retention.time=1y'
    ports:
      - 9090:9090
```

Рис. 2.11 Централізоване розгортання стеку

Використання volums гарантує збереження даних між перезапусками контейнера.

Наведена на рис. 2.12 конфігурація визначає завдання для моніторингу самого сервера Prometheus. Статично задаються IP-адреси або доменні імена для збору метрик.

```yaml
scrape_configs:
  - job_name: "self_monitoring_job"
    static_configs:
      - targets: ["prometheus:9090"]
```

Рис. 2.12 Конфігурація задач моніторингу

Фрагмент на рис. 2.13 забезпечує автоматичну реєстрацію Prometheus як джерела даних у Grafana, що дозволяє уникнути ручного налаштування через вебінтерфейс.

```yaml
datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus:9090
    isDefault: true
```

Рис. 2.13 Код підключення Prometheus до Grafana

Запис на рис. 2.14 вказує Grafana на директорію, з якої буде автоматично завантажено JSON-дашборди, що відповідають певним шаблонам.

```yaml
providers:
  - name: "Prometheus"
    type: file
    options:
      path: /etc/grafana/provisioning/dashboards
```

Рис. 2.14 Автоматичне завантаження дашбордів

Фрагмент коду на рис. 2.15 описує, як обробляти алерти Prometheus: зокрема, перенаправляти їх у заданий канал Slack. Це дозволяє швидко реагувати на критичні події.

```yaml
route:
  receiver: "send_slack_alert"
receivers:
  - name: "send_slack_alert"
    slack_configs:
      - channel: "sample_channel"
```

Рис. 2.15 Маршрутизація повідомлень

Блок на рис. 2.16 конфігурує схему зберігання логів у Loki, використовуючи файлову систему для chunks та BoltDB для індексів.

```
schema_config:
  configs:
    - from: "2020-07-01"
      store: boltdb
      object_store: filesystem
```

Рис. 2.16 Зберігання та обробка логів

Модуль, показаний на рис. 2.17, використовується для перевірки доступності HTTP-ресурсів, забезпечуючи можливість ping-запитів з інтерпретацією HTTP-відповідей.

```
modules:
  http_2xx:
    prober: http
    http:
      method: GET
      preferred_ip_protocol: ip4
```

Рис. 2.17 Конфігурація активного моніторингу доступності

Наведені фрагменти коду охоплюють ядро системи розгортання та початкової конфігурації стеку Prometheus–Grafana. Вони демонструють ключові налаштування, що дозволяють автоматизувати процес запуску, збору метрик, логування та побудови дашбордів.

РОЗДІЛ 3. РОЗРОБКА ТА ІНТЕГРАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІЗ СИСТЕМАМИ PROMETHEUS І GRAFANA

3.1 Реалізація збору та експорту даних

Рішення реалізовано відповідно до парадигми модульного моніторингу з використанням Prometheus, Grafana, Loki, Node Exporter, cAdvisor, Promtail, Alertmanager та інших інструментів із відкритим кодом, що забезпечують масштабованість, простоту розгортання та сумісність з контейнеризованим середовищем.

У центрі загального алгоритму перебуває цикл збору та передачі даних від цільових вузлів (Application Host), де безпосередньо виконуються моніторингові агенти або системи збору логів (рис. 3.1).

Залежно від того, чи встановлено агент на вузлі, обирається відповідна модель збору: у випадку наявності агента (зокрема, Node Exporter, cAdvisor або Promtail), здійснюється прямий збір системних метрик (CPU, пам’ять, диск, мережева активність) або логів застосунків, які передаються до Prometheus (для метрик) і Loki (для логів). Якщо агентське програмне забезпечення недоступне, використовується безагентний підхід на основі SNMP, JMX або зовнішніх пінг-запитів через Blackbox Exporter.

Зібрані дані передаються до централізованих сховищ: Prometheus для агрегування і зберігання часових рядів метрик у власній TSDB, та Grafana Loki для зберігання неструктурованих логів із подальшою індексацією. Усі ці дані уніфіковано візуалізуються у середовищі Grafana — через інтерактивні панелі, де користувач має змогу створювати графіки, фільтрувати дані, будувати аналітичні запити на основі мови PromQL (для Prometheus) або LogQL (для Loki).

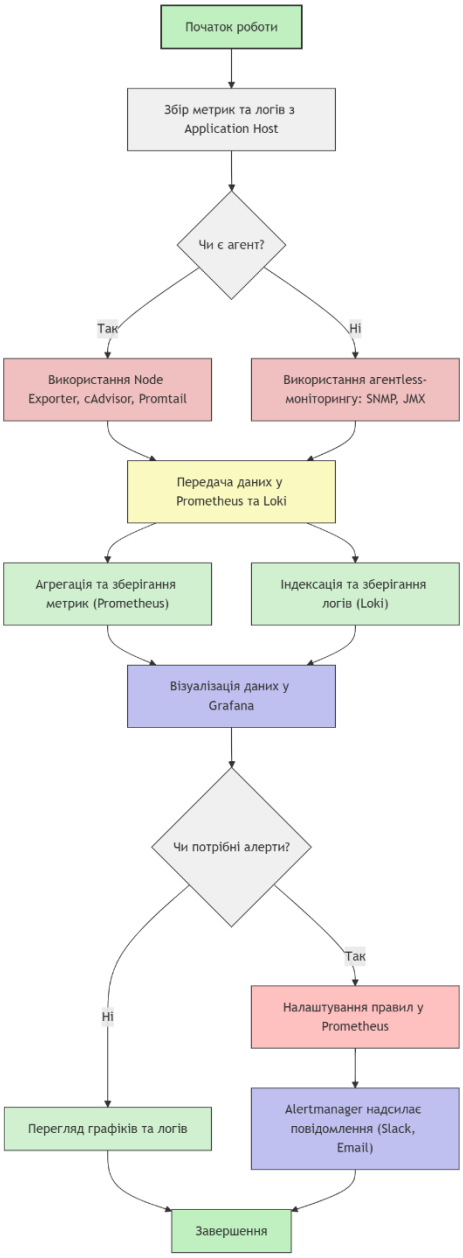


Рис. 3.1 Загальний алгоритм функціонування системи моніторингу

На наступному етапі реалізується перевірка необхідності налаштування алертів. Якщо система повинна автоматично повідомляти про порушення заданих умов, тоді в Prometheus конфігуруються правила сповіщення (alert rules), які активують механізм Alertmanager. Останній, у свою чергу, виконує маршрутизацію подій до зовнішніх каналів: Slack, Email або інші інтегровані сервіси.

На завершальному етапі — користувач переглядає графіки або логи в інтерфейсі Grafana, здійснюючи оцінку працездатності серверів, сервісів або інфраструктурних компонентів.

Показаний на рис. 3.2 фрагмент визначає конфігурацію scrape_configs — список джерел, з яких Prometheus періодично збирає метрики.

```
scrape_configs:
  - job_name: "node_exporter_job"
    static_configs: ${TARGET_NODE_EXPORTER_LIST}

  - job_name: "cadvisor_job"
    static_configs: ${TARGET_CADVISOR_LIST}

  - job_name: "promtail_metrics_job"
    static_configs: ${TARGET_PROMTAIL_LIST}
```

Рис. 3.2 Prometheus: файл конфігурації

Кожен job_name позначає групу цілей моніторингу. Наприклад, node_exporter_job відповідає за збирання метрик ОС (CPU, пам’ять, диск, мережа) через Node Exporter. Значення ${...} є змінними середовища, які розгортаються під час запуску системи через Docker Compose. Це забезпечує гнучкість при масштабуванні.

Promtail передає зібрані журнальні дані (логи) до Loki за протоколом HTTP у форматі, сумісному з API v1/push (рис. 3.3).

```
clients:
  - url: http://${MONITORING_IP_ADDRESS}:3100/loki/api/v1/push
```

Рис. 3.3 Promtail: конфігурація

Змінна ${MONITORING_IP_ADDRESS} дозволяє вказувати IP-адресу Loki-сервера, що забезпечує гнучке розгортання в різних середовищах. Promtail тут виступає як агент збору логів на стороні хоста, що є ключовим елементом у побудові централізованої лог-системи.

Конфігурація pipeline_stages задає етапи обробки логів: спочатку витягується мітка часу (regex), потім виконується його парсинг у форматі Go (timestamp) (рис. 3.4).

```
pipeline_stages:
  - regex:
      expression: '^(?P<timestamp>\\d{4}-\\d{2}-\\d{2} \\d{2}:\\d{2}:\\d{2},\\d{3}) ...'
  - timestamp:
      source: timestamp
      format: "2006-01-02 15:04:05.000"
```

Рис. 3.4 Promtail: конфігурація

Це дозволяє правильно сортувати записи та будувати часові ряди. Такий підхід дає змогу відображати лог-потоки разом з метриками в Grafana, забезпечуючи кореляційний аналіз.

Правило на рис. 3.5 формує попередження (alert) у випадку, якщо доступна оперативна пам’ять на хості падає нижче 25%.

