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

# КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інформаційно-аналітична система моніторингу хмарної інфраструктури»

на здобуття освітнього ступеня бакалавра
зі спеціальності *124 Системний аналіз*
*(код, найменування спеціальності)*
освітньо-професійної програми *Системний аналіз*
*(назва)*

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

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

Виконав: здобувач вищої освіти гр. САД-41
Олексій ГОРБУНОВ
Ім'я, ПРІЗВИЩЕ

Керівник: Михайло КУЗЬМІЧ, доктор філософії, доцент кафедри
Ім'я, ПРІЗВИЩЕ

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

**Київ 2026**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

**Навчально-науковий інститут інформаційних технологій**

Кафедра Інформаційних систем та технологій

Ступінь вищої освіти Бакалавр

Спеціальність 124 Системний аналіз

Освітньо-професійна програма Системний аналіз

**ЗАТВЕРДЖУЮ**
Завідувач кафедрою ІПЗАС
_____Каміла СТОРЧАК
«____» _____2026 р.

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ РОБОТУ

**Горбунов Олексій Євгенович**
*(прізвище, ім'я, по батькові здобувача)*

1. Тема кваліфікаційної роботи: «Інформаційно-аналітична система моніторингу хмарної інфраструктури»

керівник кваліфікаційної роботи Михайло КУЗЬМІЧ, доктор філософії, доцент кафедри
*(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)*

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «___» _____2026р. № 36

2. Строк подання кваліфікаційної роботи «___» _____20___р.
3. Вихідні дані до кваліфікаційної роботи: інформаційно-аналітична система моніторингу хмарної інфраструктури;
наукова та технічна література з хмарних технологій і систем моніторингу.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. визначення потреб у моніторингу хмарної інфраструктури; формування функціональних вимог до інформаційно-аналітичної системи; обґрунтування необхідності збору;
2. вибір та обґрунтування архітектури інформаційно-аналітичної системи моніторингу; дослідження принципів збору, зберігання та обробки даних;
3. розробка механізмів збору та обробки метрик хмарної інфраструктури; інтеграція інструментів моніторингу.
5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «___» _____2026р.

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Визначення потреб та вимог до системи моніторингу хмарної інфраструктури. Формулювання цілей та обґрунтування вибору технологій. | 15.02.2026 – 28.02.2026. | |
| 2 | Аналіз наукової та технічної літератури. Дослідження сучасних підходів до моніторингу хмарної інфраструктури та інформаційно-аналітичних систем. | 01.03.2026 – 14.03.2026 | |
| 3 | Огляд, порівняння та аналіз технологій моніторингу: Prometheus, Grafana, CloudWatch, системи збору метрик, логування та алертингу. | 15.03.2026 – 28.03.2026 | |
| 4 | Розробка рекомендацій щодо побудови ефективної системи моніторингу хмарної інфраструктури, забезпечення надійності, масштабованості та своєчасного виявлення аномалій. | 29.03.2026 – 11.04.2026 | |
| 5 | Реалізація програмної частини: розробка та інтеграція компонентів збору, зберігання та обробки метрик; налаштування системи візуалізації та алертів. | 12.04.2026 – 25.04.2026 | |
| 6 | Оформлення результатів дослідження. Написання та оформлення кваліфікаційної роботи. | 26.04.2026 – 12.05.2026 | |
| 7 | Підготовка доповіді до захисту. | 13.05.2026 – 21.05.2026 | |

Здобувач вищої освіти _____ Горбунов Олексій
*(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

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

# РЕФЕРАТ

Текстова частина бакалаврської роботи: 71 сторінки, 26 рисунків, 4 таблиці 32 джерела.

*Об'єкт дослідження* – процеси моніторингу та аналізу стану хмарної інфраструктури в інформаційних системах.

*Предмет дослідження* – методи та засоби побудови інформаційно-аналітичних систем для збору, обробки та візуалізації даних про стан хмарних ресурсів.

*Мета роботи* – розробка інформаційно-аналітичної системи моніторингу хмарної інфраструктури, яка забезпечує збір метрик, аналіз стану системи, виявлення аномалій та зручну візуалізацію даних для користувача.

*Короткий зміст роботи*. У кваліфікаційній роботі розглянуто теоретичні та практичні аспекти створення інформаційно-аналітичної системи моніторингу хмарної інфраструктури. Проаналізовано сучасні підходи до організації моніторингу, зокрема використання метрик, логів та механізмів оповіщення. Досліджено можливості сучасних інструментів, таких як Prometheus, Grafana та CloudWatch. У роботі здійснено проєктування архітектури системи, визначено її основні компоненти та принципи взаємодії між ними.

Практична частина присвячена реалізації системи моніторингу, налаштуванню інструментів збору метрик і аналізу стану інфраструктури, а також тестуванню її ефективності. Отримані результати підтверджують доцільність використання запропонованого підходу для підвищення надійності та керованості хмарних систем.

КЛЮЧОВІ СЛОВА: ХМАРНА ІНФРАСТРУКТУРА, МОНІТОРИНГ, МЕТРИКИ, PROMETHEUS, GRAFANA, CLOUDWATCH, АНАЛІЗ ДАНИХ, АНОМАЛІЇ, ДАШБОРДИ, ІНФОРМАЦІЙНО-АНАЛІТИЧНА СИСТЕМА

**ЗМІСТ**

# ВСТУП

*Актуальність теми.* У сучасних умовах стрімкого розвитку інформаційних технологій та широкого впровадження хмарних обчислень питання ефективного моніторингу хмарної інфраструктури набуває особливої актуальності. Організації все частіше використовують розподілені системи, що включають велику кількість серверів, сервісів і мережевих компонентів, які потребують постійного контролю та аналізу їх стану.

Недостатній рівень моніторингу або відсутність централізованих засобів збору та обробки даних може призводити до зниження продуктивності систем, виникнення збоїв, втрати даних та фінансових збитків. Крім того, складність сучасних хмарних середовищ ускладнює своєчасне виявлення аномалій і реагування на інциденти без використання спеціалізованих інформаційно-аналітичних систем.

У зв’язку з цим виникає потреба у розробці ефективних рішень, які забезпечують автоматизований збір метрик, аналіз стану інфраструктури, виявлення відхилень від нормальної роботи та зручну візуалізацію результатів. Саме тому дослідження, присвячене створенню інформаційно-аналітичної системи моніторингу хмарної інфраструктури, є своєчасним і має важливе практичне значення.

*Мета роботи* – розробка інформаційно-аналітичної системи моніторингу хмарної інфраструктури, яка забезпечує збір метрик, аналіз стану системи, виявлення аномалій та зручну візуалізацію даних для користувача.

Для досягнення мети, у бакалаврській роботі успішно виконано наступні завдання:

– дослідження сучасних тенденцій розвитку та використання хмарних технологій і інфраструктур;

– огляд методів моніторингу хмарної інфраструктури, зокрема аналізу метрик, логів та систем оповіщення;

– аналіз результатів впровадження інформаційно-аналітичних систем моніторингу, оцінка ефективності виявлення аномалій та забезпечення стабільності роботи хмарних сервісів.

*Об'єкт дослідження* – процеси моніторингу та аналізу стану хмарної інфраструктури в інформаційних системах.

*Предмет дослідження* – методи та засоби побудови інформаційно-аналітичних систем для збору, обробки та візуалізації даних про стан хмарних ресурсів.