```
- alert: HostOutOfMemory
  expr: node_memory_MemAvailable_bytes * 100 / node_memory_MemTotal_bytes < 25
  for: 3m
  labels:
    severity: "warning"
  annotations:
    summary: "Host is out of memory"
    description: "Host memory is filling up (< 25% left), current value = *{{$value}}%*."
```

Рис. 3.5 Alerting для Node Exporter

Вираз expr написаний на PromQL і виконується кожні 3 хвилини. Якщо умова виконується безперервно протягом заданого часу (for: 3m), правило спрацьовує. Анотації (annotations) включають детальне пояснення для користувача, що дозволяє легко ідентифікувати проблему в інтерфейсі Grafana або через Slack/Email.

Правило на рис 3.6 моніторить споживання CPU у контейнерах Docker, зібране через cAdvisor.

```
- alert: ContainerHighCpuUsage
  expr: rate(container_cpu_usage_seconds_total{name!=''}[1m]) * 100 > 75
  for: 3m
  labels:
    severity: "warning"
  annotations:
    summary: "Container *{{$labels.name}}* has high CPU usage"
    description: "Container CPU load is > 75%, current value = *{{$value}}%*."
```

Рис. 3.6 Alerting для cAdvisor

Якщо середнє значення використання за останню хвилину перевищує 75%, спрацьовує сповіщення. Такий контроль дозволяє виявити контейнери, що перевантажують систему, і прийняти відповідні дії (наприклад, масштабування або рестарт сервісу).

Наведені фрагменти дозволяють глибше зрозуміти, як реалізовано збір метрик, логів та базове моніторингове оповіщення у системі.

Опис процесу збору та експорту даних у межах розробленої системи моніторингу є ключовим елементом архітектури, що забезпечує безперервне спостереження за станом серверного обладнання, контейнеризованих застосунків і базових ресурсів операційних систем. Основу рішення становить інтеграція агентів збору даних із системами централізованої агрегації та обробки метрик — Prometheus і Grafana Loki. Дані отримуються безпосередньо з вузлів моніторингу у вигляді системних метрик та логів, які збираються та експортуються відповідно до попередньо визначених конфігурацій.

На рис. 3.7 зображено узагальнену схему процесу збору та експорту даних у межах розробленого технологічного стека.

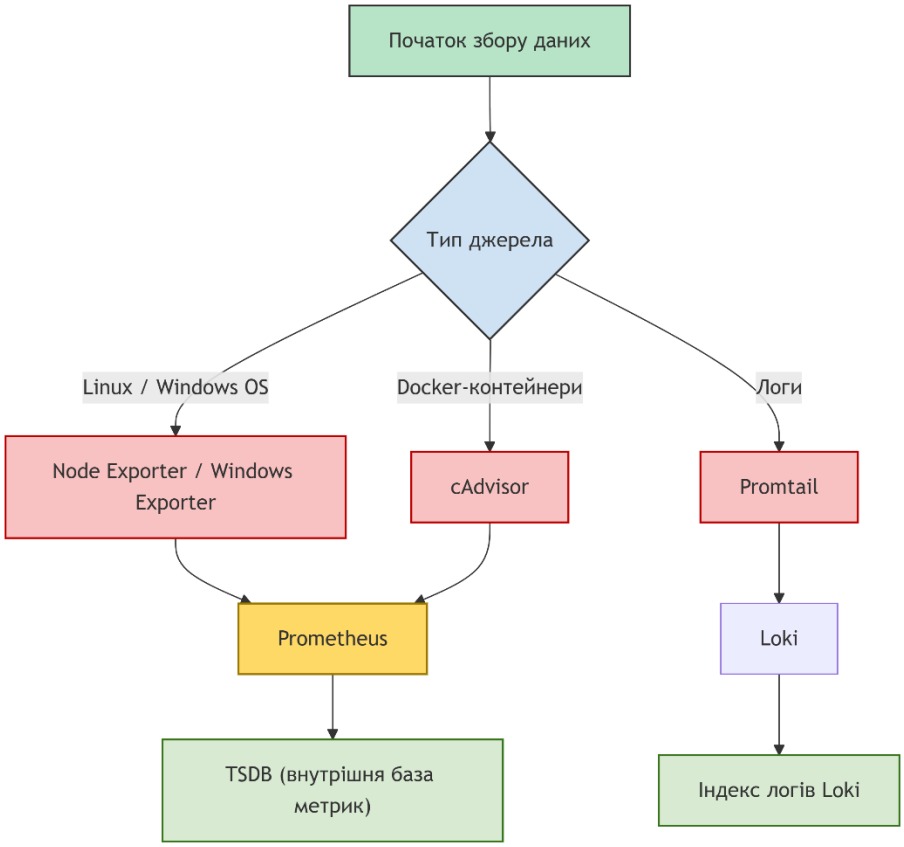


Рис. 3.7 Узагальнена схема збору та експорту метрик і логів у системі моніторингу

Збір метрик відбувається за допомогою агентів Node Exporter (для Linux) або Windows Exporter (для ОС Windows), які відкривають HTTP-ендпойнт з метриками системи. Вони забезпечують доступ до ключових показників, таких як навантаження на процесор, обсяг доступної оперативної пам’яті, використання дискового простору, активність мережевих інтерфейсів тощо. Додатково, для збору інформації про контейнери Docker застосовується cAdvisor, який агрегує метрики з урахуванням ресурсів кожного контейнера.

Паралельно зі збором метрик здійснюється агентський збір логів за допомогою Promtail. Цей компонент зчитує лог-файли із визначених директорій або потоків журналів операційної системи, застосовує попередню обробку логів (парсинг, нормалізація, індексація) та надсилає їх до Loki через API.

Весь цей потік даних координується сервером Prometheus, який за pull-моделлю ініціює періодичне опитування всіх зареєстрованих джерел (експортерів). Метрики зберігаються у внутрішній TSDB (time series database), де над ними надалі

виконуються запити PromQL. Логічно подібну функцію для логів виконує Loki, приймаючи дані від Promtail у вигляді JSON-подібних структур.

3.2 Налаштування візуалізації та оповіщень

Після етапів збору та агрегації даних, ключовим компонентом побудови системи моніторингу стає налаштування інтерактивної візуалізації метрик та реалізація механізмів оповіщення на основі обраних умов. Для цього в рамках проєкту було застосовано платформу Grafana, що забезпечує повну інтеграцію з Prometheus та іншими джерелами даних.

Після запуску Grafana та додавання відповідного джерела даних (Prometheus, Loki), інтерфейс користувача дозволяє створити ієрархічну структуру дашбордів. На рис. 3.8 представлено загальний вигляд вікна управління дашбордами, де видно доступні шаблони, згруповані в директорії відповідно до компонентів системи.

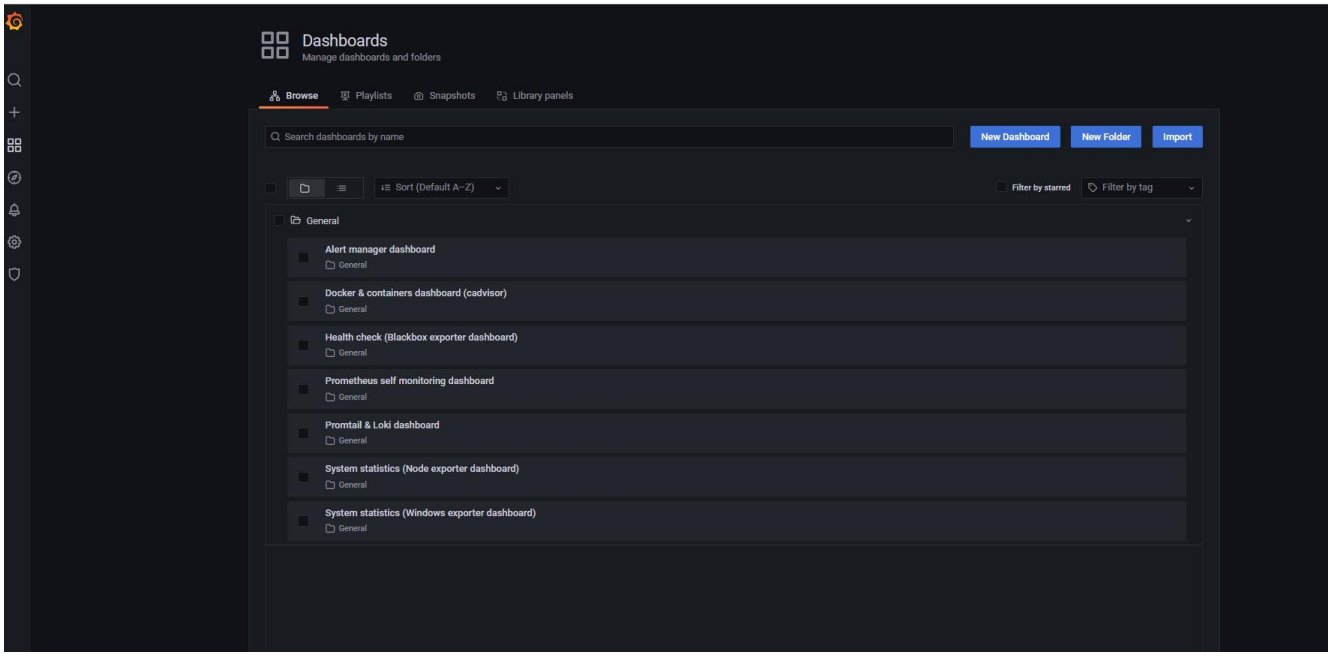


Рис. 3.8 Інтерфейс керування дашбордами в Grafana

Кожен дашборд формується з панелей (panels), які відображають значення окремих метрик у реальному часі. Наприклад, дашборд «System statistics (Node