*Методи дослідження*:

1. Аналіз сучасних підходів до організації хмарних інфраструктур та їх моніторингу.
2. Дослідження існуючих систем моніторингу (Prometheus, Grafana, CloudWatch тощо).
3. Проєктування архітектури інформаційно-аналітичної системи.
4. Розробка механізмів збору та обробки метрик.
5. Реалізація алгоритмів аналізу даних та виявлення аномалій.
6. Створення інтерфейсу користувача для візуалізації результатів.

*Наукова новизна одержаних результатів.* У ході дослідження запропоновано підхід до побудови інформаційно-аналітичної системи моніторингу хмарної інфраструктури, який базується на комплексному використанні засобів збору метрик, аналізу логів та інтелектуального виявлення аномалій. Запропонована модель передбачає інтеграцію сучасних інструментів моніторингу (Prometheus, Grafana, CloudWatch) з аналітичними механізмами обробки даних, що дозволяє підвищити точність і оперативність виявлення відхилень у роботі хмарних сервісів.

*Практична значущість одержаних результатів.* Розроблена інформаційно-аналітична система забезпечує ефективний моніторинг стану хмарної інфраструктури, своєчасне виявлення аномалій та збоїв, а також зручну візуалізацію даних для користувача. Отримані результати можуть бути використані для підвищення надійності, продуктивності та керованості хмарних сервісів у реальних умовах експлуатації.

*Апробація результатів та публікації.* Основні положення і результати бакалаврської роботи доповідались на науково практичних конференціях, що проходили на базі Державного університету інформаційно-комунікаційних технологій. Горбунов О. Є. Аналіз засобів моніторингу та управління хмарною інфраструктурою в IoT-системах // Сучасний стан та перспективи розвитку IoT: матеріали VII Міжнародної науково-технічної конференції. – Київ : ДУІКТ, 2026.

# 1 ТЕОРЕТИЧНІ ОСНОВИ МОНІТОРИНГУ ХМАРНОЇ ІНФРАСТРУКТУРИ

## 1.1 Поняття хмарної інфраструктури та її складові

Хмарна інфраструктура є однією з технологічних основ сучасних інформаційних систем та цифрових сервісів. Її активне впровадження зумовлене стрімким розвитком хмарних обчислень, необхідністю забезпечення масштабованості, високої доступності та ефективного використання обчислювальних ресурсів. Сьогодні хмарні технології використовуються у більшості сфер діяльності, включаючи бізнес, фінансовий сектор, державне управління, електронну комерцію, медичні та освітні системи.

Під хмарною інфраструктурою розуміють сукупність апаратних і програмних компонентів, які забезпечують надання обчислювальних ресурсів через мережу Інтернет у вигляді сервісів. До таких ресурсів належать сервери, системи зберігання даних, мережеве обладнання, віртуальні машини, контейнери, бази даних та програмні платформи. Основною особливістю хмарної інфраструктури є можливість гнучкого управління ресурсами та їх динамічного масштабування відповідно до потреб користувачів.

Хмарна інфраструктура базується на технологіях віртуалізації, які дозволяють ефективно використовувати фізичні ресурси серверів шляхом створення декількох ізольованих віртуальних середовищ. Завдяки цьому забезпечується підвищення продуктивності, оптимізація використання обладнання та зниження витрат на підтримку інформаційних систем. Крім того, використання хмарних платформ дозволяє організаціям уникнути необхідності створення власних дата-центрів та спрощує процес адміністрування інфраструктури.

У разі збільшення навантаження система може автоматично виділяти додаткові ресурси, що забезпечує стабільність роботи сервісів навіть при значній кількості користувачів або високому рівні обробки даних. Також хмарні технології забезпечують високий рівень доступності сервісів, резервування даних та можливість швидкого відновлення після збоїв.

Складові хмарної інфраструктури, до них належать обчислювальні ресурси, системи зберігання даних, мережеві сервіси, засоби віртуалізації та системи моніторингу. Обчислювальні ресурси забезпечують виконання програмних застосунків та сервісів, тоді як системи зберігання відповідають за обробку й збереження інформації. Мережеві компоненти забезпечують передачу даних між серверами, користувачами та окремими сервісами.

Хмарні технології базуються на декількох основних моделях надання послуг, які визначають рівень доступу користувача до інфраструктури, програмного забезпечення та інструментів керування системою. Найпоширенішими моделями хмарних обчислень є IaaS, PaaS та SaaS. Кожна з них має власні особливості, переваги та сфери застосування (рис. 1.1).

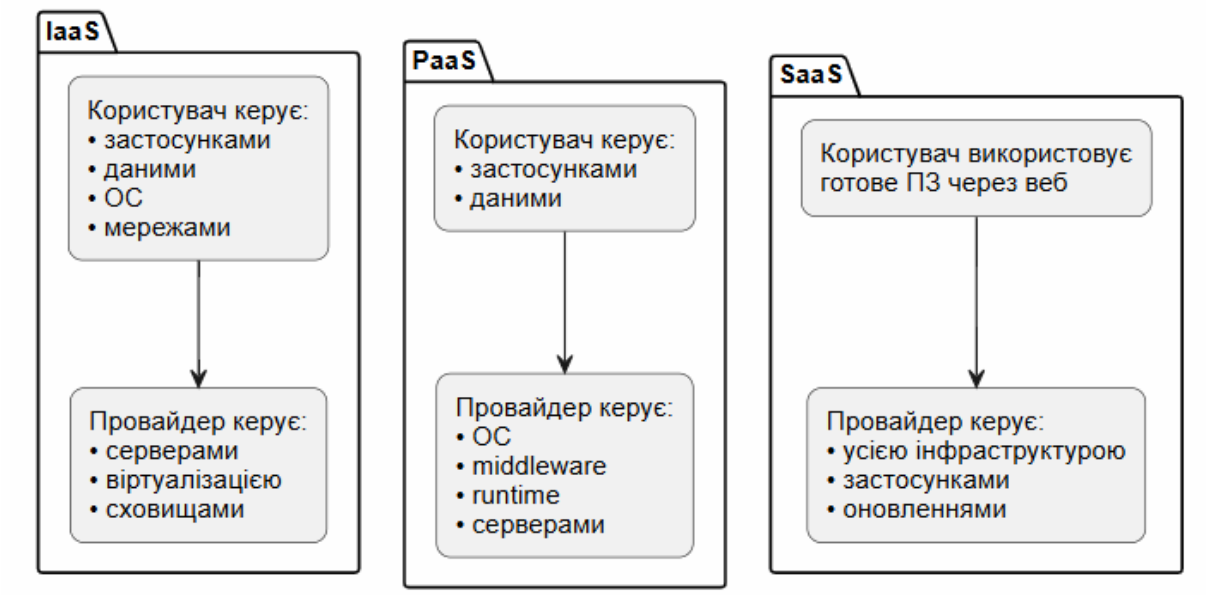


Рисунок 1.1 – Основні моделі хмарних обчислень та розподіл відповідальності між користувачем і провайдером

IaaS (Infrastructure as a Service — інфраструктура як сервіс) є моделлю хмарних обчислень, у межах якої користувачу надаються базові обчислювальні ресурси у вигляді віртуалізованої інфраструктури. До таких ресурсів належать сервери, системи зберігання даних, мережеве обладнання, віртуальні машини та засоби балансування навантаження.

Основною особливістю IaaS є те, що провайдер хмарних послуг забезпечує фізичну інфраструктуру та засоби віртуалізації, тоді як користувач самостійно налаштовує операційні системи, програмне забезпечення, мережеві параметри та служби. Такий підхід надає високий рівень гнучкості та дозволяє повністю контролювати конфігурацію середовища.

Модель IaaS широко використовується для створення корпоративних серверних середовищ, веб-сервісів, систем резервного копіювання, тестових платформ та великих інформаційних систем. Вона дозволяє швидко масштабувати ресурси залежно від навантаження та уникати значних витрат на придбання фізичного обладнання.

До переваг IaaS належать:

- гнучке масштабування ресурсів;
- високий рівень контролю над системою;
- зменшення витрат на фізичну інфраструктуру;
- можливість швидкого розгортання серверів;
- підтримка резервування та відновлення даних.

Водночас модель IaaS потребує високого рівня адміністрування, оскільки користувач самостійно відповідає за налаштування безпеки, оновлення програмного забезпечення та підтримку працездатності сервісів.

Прикладами IaaS-платформ є Amazon EC2, Microsoft Azure Virtual Machines та Google Compute Engine.

PaaS (Platform as a Service — платформа як сервіс) є моделлю хмарних обчислень, у межах якої користувач отримує готове середовище для розробки, тестування та розгортання програмних застосунків. У цьому випадку провайдер не лише надає інфраструктуру, а й забезпечує операційні системи, сервери застосунків, бази даних, середовища виконання та інструменти розробки.

Головною перевагою PaaS є спрощення процесу створення програмного забезпечення. Розробникам не потрібно займатися адмініструванням серверів або налаштуванням інфраструктури, оскільки ці функції виконує хмарний провайдер. Це дозволяє зосередитися безпосередньо на розробці функціональності застосунку.

PaaS-платформи активно використовуються для створення веб-застосунків, API-сервісів, мобільних платформ та корпоративних систем. Вони підтримують автоматичне масштабування, інтеграцію з базами даних, системами безпеки та засобами CI/CD.

До переваг PaaS належать:

- спрощення процесу розробки;
- швидке розгортання застосунків;
- автоматичне оновлення середовища;
- інтеграція інструментів розробки;
- підтримка командної роботи.

Недоліком PaaS є менший рівень контролю над інфраструктурою порівняно з IaaS, а також залежність від технологічних рішень конкретного провайдера.

Прикладами PaaS є Heroku, Google App Engine, Railway та Microsoft Azure App Services.

SaaS (Software as a Service — програмне забезпечення як сервіс) є моделлю, у якій користувач отримує готовий програмний продукт через мережу Інтернет без необхідності встановлення або адміністрування програмного забезпечення на власному обладнанні.

У моделі SaaS усі компоненти системи, включаючи інфраструктуру, сервери, бази даних, програмне забезпечення та оновлення, повністю підтримуються провайдером. Користувач взаємодіє із сервісом через веб-інтерфейс або клієнтський застосунок.

SaaS-рішення широко застосовуються у сфері електронної пошти, офісних платформ, CRM-систем, систем управління проєктами, відеоконференцій та хмарних сховищ. Завдяки простоті використання ця модель є найбільш поширеною серед кінцевих користувачів.

До основних переваг SaaS належать:

- відсутність необхідності встановлення програм;
- автоматичні оновлення;
- доступ до сервісів із будь-якого пристрою;

– зменшення витрат на технічне обслуговування;

– швидкий початок роботи.

Основним недоліком SaaS є обмежений рівень контролю над системою та залежність від постачальника послуг і стабільності мережі Інтернет.

Прикладами SaaS-сервісів є Google Workspace, Microsoft 365, Dropbox, Zoom та Salesforce.

Залежно від способу організації та доступу до ресурсів хмарні інфраструктури поділяються на декілька основних типів: public cloud, private cloud та hybrid cloud. Кожен із цих типів має власні особливості, переваги, рівень безпеки та сфери застосування. Вибір конкретного типу хмари залежить від вимог організації до масштабованості, контролю над даними, продуктивності та захисту інформації (рис. 1.2).

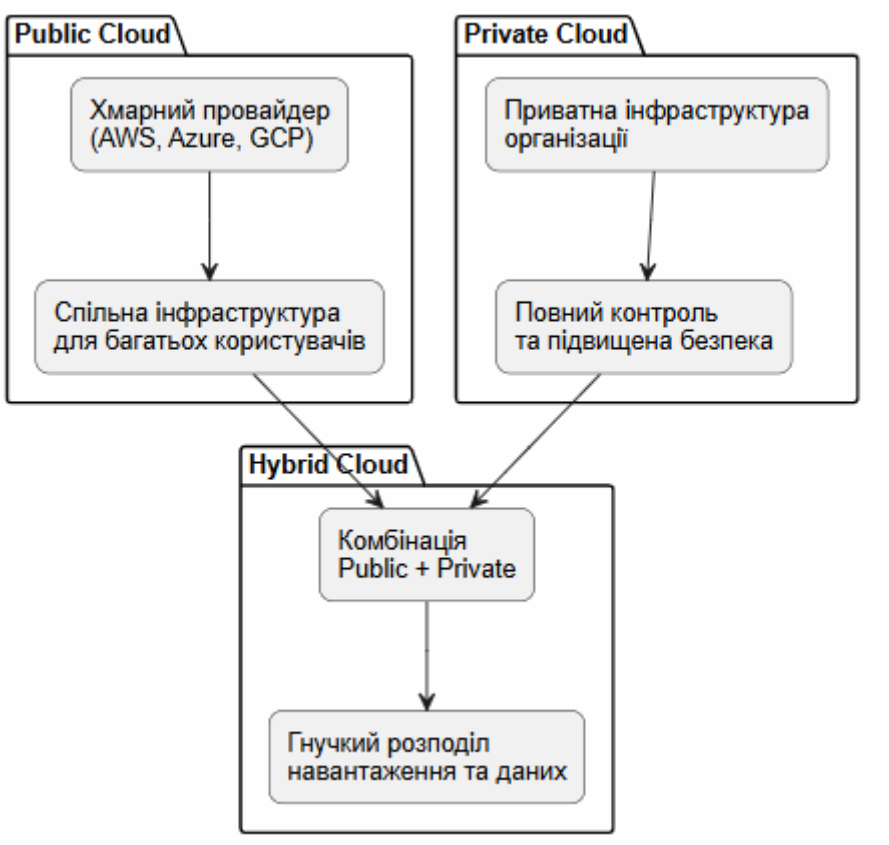


Рисунок 1.2 – Основні типи хмарних інфраструктур та їх характеристики

Public cloud (публічна хмара) є моделлю хмарної інфраструктури, у якій обчислювальні ресурси надаються користувачам через мережу Інтернет стороннім провайдером хмарних послуг. У цьому випадку фізична інфраструктура, сервери,

мережеве обладнання та системи зберігання даних належать постачальнику послуг, а користувач отримує доступ до ресурсів у вигляді сервісу.

Публічні хмари характеризуються високим рівнем масштабованості та доступності. Користувач може швидко збільшувати або зменшувати кількість ресурсів залежно від навантаження системи, що особливо важливо для веб-застосунків, онлайн-сервісів та платформ із великою кількістю користувачів.

Однією з головних переваг public cloud є відсутність необхідності придбання власного серверного обладнання та підтримки дата-центрів. Це дозволяє суттєво знизити витрати на розгортання інформаційних систем і спростити адміністрування інфраструктури.