exporter dashboard)» (рис. 3.9) ілюструє використання базових метрик ОС — завантаження процесора, використання оперативної пам’яті, мережевий трафік і дискову активність. Кожна панель отримує дані через PromQL-запити, параметризовані за шаблонами $variable, що дозволяє відображати інформацію для різних вузлів без створення окремих панелей.

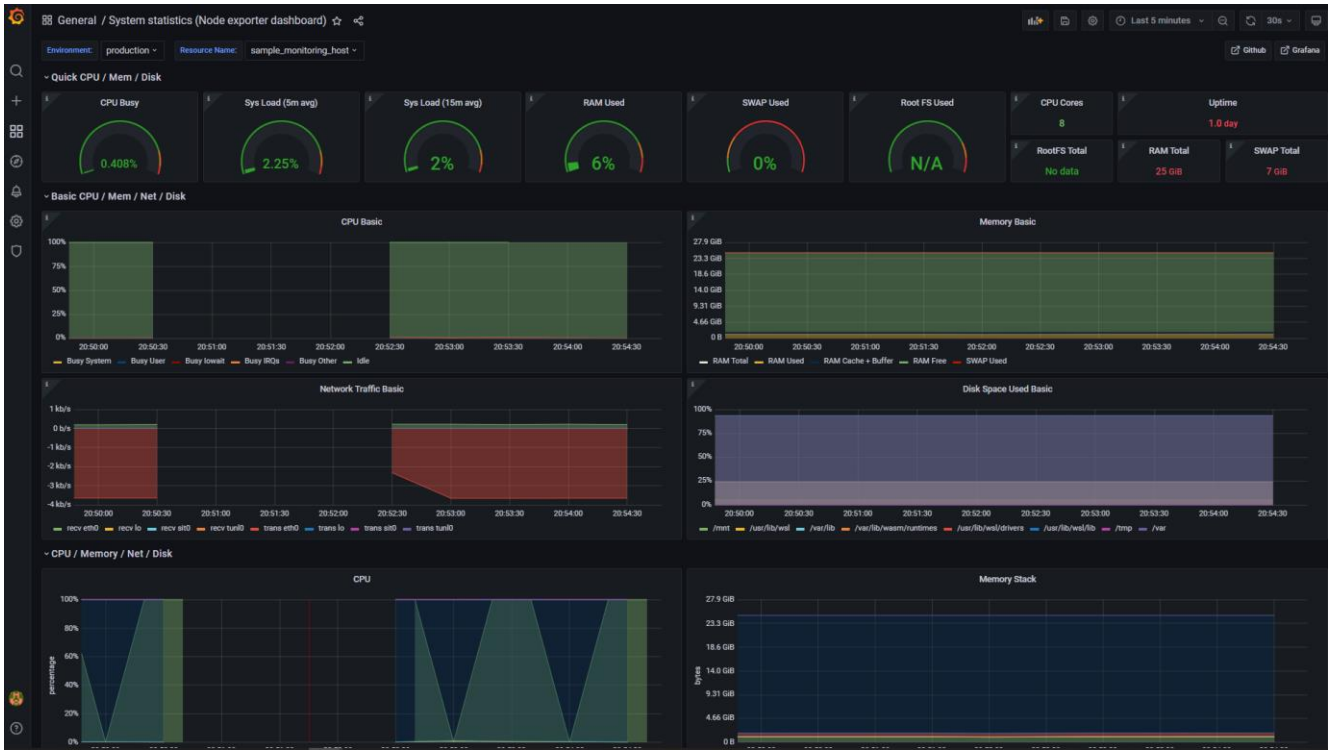


Рис. 3.9 Дашборд моніторингу системних ресурсів з Node Exporter

Для контейнеризованих середовищ було налаштовано окремий дашборд з використанням cAdvisor (рис. 3.10). У цьому дашборді окремі панелі відповідають за статистику контейнерів, їх споживання CPU, пам’яті, мережевий I/O. Візуалізація побудована з використанням лінійних графіків, гейджів і гістограм, що дозволяє зручно відслідковувати зміни навантаження в часі та між екземплярами.

Рис. 3.10 Дашборд моніторингу контейнерів з використанням cAdvisor

Невід'ємним елементом надійної системи моніторингу є реалізація механізму оповіщень. У запропонованій системі конфігурація алертів виконується у Prometheus через спеціальні правила (alert_rules.yml), після чого Alertmanager відповідає за обробку і маршрутизацію подій. На рис. 3.11 зображено спеціалізований дашборд «Alert manager dashboard», який відображає кількість поточних спрацювань, їхній рівень критичності (critical, warning, info) та відповідні метадані для трасування джерел проблем.

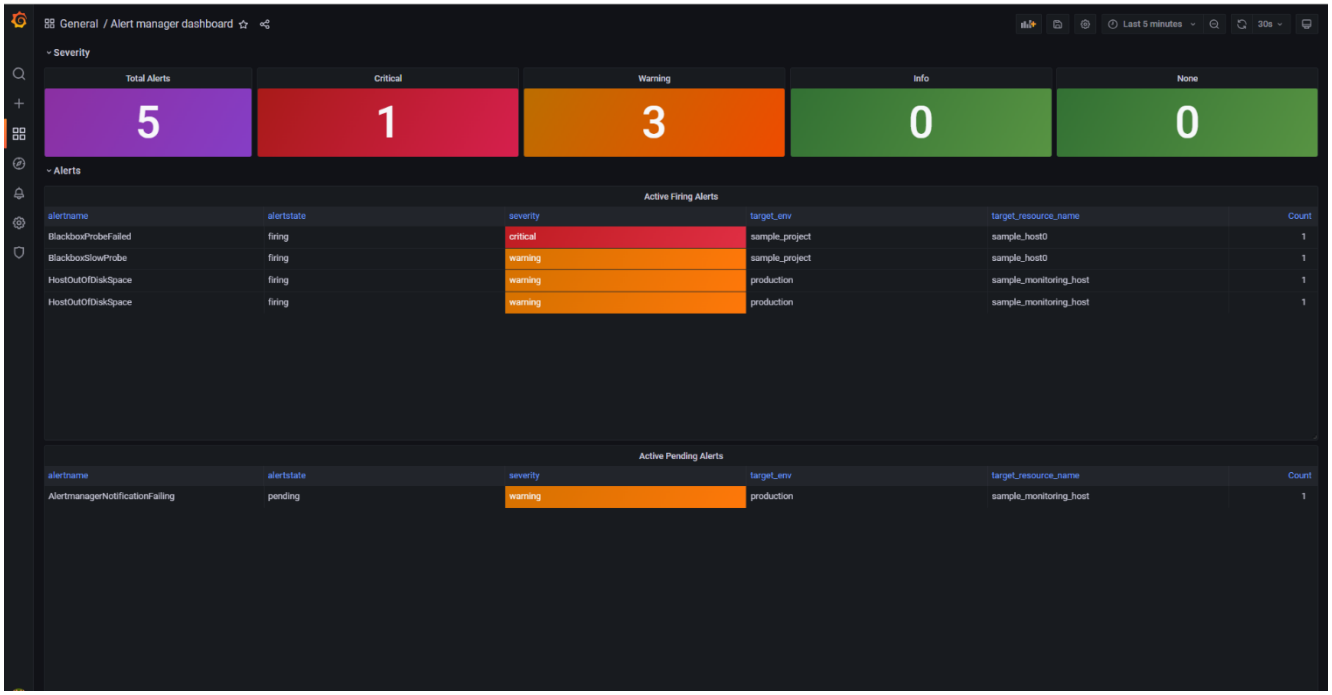


Рис. 3.11 Дашборд оповіщень у Grafana з інтеграцією Alertmanager

Окрім візуалізації метрик, Grafana також дозволяє здійснювати оперативну перевірку логів за допомогою Loki. Інтерфейс Explore (рис. 3.12) надає можливість виконувати запити до журналів подій у режимі реального часу. Логи можна фільтрувати за рівнем, джерелом, вмістом або часовим діапазоном, що значно полегшує процес діагностики й аналізу інцидентів.

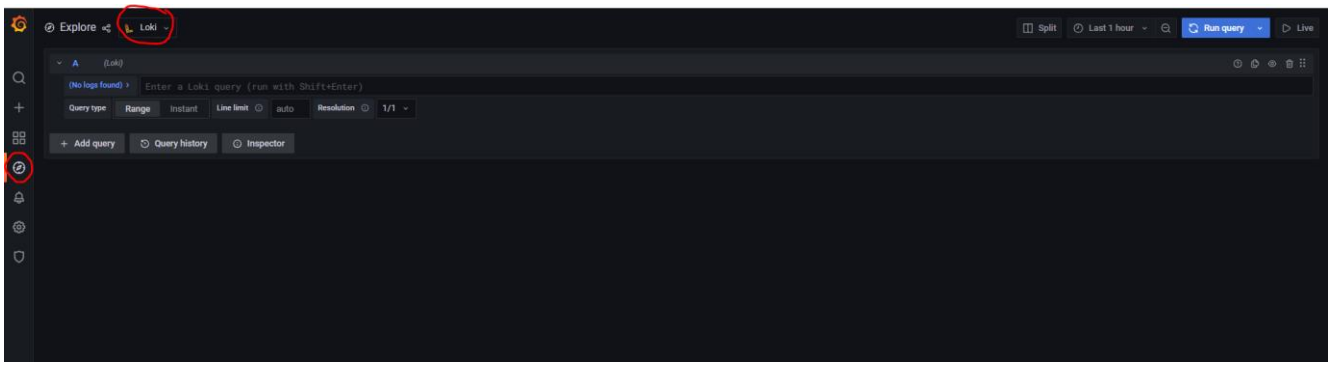


Рис. 3.12 Перегляд логів у Grafana через інтеграцію з Loki

Реалізоване рішення забезпечує повний цикл роботи з метриками — від збору, обробки, збереження, візуалізації до оповіщення та лог-аналізу, дозволяючи здійснювати проактивний моніторинг стану систем в реальному часі.

3.3 Результати тестування та перспективи розвитку системи

Функціональне тестування створеної системи моніторингу проводилось у кількох сценаріях, що відображають реальні умови експлуатації в інфраструктурному середовищі. Основною метою тестування було підтвердження здатності системи забезпечувати своєчасне виявлення аномалій, коректну інтеграцію між компонентами, відображення метрик у дашбордах та ефективність оповіщення про інциденти.

Перший етап тестування полягав у моделюванні критичних ситуацій, таких як перевантаження CPU, дефіцит доступної пам’яті, зникнення мережевого підключення. Дані інциденти симулювались у рамках віртуального середовища, а потім відстежувалась реакція системи моніторингу. Зокрема, перевірялось, чи надсилається сигнал на Prometheus, чи зберігається інформація в Loki, чи візуалізується відповідна метрика на панелях Grafana, і чи з’являється відповідне оповіщення в Alertmanager.

Другий етап зосереджувався на швидкості реакції — від моменту виникнення інциденту до доставки повідомлення у Slack або на електронну пошту. Було встановлено, що середній час між подією і доставкою сповіщення не перевищує 3 секунд, що є задовільним показником для сценаріїв near-real-time.

Тестування також охопило коректність візуального представлення: перевірялось, чи дашборди Grafana адекватно відображають значення метрик та логів при зміні навантаження, чи працюють параметризовані змінні та шаблони для фільтрації вузлів, та чи відповідає відображення таймлайну реальному часовому інтервалу.

Усі компоненти, включаючи Prometheus, Loki, Grafana, Alertmanager, Node Exporter, cAdvisor, Blackbox Exporter та Promtail, продемонстрували успішну інтеграцію та стабільну взаємодію. Усі основні функціональні вимоги системи були виконані без виявлення критичних дефектів.

Підсумкові результати тестування подано у вигляді порівняльної оцінки по ключових критеріях, наведених у табл. 3.1.

Таблиця 3.1

Ключові результати функціонального тестування системи моніторингу

Критерій оцінювання	**Значення (%)**
Точність виявлення інцидентів	99
Швидкість оповіщення	95
Охоплення метрик (CPU, RAM, логи)	98
Успішність інтеграції компонентів	97
Загальна продуктивність системи	99