До переваг public cloud належать:

– швидке розгортання сервісів;
– автоматичне масштабування;
– висока доступність;
– зниження витрат на обладнання;
– підтримка резервування даних.

Водночас публічна хмара має певні недоліки. Основним із них є обмежений контроль над фізичною інфраструктурою та підвищені вимоги до забезпечення конфіденційності даних. У деяких випадках організації не можуть зберігати критично важливу інформацію у публічних середовищах через вимоги безпеки або законодавчі обмеження.

Прикладами public cloud є Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform.

Private cloud (приватна хмара) є моделлю хмарної інфраструктури, у якій ресурси використовуються лише однією організацією. Така інфраструктура може розміщуватися у власному дата-центрі компанії або на виділених серверах хмарного провайдера, але доступ до ресурсів мають лише авторизовані користувачі організації.

Головною особливістю private cloud є високий рівень контролю над інфраструктурою та даними. Організація самостійно визначає політики безпеки,

способи зберігання інформації, правила доступу та параметри функціонування системи.

Приватні хмари часто використовуються у банківському секторі, державних установах, медичних інформаційних системах та великих корпоративних мережах, де особливе значення має захист конфіденційної інформації.

До переваг private cloud належать:

– високий рівень безпеки;
– повний контроль над ресурсами;
– можливість індивідуального налаштування;
– підвищена продуктивність;
– ізоляція інфраструктури від сторонніх користувачів.

Основними недоліками private cloud є висока вартість розгортання та підтримки, а також необхідність постійного адміністрування серверної інфраструктури.

Для створення приватних хмар часто використовуються OpenStack, VMware vSphere та Microsoft Private Cloud.

Hybrid cloud (гібридна хмара) є моделлю, яка поєднує можливості public cloud та private cloud в єдиній інфраструктурі. У такому середовищі частина ресурсів розміщується у приватній хмарі, а інша — у публічній.

Гібридна модель дозволяє організаціям поєднати переваги обох типів хмар. Критично важливі дані та внутрішні сервіси можуть зберігатися у приватній хмарі, тоді як менш чутливі сервіси або ресурсоємні задачі можуть виконуватися у public cloud.

Однією з головних переваг hybrid cloud є гнучкість. Організація може динамічно переносити навантаження між приватною та публічною інфраструктурою залежно від поточних потреб системи. Це особливо важливо для масштабованих веб-сервісів та корпоративних платформ.

До переваг hybrid cloud належать:

– поєднання безпеки та масштабованості;
– оптимізація витрат;

- гнучке управління ресурсами;
- можливість резервування сервісів;
- підвищення відмовостійкості системи.

Недоліком hybrid cloud є складність адміністрування та інтеграції різних середовищ, а також необхідність забезпечення безпечної взаємодії між приватною та публічною частинами інфраструктури.

Гібридні хмари активно використовуються великими компаніями, які потребують високого рівня безпеки та одночасно використовують масштабовані публічні сервіси.

Хмарна інфраструктура складається з великої кількості взаємопов'язаних компонентів, які забезпечують функціонування інформаційних систем, обробку даних, зберігання інформації та надання сервісів користувачам. Кожен із компонентів виконує окрему роль у загальній архітектурі хмарного середовища та забезпечує стабільність, продуктивність і масштабованість системи.

Серверне обладнання забезпечує виконання обчислювальних процесів, запуск програмних застосунків та обробку запитів користувачів. У хмарних середовищах сервери об'єднуються у великі дата-центри, що дозволяє забезпечити високу продуктивність і відмовостійкість системи. Сучасні хмарні платформи використовують фізичні та віртуальні сервери, які можуть автоматично масштабуватися залежно від навантаження.

Мережеві технології дозволяють організовувати взаємодію між компонентами системи, забезпечувати балансування навантаження та підтримувати безпечний доступ до ресурсів. Для цього використовуються маршрутизатори, комутатори, VPN-технології, міжмережеві екрани та системи балансування навантаження.

Віртуальні машини дозволяють запускати декілька ізольованих операційних систем на одному фізичному сервері. Використання віртуалізації забезпечує ефективніше використання апаратних ресурсів, спрощує адміністрування та дозволяє швидко створювати нові середовища для роботи сервісів. Віртуальні

машини активно використовуються у корпоративних системах, тестових середовищах та хмарних сервісах.

Контейнеризація дозволяє ізолювати програмні застосунки разом із необхідними бібліотеками та залежностями, що значно спрощує процес розгортання програмного забезпечення. Контейнери споживають менше ресурсів порівняно з віртуальними машинами та забезпечують швидший запуск сервісів. Найбільш поширеними платформами контейнеризації є Docker та Kubernetes.

Невід'ємною складовою хмарної інфраструктури є бази даних, які забезпечують зберігання, обробку та управління інформацією. У хмарних середовищах можуть використовуватися як реляційні бази даних (PostgreSQL, MySQL), так і NoSQL-рішення (MongoDB, Cassandra). Бази даних забезпечують централізоване зберігання інформації, підтримують резервування даних та забезпечують швидкий доступ до інформаційних ресурсів.

Для довготривалого зберігання інформації використовуються сховища даних. Хмарні сховища дозволяють зберігати великі обсяги інформації, забезпечують резервне копіювання та підтримують механізми відновлення після збоїв. Вони можуть використовуватися для зберігання файлів, резервних копій, журналів подій та мультимедійного контенту.

Хмарна інфраструктура має значну кількість переваг, що обумовлює її широке використання у сучасних інформаційних системах. Однією з головних переваг є масштабованість, яка дозволяє швидко змінювати кількість ресурсів залежно від навантаження системи. Це забезпечує стабільність роботи сервісів навіть при значному збільшенні кількості користувачів.

Використання хмарних сервісів дозволяє організаціям уникнути значних витрат на придбання та підтримку власного серверного обладнання. Оплата здійснюється лише за фактично використані ресурси, що робить використання хмарних платформ більш гнучким.

До переваг також належать:

– висока доступність сервісів;
– автоматичне резервування даних;

– швидке розгортання інфраструктури;
– централізоване адміністрування;
– підтримка віддаленого доступу.

Водночас хмарна інфраструктура має і певні недоліки. Одним із ризиків є залежність від стабільності мережі Інтернет та роботи хмарного провайдера. У разі збою на стороні постачальника послуг доступ до сервісів може бути тимчасово втрачений.

Іншим важливим недоліком є питання безпеки та конфіденційності даних. Оскільки інформація зберігається на сторонніх серверах, виникає потреба у використанні додаткових механізмів шифрування, автентифікації та контролю доступу.

Також до недоліків належать:

– складність контролю фізичної інфраструктури;
– ризик витоку даних;
– залежність від постачальника послуг;
– складність міграції між платформами.

З основних проблем сучасних хмарних середовищ є складність адміністрування та контролю великої кількості компонентів системи. У масштабних інфраструктурах одночасно можуть функціонувати сотні серверів, контейнерів, баз даних і мережевих сервісів, що значно ускладнює процес ручного контролю за їх станом.

У разі перевантаження серверів, нестачі пам’яті або проблем із мережею можуть виникати затримки у роботі сервісів або повне припинення їх функціонування. Саме тому сучасні хмарні системи потребують впровадження автоматизованих засобів моніторингу.

Метрики, журнали подій та системні повідомлення надходять із великої кількості джерел, тому виникає потреба у створенні єдиної інформаційно-аналітичної системи для їх обробки та візуалізації. Окрім цього, адміністрування хмарної інфраструктури ускладнюється необхідністю забезпечення:

– безпеки даних;

- резервування інформації;
- балансування навантаження;
- автоматичного масштабування;
- контролю доступу користувачів;
- постійного моніторингу продуктивності системи.