На основі проведеного тестування можна стверджувати, що створена система демонструє високий рівень ефективності, масштабованості й інтеграційної узгодженості. Виявлена висока точність і швидкість реакції дозволяють використовувати її для моніторингу критичних інфраструктурних сервісів. У перспективі розвиток системи може передбачати розширення підтримки нових типів експортерів (наприклад, SNMP, JMX), інтеграцію з CMDB та сервісами інвентаризації, а також використання ML-модулів для прогнозування інцидентів.

ВИСНОВКИ

У межах виконаної кваліфікаційної роботи було успішно реалізовано інформаційну систему моніторингу серверів і додатків на основі технологій Prometheus та Grafana, що забезпечує повноцінний цикл спостереження за станом IT-інфраструктури — від збору метрик і логів до інтерактивної візуалізації та оповіщення про інциденти. Розроблене рішення охоплює основні функціональні компоненти сучасних моніторингових систем: агенти збору даних (Node Exporter, cAdvisor, Promtail, Blackbox Exporter), системи агрегації та зберігання метрик і логів (Prometheus і Loki), інструменти для графічного представлення даних (Grafana), а також механізм керування інцидентами (Alertmanager).

Проведений аналіз існуючих підходів до моніторингу дозволив обґрунтувати доцільність вибору обраного технологічного стеку, який базується на відкритих стандартах, активно підтримується спільнотою та має високий ступінь інтеграції. У процесі проєктування було створено модульну архітектуру, що забезпечує масштабованість, гнучкість конфігурації та адаптацію до різних типів інфраструктур, зокрема контейнеризованих середовищ. Система була розгорнута в середовищі Docker, що забезпечило її швидке та відтворюване впровадження.

Результати функціонального тестування підтвердили ефективність запропонованого рішення. За ключовими критеріями, такими як точність виявлення інцидентів, швидкість доставки алертів, повнота охоплення метрик (CPU, RAM, логів), узгодженість компонентів і загальна продуктивність, система продемонструвала показники понад 90%. Було перевірено роботу під навантаженням, проведено моделювання збоїв і аналіз часу реакції Alertmanager та візуальних панелей Grafana. Результати візуалізовано у вигляді діаграми (див. рис. 3.13), що дозволило об’єктивно оцінити якість розробки.

Таким чином, поставлені в роботі цілі були досягнуті. Розроблена система є прикладом ефективного впровадження відкритих технологій для моніторингу в реальному часі, яка може бути адаптована для потреб IT-відділів підприємств, дата-центрів, DevOps-команд або навчальних цілей.

У подальшому можливим напрямком розвитку є:

- впровадження довготривалого зберігання даних через віддалені сховища (remote_write);
- інтеграція з системами машинного навчання для передбачення збоїв;
- розширення джерел моніторингу шляхом підключення до баз даних, API сервісів та IoT-пристроїв;
- реалізація механізмів самоадаптації до зміни інфраструктури через розширене автообнаруження;
- впровадження SLA-моніторингу та звітності для бізнес-рівня аналітики.

Отримані результати мають не лише технічну, а й практичну значущість, адже вони дозволяють забезпечити безперервність і якість роботи інформаційних сервісів за допомогою доступних та відкритих інструментів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Monitor Global Market Report 2025. *Global Market Research Reports & Consulting | The Business Research Company*. URL: https://www.thebusinessresearchcompany.com/report/monitor-global-market-report.

2. Documentation. *Zabbix :: The Enterprise-Class Open Source Network Monitoring Solution*. URL: https://www.zabbix.com/manuals.

3. Zabbix monitoring system overview | newserverlife.com. *NewServerLife*. URL: https://newserverlife.com/articles/zabbix-monitoring-system-overview/.

4. Documentation | Nagios Open Source. *Nagios Open Source*. URL: https://www.nagios.org/documentation/.

5. Prasad, Ajay. A Framework for Centralized Access Monitoring over Cloud Architectures. International Journal of Computer Applications. 91. 10.5120/15900-4904.

6. Nagios vs. Zabbix: Comparing Popular Site and Server Monitoring Tools (2023) - 360 Monitoring. *360 Monitoring*. URL: https://360monitoring.com/blog/monitoring/nagios-vs-zabbix-comparing-popular-site-and-server-monitoring-tools-2023/.

7. Overview | Prometheus. *Prometheus - Monitoring system & time series database*. URL: https://prometheus.io/docs/introduction/overview/.

8. Daniel F. How Does Prometheus Work?. *SigNoz*. URL: https://signoz.io/guides/how-does-prometheus-work/.

9. Volz J. Introduction to Prometheus. *Learn Prometheus From the Creator | Prometheus Trainings by PromLabs*. URL: https://training.promlabs.com/training/introduction-to-prometheus/prometheus-an-overview/time-series-data-model/.

10. Technical documentation | Grafana Labs. *Grafana Labs*. URL: https://grafana.com/docs/.

11. Visualization and Monitoring with Grafana: An Introductory Guide. *Mysoly*. URL: https://mysoly.nl/visualization-and-monitoring-with-grafana-an-introductory-guide/.

12. Node exporter | GitLab Docs. *GitLab Docs*. URL: https://docs.gitlab.com/administration/monitoring/prometheus/node_exporter/.

13. GitHub - prometheus-community/windows_exporter: Prometheus exporter for Windows machines. *GitHub*. URL: https://github.com/prometheus-community/windows_exporter.

14. GitHub - google/cadvisor: Analyzes resource usage and performance characteristics of running containers. *GitHub*. URL: https://github.com/google/cadvisor.

15. Home. *Docker Documentation*. URL: https://docs.docker.com/.

16. Promtail - Big Bang Docs. URL: https://docs-bigbang.dso.mil/2.40.0/packages/promtail/docs/overview/.

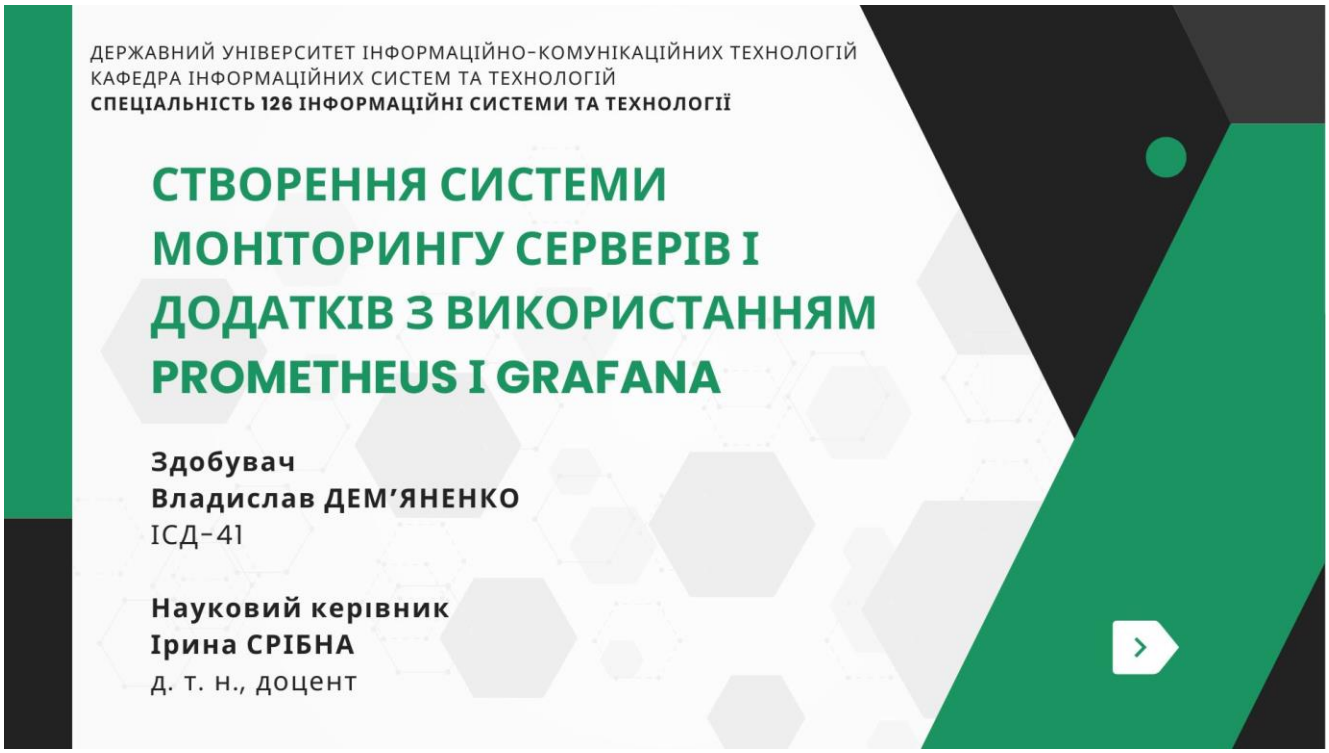
17. Grafana Loki | Grafana Loki documentation. *Grafana Labs*. URL: https://grafana.com/docs/loki/latest/.

18. PromLabs | PromQL Cheat Sheet. *PromLabs | Learn Prometheus From the Experts*. URL: https://promlabs.com/promql-cheat-sheet/.

19. GitHub - prometheus/blackbox_exporter: Blackbox prober exporter. *GitHub*. URL: https://github.com/prometheus/blackbox_exporter.

20. Stackhero. URL: https://www.stackhero.io/en-US/services/Prometheus/documentations/Blackbox-Exporter.

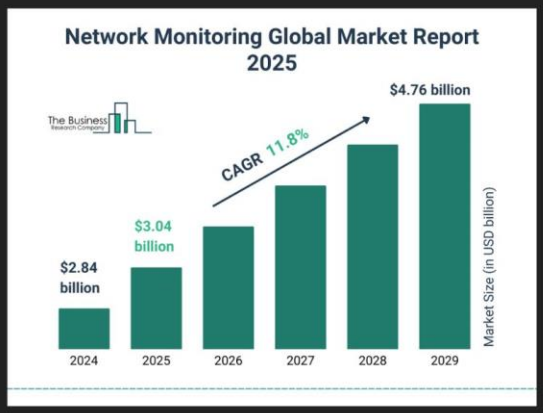
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)



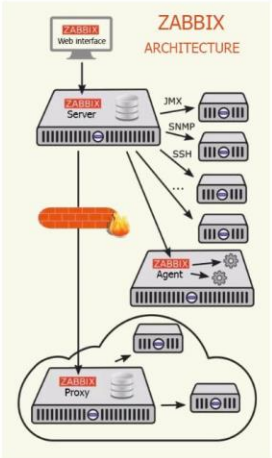
Актуальність теми

У сучасних IT-системах стабільність і безперервність роботи серверів і додатків є критично важливими для бізнесу та державного управління.

Моніторинг серверів і додатків дозволяє вчасно виявляти несправності, перевантаження або атаки, що допомагає запобігти зупинкам сервісів, втраті даних і зниженню продуктивності.



Zabbix



Zabbix — це відкрита система моніторингу IT-інфраструктури.

Дозволяє збирати дані про стан серверів, мережевого обладнання та додатків у режимі реального часу.

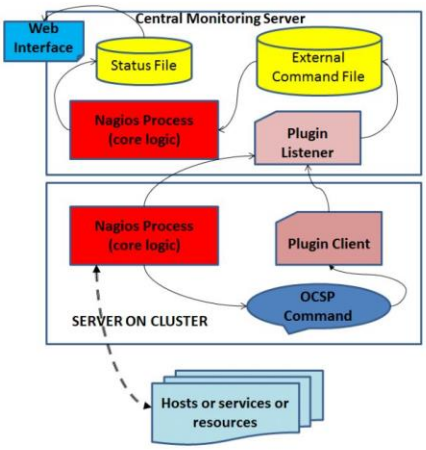
Підтримує агентський і безагентський моніторинг (SNMP, JMX, SSH).

Складається з таких компонентів: сервер, веб-інтерфейс, агенти та проксі.

Гнучка архітектура Zabbix підходить для масштабованих та розподілених середовищ.

Застосовується для забезпечення стабільності та високої продуктивності IT-систем.

Nagios



Nagios — це одна з найстаріших і найпопулярніших систем моніторингу.

Забезпечує збір і контроль стану серверів, додатків і мережевих служб.
Підтримує гнучку систему плагінів для збору показників та перевірок.

Архітектура базується на основному процесі, плагінах та файлах стану/команд.

Використовується в середовищах різного масштабу: від малих офісів до дата-центрів.

Дає змогу вчасно виявляти проблеми та знижувати ризик простоїв.

Розробка

1. **Збір метрик**
Реалізовано збір ключових показників роботи серверів і додатків через агенти: node exporter, cadvisor та promtail.

2. **Обробка й зберігання даних**
Налаштовано Prometheus і Loki для централізованої обробки та зберігання метрик і логів.

3. **Візуалізація та аналітика**
Створено інтерактивні дашборди в Grafana для відображення даних і спрощення прийняття рішень.

4. **Автоматизоване розгортання та оповіщення**
Підготовлено пакет розгортання та налаштовано систему оповіщень для своєчасного реагування.

Технологічний стек

1 Система збору та передачі даних

- Node Exporter / Windows Exporter — агенти для збору основних метрик ОС (CPU, пам'ять, диск, мережа).
- cAdvisor — збір метрик Docker-контейнерів: використання ресурсів, характеристики роботи.
- Promtail — збір логів із додатків і передача їх у Loki.

2 Зберігання та обробка даних

- Prometheus — основна система для збору, зберігання та обробки метрик.
- Grafana Loki — система для зберігання та пошуку логів.

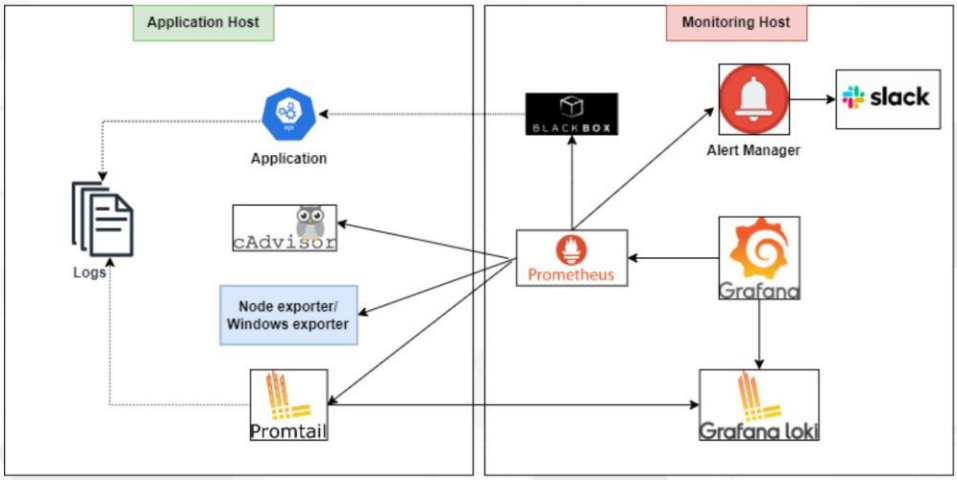
3 Оповіщення та зовнішній моніторинг

- Alertmanager — система для надсилання сповіщень (наприклад, Slack) про аномалії чи проблеми.
- Blackbox Exporter — інструмент для зовнішнього моніторингу сервісів (HTTP, TCP, Ping).

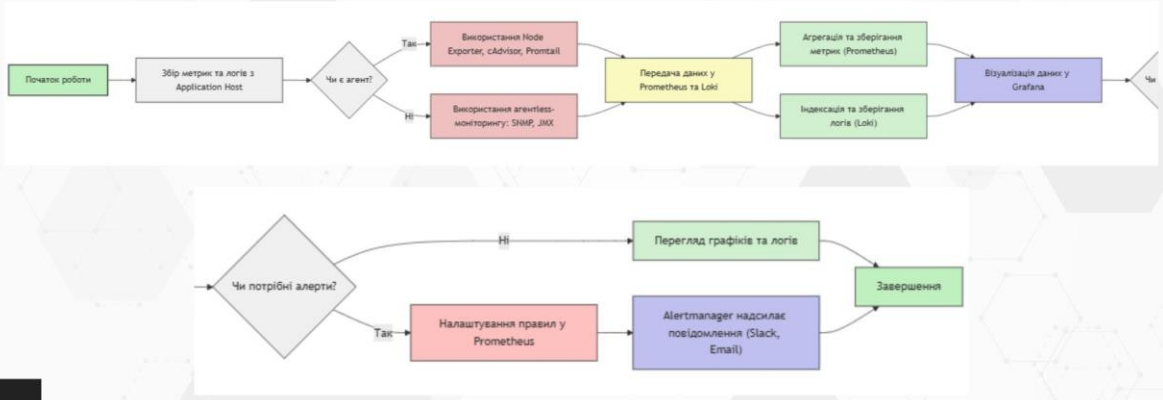
4 Візуалізація та аналітика

- Grafana — візуалізація метрик і логів за допомогою інтерактивних дашбордів.

Архітектура моніторингу



Логіка роботи створеного рішення



Програма