Сучасні хмарні інфраструктури є складними багаторівневими системами, які потребують використання ефективних засобів адміністрування, моніторингу та аналітики для забезпечення стабільності й безпеки їх функціонування.

## 1.2 Основні підходи до моніторингу (метрики, логи, алерти)

У інформаційних системах моніторинг є елементом забезпечення стабільної та безпечної роботи хмарної інфраструктури. Зі збільшенням кількості серверів, сервісів, контейнерів і мережевих компонентів значно ускладнюється процес контролю за станом системи, що обумовлює необхідність використання спеціалізованих засобів моніторингу.

Під моніторингом розуміють процес постійного спостереження, збору, аналізу та обробки інформації про стан інформаційної системи або її окремих компонентів. Основною метою моніторингу є отримання актуальних даних про роботу інфраструктури, виявлення збоїв, контроль продуктивності та забезпечення своєчасного реагування на потенційні проблеми.

Ефективний моніторинг хмарної інфраструктури базується на зборі та аналізі різних типів даних, які дозволяють отримувати повну інформацію про стан системи, продуктивність сервісів та можливі збої. У сучасних системах моніторингу основними типами даних є метрики, логи, трейси та події. Кожен із цих типів виконує окрему функцію в процесі аналізу інфраструктури.

Метрики є числовими показниками, які відображають поточний стан системи або окремих її компонентів у певний момент часу. Вони використовуються для оцінки продуктивності серверів, сервісів, мережевих ресурсів та програмного забезпечення.

Метрики збираються через визначені інтервали часу та зберігаються у вигляді часових рядів. Це дозволяє аналізувати зміни навантаження, прогнозувати можливі перевантаження та виявляти відхилення у роботі системи.

До найбільш поширених метрик належать:

– завантаження процесора;
– використання оперативної пам’яті;
– обсяг дискового простору;
– мережева активність;
– кількість запитів;
– швидкість відповіді сервісів.

Основною перевагою метрик є можливість швидкого аналізу продуктивності системи та побудови графіків для візуалізації стану інфраструктури.

Логи (журнали подій) являють собою текстові записи про події, які відбуваються в системі під час її роботи. Логування використовується для фіксації помилок, дій користувачів, змін конфігурацій та системних повідомлень.

На відміну від метрик, логи містять детальнішу інформацію про конкретні події та дозволяють проводити глибокий аналіз інцидентів. Логи можуть генеруватися:

операційними системами;
– веб-серверами;
– базами даних;
– контейнерами;
– мережевими сервісами;
– програмними застосунками.

Аналіз логів дозволяє:

– виявляти помилки;
– знаходити причини збоїв;
– аналізувати активність користувачів;
– контролювати безпеку системи;

– виявляти спроби несанкціонованого доступу.

У великих хмарних інфраструктурах використовується централізоване логування, що дозволяє збирати інформацію з різних джерел в єдине сховище.

Трейси використовуються для відстеження шляху виконання запиту через різні компоненти системи. Особливе значення трейси мають у мікросервісних архітектурах, де один запит може проходити через десятки сервісів.

Трейс містить інформацію про:

– послідовність викликів;

– час виконання операцій;

– взаємодію між сервісами;

– затримки у роботі компонентів.

Використання трейсів дозволяє:

– виявляти вузькі місця системи;

– аналізувати затримки;

– оптимізувати роботу мікросервісів;

– спрощувати пошук причин помилок.

Для реалізації distributed tracing часто використовуються Jaeger, Zipkin та OpenTelemetry.

Події є повідомленнями про певні дії або зміни стану системи. Події можуть створюватися автоматично у разі виникнення помилок, запуску сервісів, зміни конфігурації або перевищення встановлених порогових значень.

Події використовуються для:

– запуску автоматичних сценаріїв;

– створення сповіщень;

– аудиту дій користувачів;

– контролю безпеки;

– реєстрації змін у системі.

Події можуть бути як інформаційними, так і критичними, що вимагають негайного реагування адміністратора.

У процесі моніторингу хмарної інфраструктури особлива увага приділяється контролю ключових показників продуктивності системи.

CPU (Central Processing Unit) характеризує рівень навантаження на процесор. Цей показник дозволяє визначити, наскільки активно використовуються обчислювальні ресурси сервера.

Високе навантаження на CPU може свідчити про:

- перевантаження системи;
- недостатню кількість ресурсів;
- помилки у програмному забезпеченні;
- DDoS-атаки;
- неефективні алгоритми.

Для кращого розуміння процесу моніторингу хмарної інфраструктури доцільно представити діаграму взаємодії основних компонентів системи моніторингу, яка демонструє процес збору метрик, аналізу даних, формування алертів та візуалізації інформації (рис. 1.3)

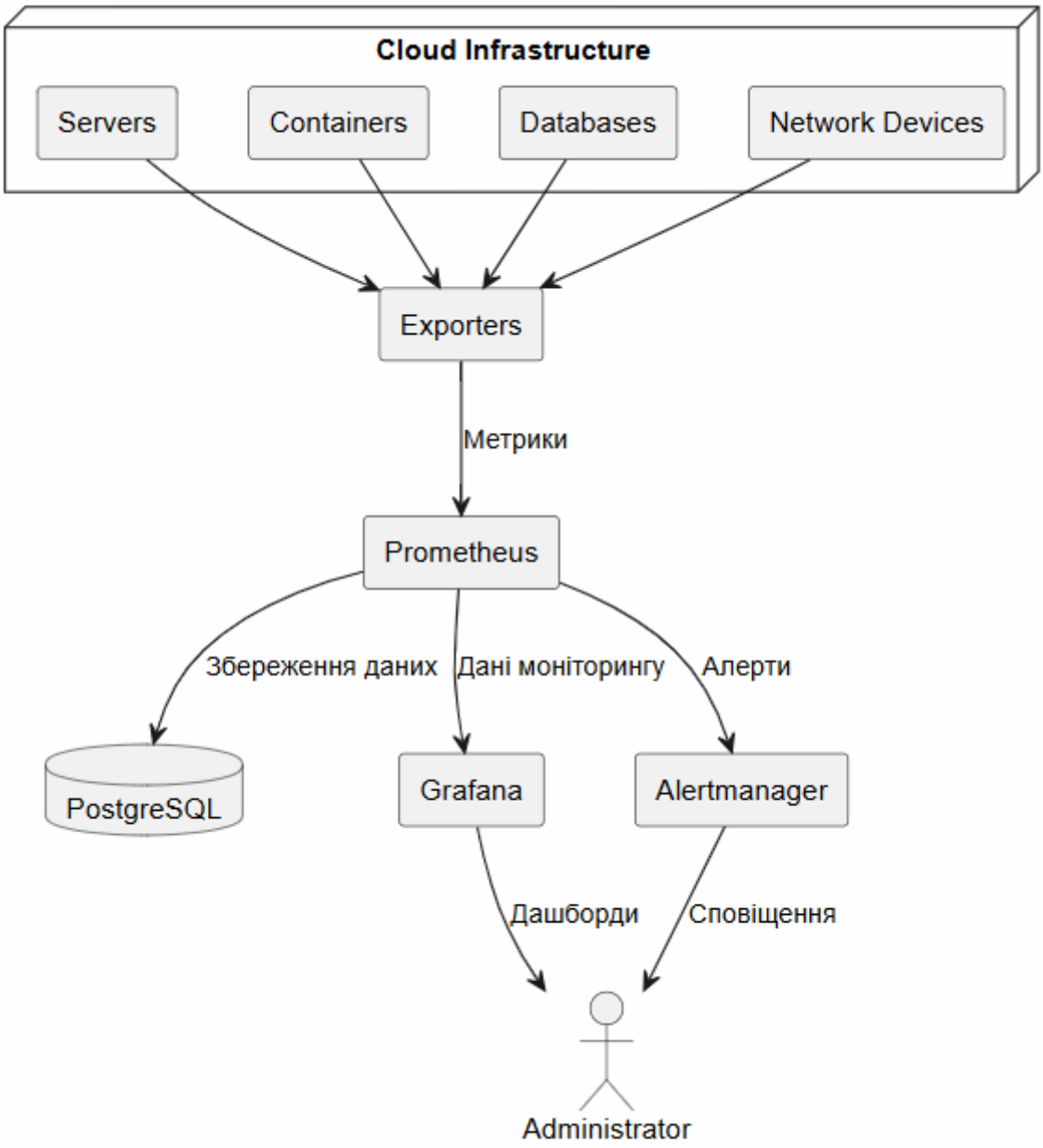


Рисунок 1.3 – UML-діаграма архітектури системи моніторингу хмарної інфраструктури

У процесі моніторингу хмарної інфраструктури особлива увага приділяється контролю ключових показників продуктивності системи.

Моніторинг CPU дозволяє своєчасно масштабувати інфраструктуру та уникати деградації продуктивності.

RAM характеризує використання оперативної пам’яті системи. Нестача пам’яті може призводити до значного зниження швидкодії сервісів або аварійного завершення процесів.

Контроль RAM дозволяє:

– виявляти memory leaks;
– контролювати навантаження застосунків;
– оптимізувати використання ресурсів.

Показники Disk відображають стан дискової підсистеми, включаючи:

– використання дискового простору;
– швидкість читання/запису;
– I/O операції;
– стан файлової системи.

Моніторинг дисків дозволяє уникати переповнення сховищ та втрати даних.

Network характеризує мережеву активність системи. До основних показників належать:

– швидкість передачі даних;
– кількість пакетів;
– затримки;
– втрати пакетів;
– пропускна здатність каналу.

Мережевий моніторинг є критично важливим для хмарних сервісів, оскільки більшість компонентів взаємодіють через мережу.

Uptime показує тривалість безперервної роботи системи або сервісу. Високий uptime є одним із головних показників надійності інфраструктури.

Моніторинг uptime дозволяє:

– контролювати доступність сервісів;
– аналізувати стабільність роботи;
– оцінювати якість інфраструктури.

Response time характеризує швидкість відповіді сервісу на запит користувача. Цей показник є одним із ключових критеріїв оцінки продуктивності веб-застосунків та API.

Високий response time може бути наслідком:

– перевантаження серверів;
– проблем із мережею;
– повільної роботи бази даних;
– помилок у застосунках.

Системи алертингу використовуються для автоматичного сповіщення адміністраторів про проблеми у роботі інфраструктури. Вони аналізують метрики та події, після чого генерують повідомлення у разі перевищення критичних порогів.

Сучасні системи алертингу підтримують:

Email-сповіщення;

Telegram;

Slack;

SMS;

Webhook-повідомлення.

Однією з найпоширеніших систем алертингу є Alertmanager, який інтегрується з Prometheus.

Виявлення аномалій – це одна з дій моніторингу хмарної інфраструктури. Основною метою є автоматичне визначення відхилень від нормальної поведінки системи.

Для виявлення аномалій використовуються:

– порогові значення;
– статистичний аналіз;
– машинне навчання;
– поведінкові моделі;
– прогнозування часових рядів.

Аномаліями можуть бути:

– різке зростання навантаження;
– підозріла мережева активність;
– збільшення кількості помилок;
– нестандартна поведінка сервісів.

У сучасних хмарних середовищах використовується централізований моніторинг, який передбачає збір інформації з усіх компонентів інфраструктури в єдину систему.

Централізований підхід дозволяє:

– отримувати повну картину стану системи;
– спрощувати адміністрування;
– проводити комплексний аналіз;
– централізовано керувати алертами;
– візуалізувати дані у вигляді дашбордів.

Для реалізації централізованого моніторингу часто використовуються Prometheus, Grafana, ELK Stack та Zabbix.

Для відображення послідовності обробки інформації в системі моніторингу доцільно представити UML-діаграму потоку даних, яка демонструє етапи збору, аналізу, виявлення аномалій та формування сповіщень у хмарному середовищі (рис. 1.4).

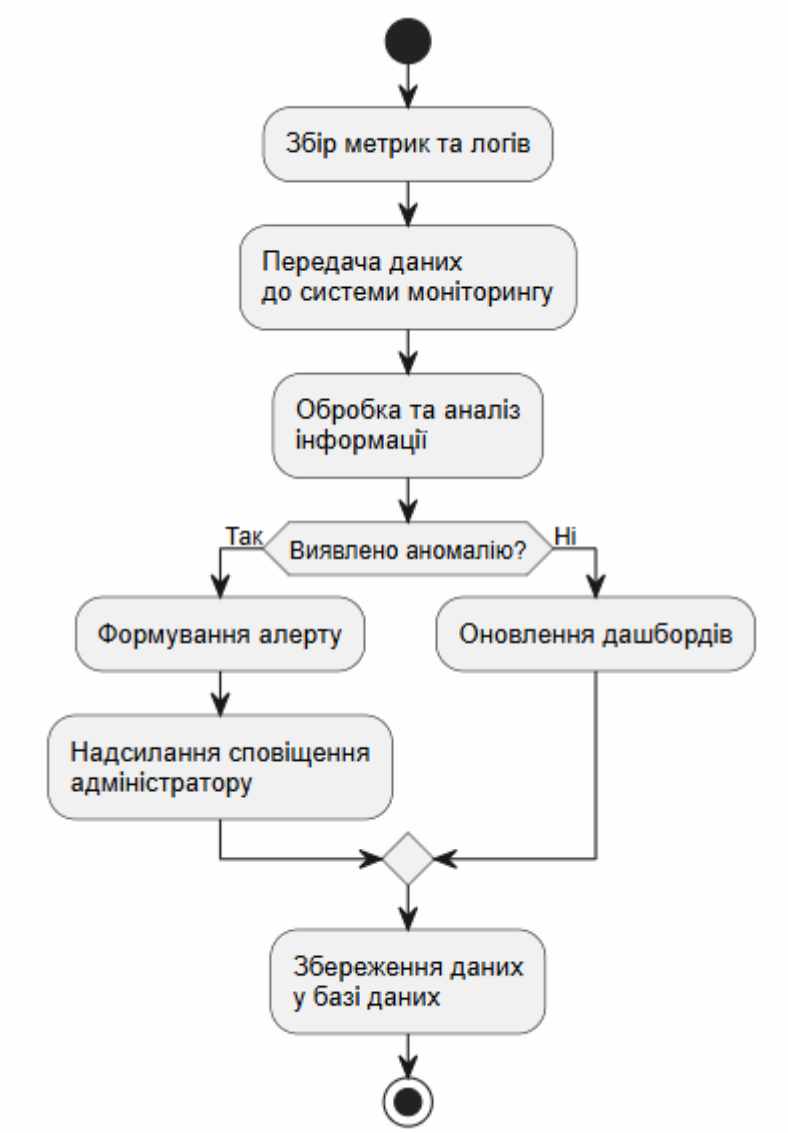


Рисунок 1.4 – UML-діаграма потоку даних у системі моніторингу хмарної інфраструктури

## 1.3 Огляд сучасних інструментів моніторингу (Prometheus, Grafana)

У сучасних хмарних інфраструктурах системи моніторингу є невід'ємною складовою забезпечення стабільної, безпечної та безперервної роботи інформаційних сервісів. З розвитком хмарних технологій, мікросервісної архітектури та контейнеризації збільшилась кількість компонентів, які необхідно контролювати в режимі реального часу. Саме тому використання засобів

моніторингу стало важливим для ефективного адміністрування інфраструктури та своєчасного реагування на можливі проблеми.