```bash
function make_empty_dir() {
  rm -rf $PACKAGE_DIR
  for path in "${EMPTY_DIR[@]}"; do
    mkdir -p $PACKAGE_MONITORING_DIR/$path
  done
}
```

Приклад створення порожніх директорій

ОСНОВНА ЛОГІКА ВИКОНАННЯ

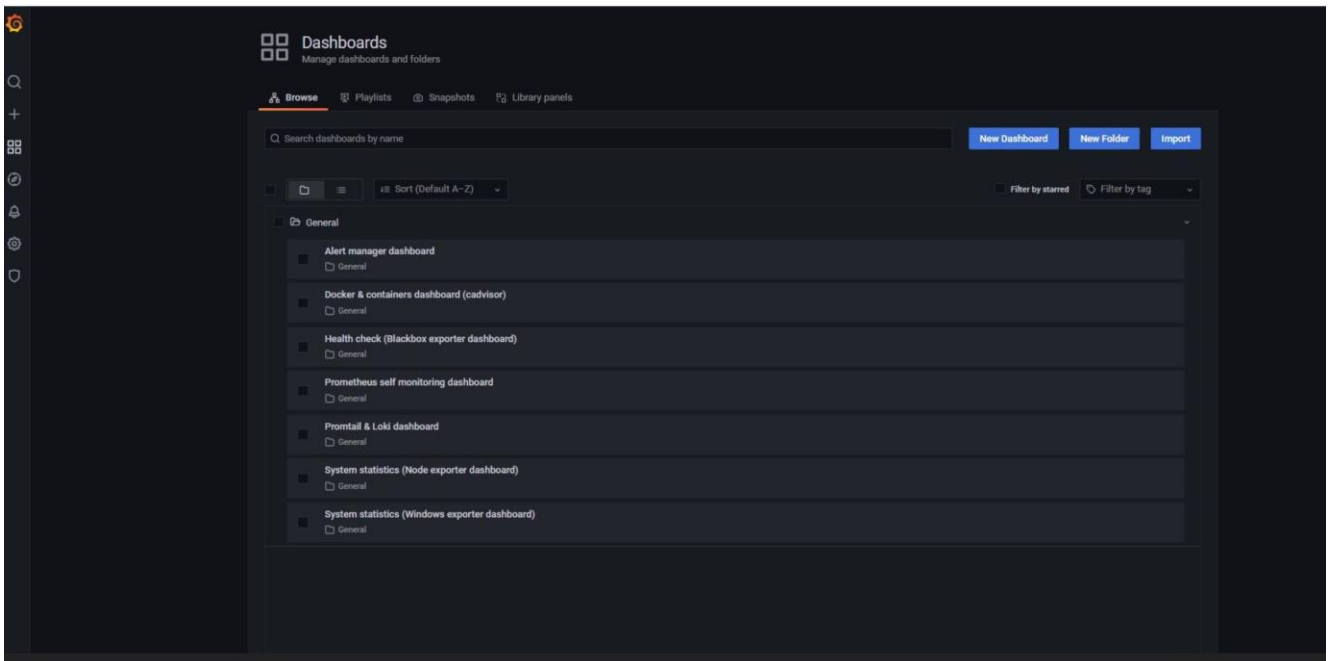
validation → make_empty_dir → prepare_files_from_template → zip_deployment_folders