Системи моніторингу дозволяють автоматизувати процес збору метрик, аналізу продуктивності серверів, контролю стану мережі, обробки журналів подій та виявлення аномалій у роботі сервісів. Такі системи забезпечують централізоване отримання інформації про стан усіх компонентів інфраструктури та дозволяють адміністраторам оперативно реагувати на перевантаження, помилки або збої у функціонуванні системи.

Prometheus – це сучасна open-source система збору та обробки метрик, яка широко використовується у DevOps-середовищах, контейнеризованих системах та хмарних платформах. Основний принцип роботи Prometheus базується на періодичному опитуванні серверів та сервісів для отримання метрик. Система самостійно звертається до визначених джерел інформації через певні проміжки часу та зберігає отримані дані у вигляді часових рядів. Prometheus підтримує спеціальну мову запитів PromQL, яка дозволяє виконувати складний аналіз метрик, формувати статистику, будувати графіки та виявляти аномальні зміни у роботі системи. Однією з головних переваг цієї системи є її висока продуктивність та можливість ефективно працювати у великих розподілених інфраструктурах (рис. 1.5).

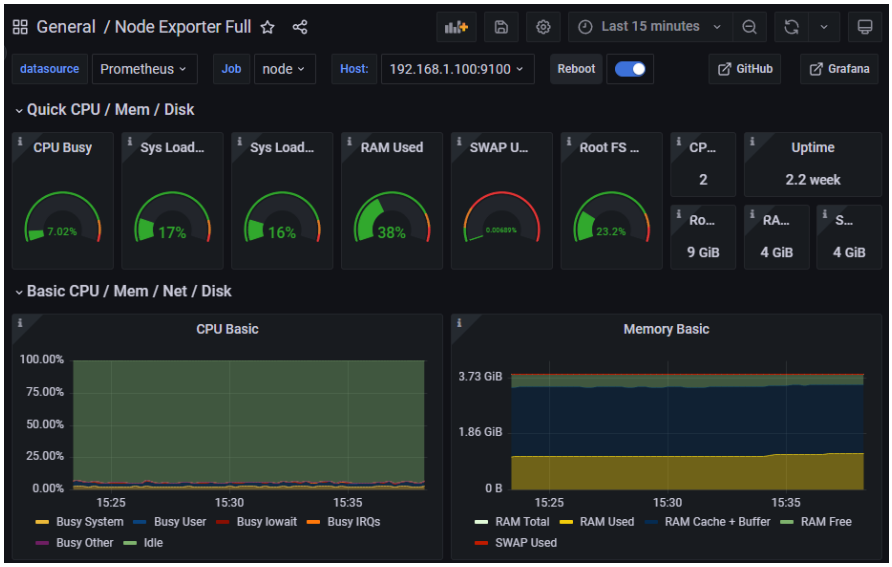


Рисунок 1.5 – Інтерфейс Grafana для моніторингу основних показників хмарної інфраструктури

Для отримання інформації з серверів та сервісів Prometheus використовує exporters. Exporter є спеціальним програмним компонентом, який збирає дані про стан системи та надає їх Prometheus у відповідному форматі. Найчастіше використовуються Node Exporter для моніторингу серверів Linux, PostgreSQL Exporter для баз даних, Docker Exporter для контейнерів та Kubernetes Exporter для контейнеризованих платформ. Завдяки exporters система може отримувати інформацію про навантаження процесора, використання оперативної пам'яті, стан дискової системи, мережеву активність та інші параметри інфраструктури.

Для візуалізації отриманих даних часто використовується Grafana. Ця система дозволяє створювати інтерактивні дашборди, які забезпечують наочне представлення інформації про стан інфраструктури. Grafana підтримує інтеграцію з різними джерелами даних, включаючи Prometheus, PostgreSQL, Elasticsearch та CloudWatch. Використання дашбордів значно спрощує процес аналізу інформації, оскільки адміністратор може в режимі реального часу спостерігати за навантаженням серверів, доступністю сервісів, кількістю помилок та іншими показниками (рис. 1.6).


Рисунок 1.6 – Інформаційно-аналітична панель Grafana для візуалізації системних метрик і мережевої активності

Alertmanager – це компонент відповідає за обробку алертів та надсилання сповіщень адміністраторам у разі виникнення критичних ситуацій. Alertmanager дозволяє налаштовувати правила оповіщення, групувати повідомлення та усувати дублювання алертів. Сповіщення можуть надсилатися через електронну пошту, Telegram, Slack, Discord або інші канали комунікації. Завдяки цьому адміністратори отримують можливість швидко реагувати на проблеми та мінімізувати ризики простою системи.

Окрім open-source рішень, у сучасних хмарних інфраструктурах активно використовуються хмарні системи моніторингу, зокрема Amazon CloudWatch. CloudWatch є сервісом моніторингу платформи Amazon Web Services, який забезпечує збір метрик, аналіз журналів подій та контроль стану ресурсів AWS. Система підтримує моніторинг віртуальних машин, контейнерів, серверless-сервісів, баз даних та мережевих компонентів. Однією з головних переваг CloudWatch є глибока інтеграція з екосистемою AWS та можливість автоматичного масштабування моніторингу відповідно до змін у хмарному середовищі.

Іструменти моніторингу мають різні підходи до збору та аналізу даних. Наприклад, Prometheus орієнтований на високопродуктивний збір метрик і чудово підходить для Kubernetes та контейнеризованих середовищ, тоді як Grafana забезпечує ефективну візуалізацію даних і побудову дашбордів. CloudWatch переважно використовується в AWS-інфраструктурі та забезпечує інтеграцію з іншими сервісами Amazon.

Особливу популярність у сучасних системах отримали open-source рішення. Їх використання дозволяє організаціям уникати значних ліцензійних витрат, адаптувати системи моніторингу під власні потреби та інтегрувати їх із внутрішніми сервісами. Крім того, open-source системи активно підтримуються спільнотою розробників, що забезпечує регулярне оновлення функціональності та швидке виправлення помилок.

Для визначення особливостей сучасних систем моніторингу доцільно провести порівняльний аналіз найбільш поширених інструментів, які використовуються у хмарних та корпоративних інфраструктурах. Порівняння

дозволяє оцінити можливості систем щодо збору метрик, візуалізації даних, масштабованості та інтеграції з сучасними DevOps і Cloud-платформами (табл. 1.1).