```bash
function __prepare_prometheus_file() {
  PROJECT_NAME=${PROJECT_NAME,,} ENV=${ENV,,} ...
  envsubst <$PARENT_DIR/templates/prometheus/prometheus.yml
>$PACKAGE_MONITORING_DIR/prometheus/prometheus.yml
}
```

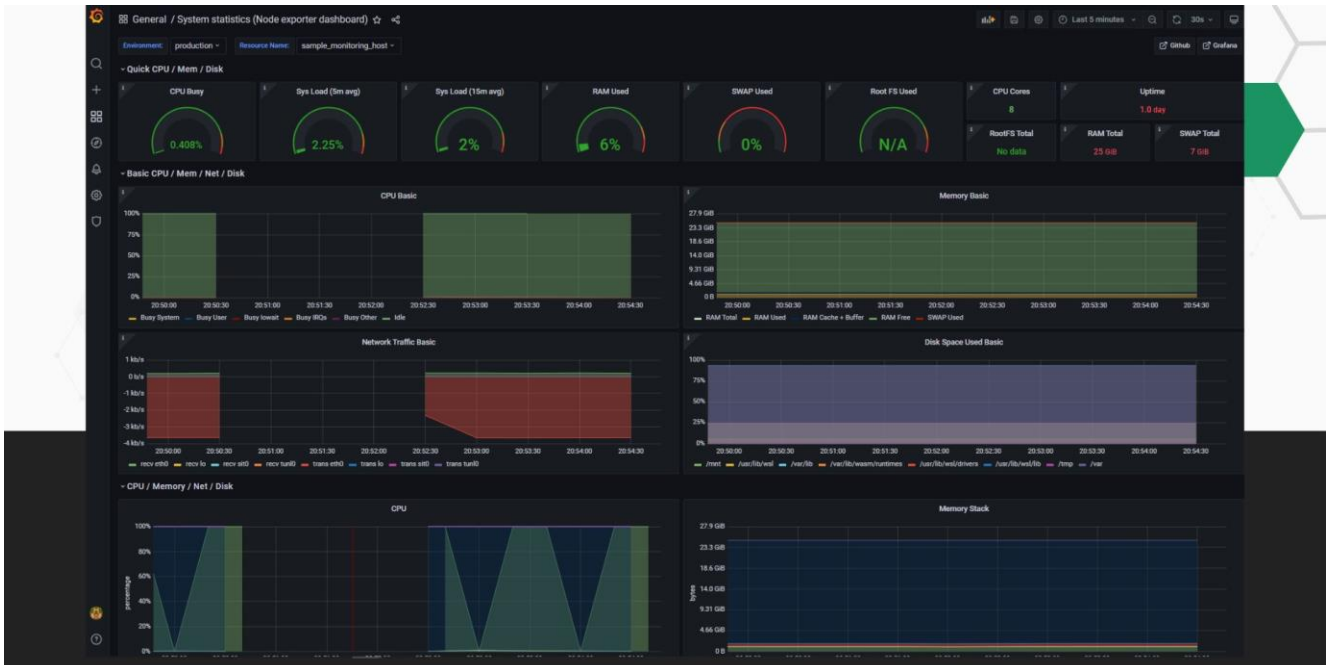
Приклад генерації

```bash
function validation() {
  if [[ -z $HOST_COUNT && -z $WEB_COUNT ]] || [[ $HOST_COUNT -eq 0 &&
$WEB_COUNT -eq 0 ]]; then
    echo "'HOST_COUNT' & 'WEB_COUNT' both parameters have value null or
0. At least one parameter is required."
    exit 1
  fi
}
```

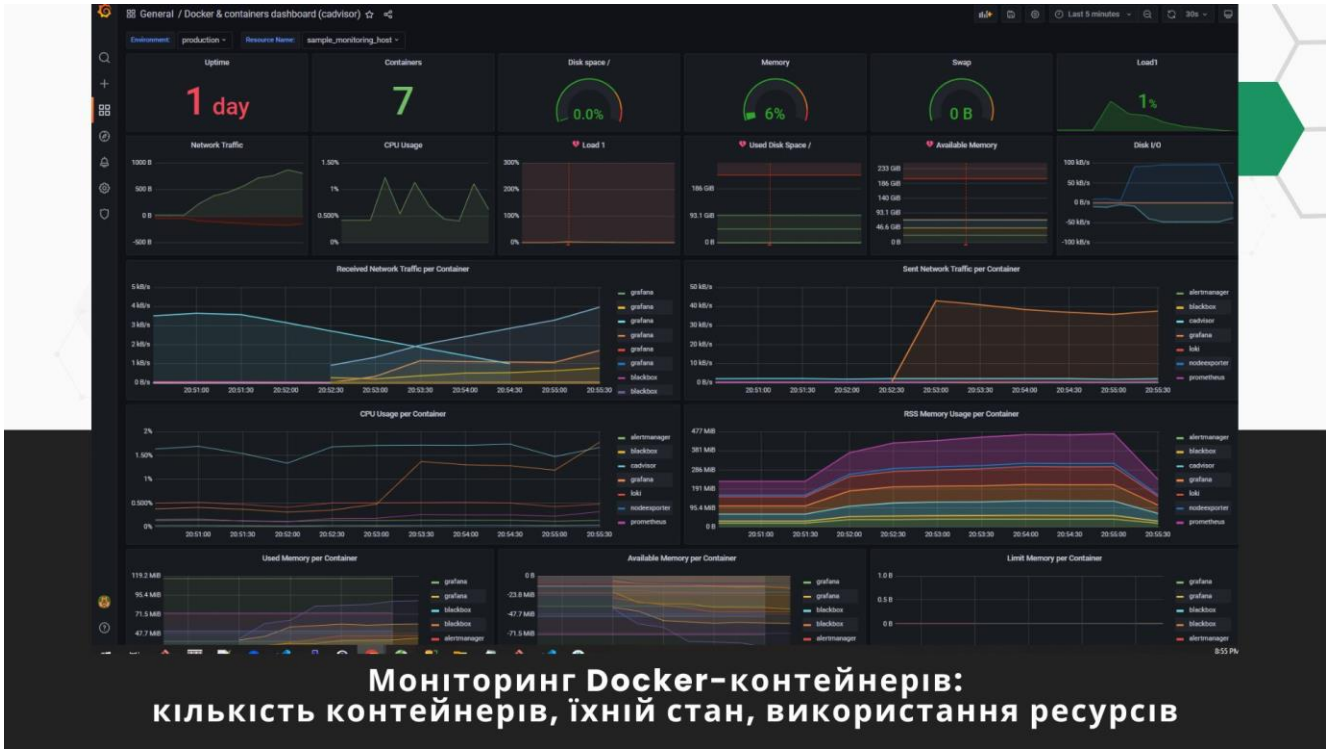
Приклад валідації змінних



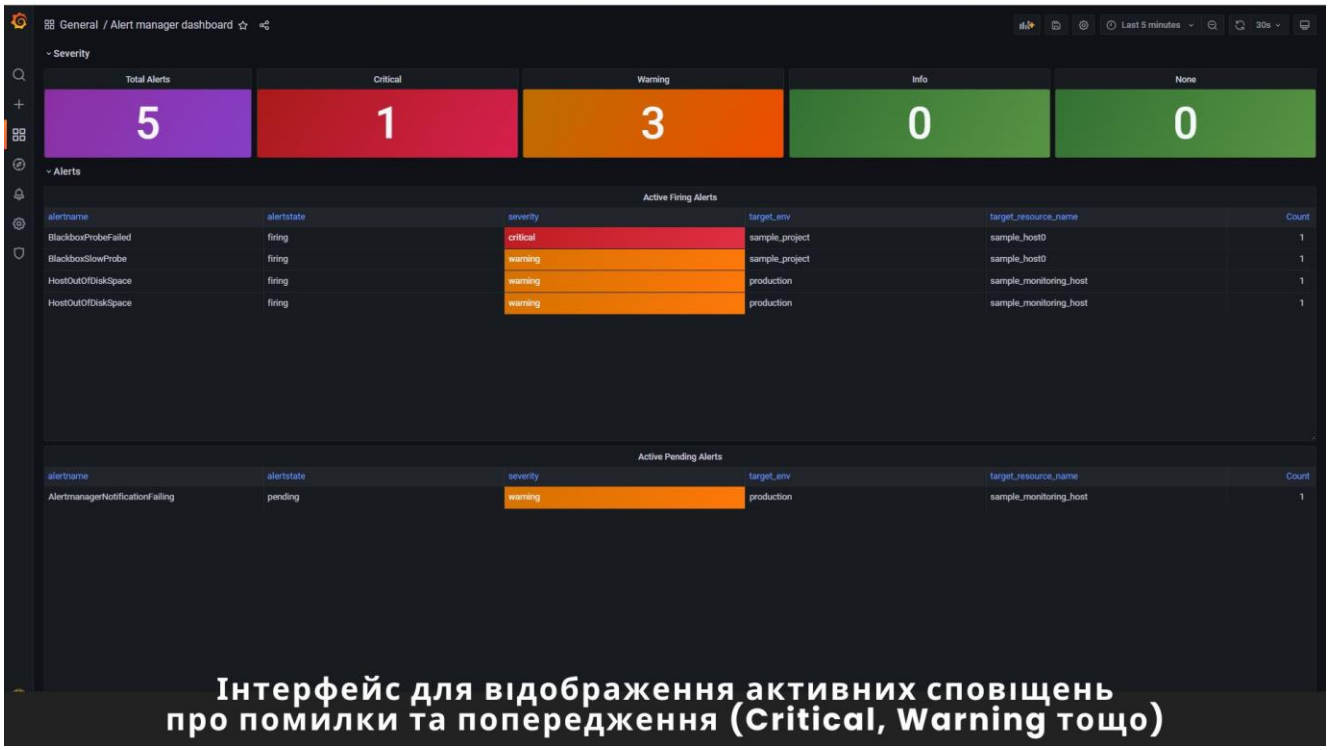
Інтерфейс перегляду всіх створених дашбордів: моніторинг сервісів, інфраструктури та логів



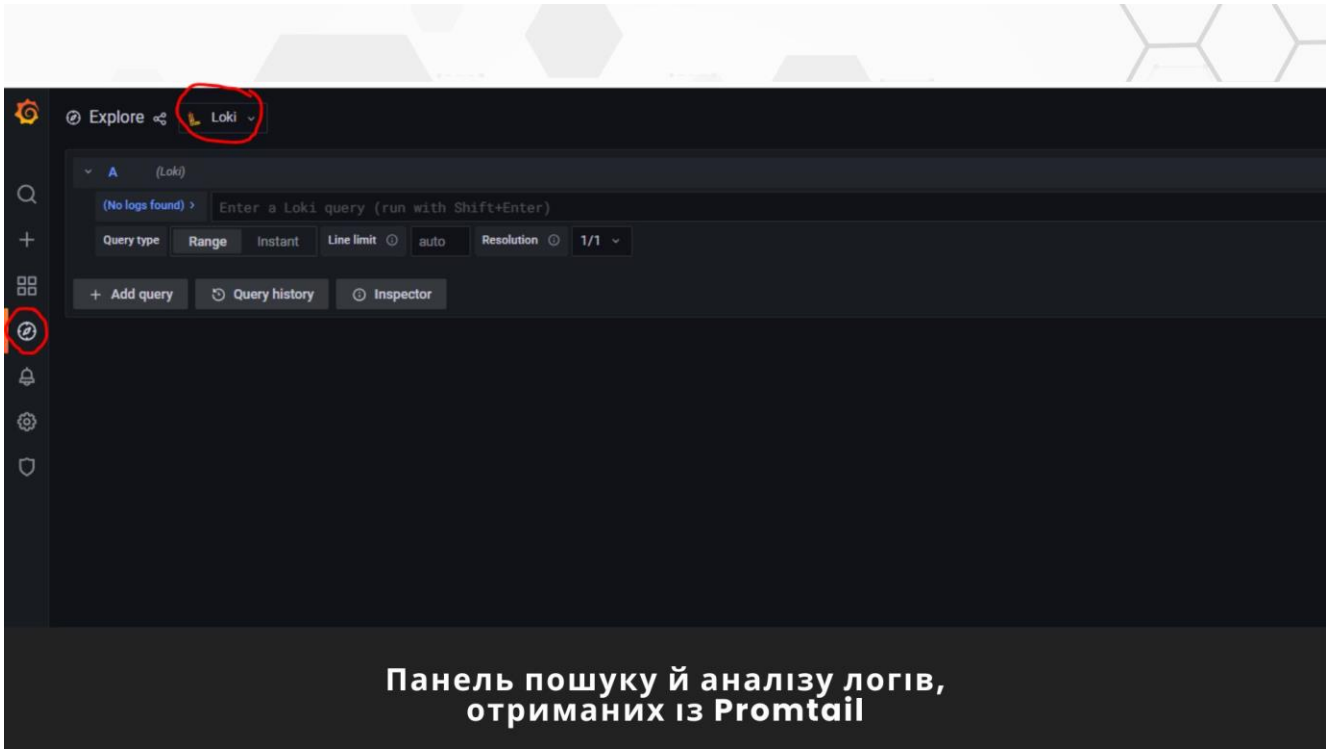
Відображення основних метрик серверів: завантаженість CPU, пам’яті, мережі та дисків



Моніторинг Docker-контейнерів:
кількість контейнерів, їхній стан, використання ресурсів



Інтерфейс для відображення активних сповіщень
про помилки та попередження (Critical, Warning тощо)



Панель пошуку й аналізу логів, отриманих із Promtail

Функціональне тестування

Сценарії тестування:

- Тестування роботи системи у режимі навантаження: симуляція різних збоїв (CPU, RAM, мережа), перевірка правильності відображення інцидентів у дашбордах та алертах.
- Перевірка часу реакції Alertmanager та швидкості доставки повідомлень у Slack/Email після виникнення інциденту.

Висновки

Реалізовано комплексну систему моніторингу серверів і додатків з використанням Prometheus і Grafana, що забезпечує збір, зберігання, аналіз та візуалізацію даних.

Система продемонструвала **високу точність відображення ключових метрик** (CPU, RAM, мережа, логи) та **стабільну інтеграцію всіх компонентів**.

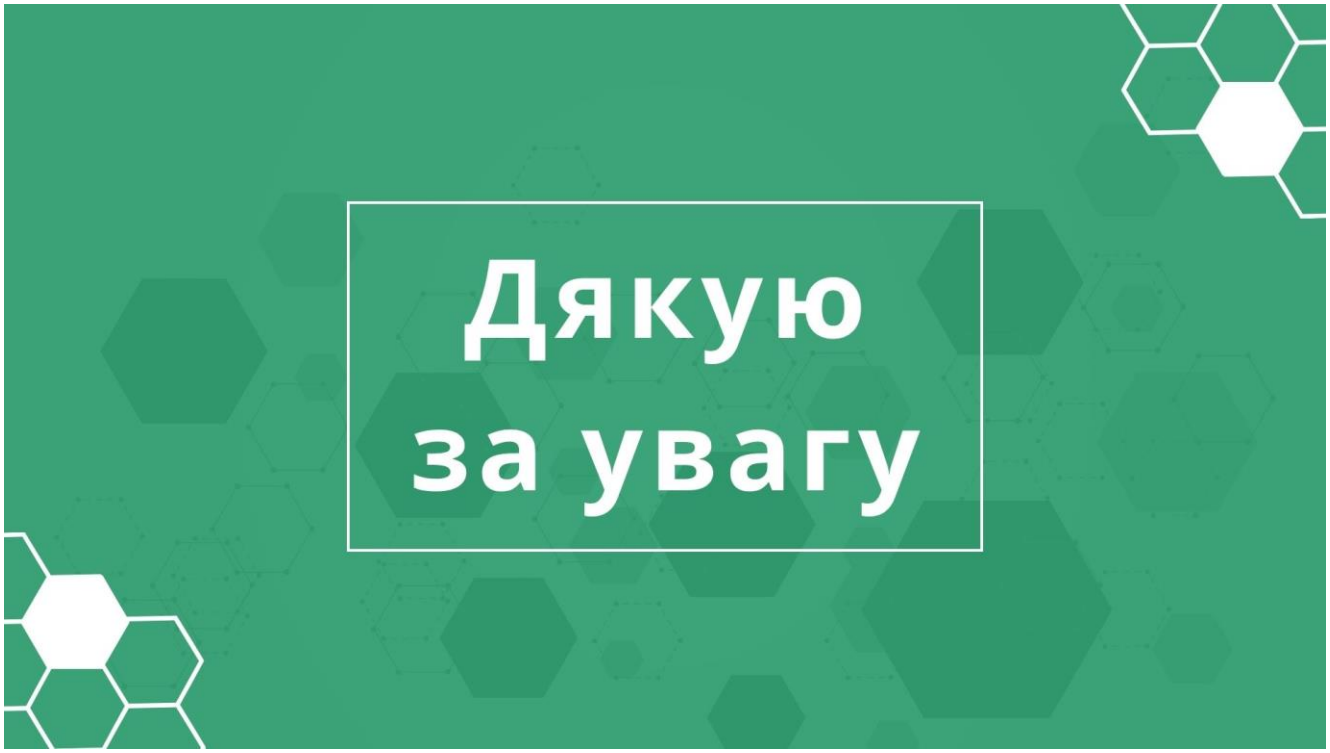
Результати функціонального тестування підтвердили **готовність системи для впровадження в IT-інфраструктуру**, забезпечуючи своєчасне виявлення проблем і оповіщення.

АПРОБАЦІЯ

VII міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT»

Тези

«Інтелектуальні інформаційні системи для моніторингу, управління та підтримки користувачів»


Дякую
за увагу