Таблиця 1.1 – Порівняльна характеристика сучасних систем моніторингу

| **Критерій** | **Prometheus** | **Grafana** | **Zabbix** | **Nagios** | **CloudWatch** |
|---|---|---|---|---|---|
| Тип системи | Система збору та обробки метрик | Платформа візуалізації даних | Комплексна система моніторингу | Система моніторингу серверів і мереж | Хмарний сервіс моніторингу AWS |
| Основне призначення | Моніторинг контейнерів, сервісів та хмарних систем | Побудова дашбордів і графіків | Моніторинг серверів, мереж та сервісів | Контроль доступності сервісів | Моніторинг ресурсів AWS |
| Тип ліцензії | Open-source | Open-source | Open-source | Open-source | Комерційний сервіс |
| Архітектура | Pull-модель збору метрик | Працює з зовнішніми джерелами даних | Agent-based | Plugin-based | Вбудована хмарна архітектура |
| Збір метрик | Через exporters | Не збирає метрики самостійно | Через агенти | Через плагіни | Автоматичний збір AWS-метрик |
| Підтримка контейнерів | Висока | Висока | Середня | Обмежена | Висока |
| Інтеграція з Kubernetes | Повна підтримка | Повна підтримка | Часткова підтримка | Обмежена | Підтримуєтьс я в AWS EKS |
| Візуалізація даних | Базова | Розширена | Вбудовані графіки | Мінімальна | Вбудовані графіки |
| Дашборди | Обмежені | Інтерактивні та гнучкі | Стандартні | Прості | AWS Dashboard |
| Система алертингу | Alertmanager | Через інтеграції | Вбудована | Вбудована | AWS Alarms |
| Масштабованість | Висока | Висока | Середня | Низька | Висока |

Продовження таблиці 1.1

| **Критерій** | **Prometheus** | **Grafana** | **Zabbix** | **Nagios** | **CloudWatch** |
|---|---|---|---|---|---|
| Продуктивність | Висока при роботі з часовими рядами | Залежить від джерела даних | Середня | Середня | Висока |
| Підтримка логів | Через інтеграції | Через Loki/Elastic | Часткова | Обмежена | Підтримується |
| Підтримка трейсів | Через OpenTelemetry | Через Tempo | Ні | Ні | Частково |
| Простота налаштування | Середня | Висока | Середня | Складна | Висока в AWS |
| Гнучкість конфігурації | Висока | Висока | Висока | Середня | Обмежена AWS |
| Інтеграція з DevOps | Висока | Висока | Середня | Низька | Висока |
| Вартість використання | Безкоштовно | Безкоштовно | Безкоштовно | Безкоштовно | Оплата за використання |
| Основні переваги | Швидкість, Kubernetes, PromQL | Зручна аналітика та дашборди | Комплексний моніторинг | Надійність та простота | Інтеграція з AWS |
| Основні недоліки | Потребує окремої візуалізації | Не збирає метрики самостійно | Складність масштабування | Застарілий інтерфейс | Прив’язка до AWS |

Проведене порівняння сучасних систем моніторингу показало, що кожен із розглянутих інструментів має власні особливості, переваги та сфери застосування. Open-source рішення, зокрема Prometheus і Grafana, характеризуються високою гнучкістю, масштабованістю та широкими можливостями інтеграції з контейнеризованими та хмарними середовищами. На основі проведеного аналізу можна зробити висновок, що для побудови інформаційно-аналітичної системи моніторингу хмарної інфраструктури доцільним є використання Prometheus у поєднанні з Grafana, оскільки така комбінація забезпечує ефективний збір метрик, централізований моніторинг та зручну візуалізацію даних.

# Висновок до розділу 1

Розглянуто теоретичні основи моніторингу хмарної інфраструктури та проаналізовано сучасні підходи до організації хмарних середовищ. Досліджено поняття хмарної інфраструктури, її основні складові та особливості функціонування в умовах сучасних інформаційних систем. Визначено, що використання хмарних технологій забезпечує високу масштабованість, гнучкість і доступність сервісів, проте одночасно створює нові виклики у сфері адміністрування, контролю та забезпечення стабільності роботи інфраструктури.

Розглянуто основні моделі хмарних обчислень — IaaS, PaaS та SaaS, а також типи хмарних середовищ, зокрема public cloud, private cloud і hybrid cloud. Встановлено, що вибір моделі та типу хмари залежить від вимог до безпеки, продуктивності, рівня контролю над ресурсами та особливостей функціонування інформаційної системи.

Розглянуто основні типи даних моніторингу, включаючи метрики, журнали подій, трейси та системні події. Проаналізовано ключові показники продуктивності системи, такі як навантаження на процесор, використання оперативної пам’яті, стан дискової підсистеми, мережева активність, uptime та response time. Визначено, що ефективний моніторинг дозволяє своєчасно виявляти збої, аналізувати продуктивність сервісів та забезпечувати стабільність роботи хмарного середовища.

Проведено огляд сучасних систем моніторингу, серед яких Prometheus, Grafana, Zabbix, Nagios та CloudWatch. У результаті порівняльного аналізу встановлено, що найбільш ефективним рішенням для побудови інформаційно-аналітичної системи моніторингу хмарної інфраструктури є використання Prometheus у поєднанні з Grafana. Такий підхід забезпечує високопродуктивний збір метрик, централізований моніторинг, гнучку систему алертингу та зручну візуалізацію даних у вигляді дашбордів.

# 2 ПРОЄКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ

## 2.1 Формування вимог до системи моніторингу

Одним із етапів проєктування інформаційно-аналітичної системи моніторингу хмарної інфраструктури є формування функціональних вимог. Саме функціональні вимоги визначають основні можливості системи, її поведінку та перелік задач, які вона повинна виконувати в процесі експлуатації. Від правильності формування вимог залежить ефективність роботи системи, її масштабованість, стабільність та зручність використання.

У межах розроблюваної системи моніторингу основна увага приділяється забезпеченню безперервного контролю за станом хмарної інфраструктури, своєчасному виявленню проблем та наданню адміністраторам актуальної інформації про роботу серверів і сервісів.

Система повинна автоматично отримувати дані про стан інфраструктури з різних джерел, включаючи сервери, віртуальні машини, контейнери, бази даних та мережеві сервіси. Збір метрик має здійснюватися у режимі реального часу через спеціалізовані компоненти моніторингу, зокрема exporters. До основних метрик, які необхідно збирати, належать навантаження на процесор, використання оперативної пам’яті, стан дискової системи, мережева активність, доступність сервісів та швидкість обробки запитів.

Система повинна забезпечувати централізоване збереження метрик, журналів подій та інформації про алерти. Збереження історичних даних дозволяє аналізувати зміни навантаження, проводити аудит роботи інфраструктури та прогнозувати можливі перевантаження системи. Для реалізації цієї функції доцільно використовувати базу даних PostgreSQL, яка забезпечує надійність, продуктивність та підтримку роботи з великими обсягами інформації.

Система моніторингу повинна надавати користувачам зручний графічний інтерфейс для перегляду стану інфраструктури у реальному часі. Візуалізація реалізується за допомогою дашбордів Grafana, які дозволяють відображати

графіки, діаграми, таблиці та інші аналітичні елементи. Завдяки цьому адміністратори можуть швидко оцінювати продуктивність системи та виявляти проблемні компоненти інфраструктури.

У разі виникнення проблем система повинна формувати повідомлення та надсилати їх адміністраторам через визначені канали зв’язку, наприклад електронну пошту або месенджери. Використання алертингу дозволяє оперативно реагувати на інциденти та мінімізувати ризик простою сервісів.

Система також повинна підтримувати аналіз стану інфраструктури. Це передбачає обробку отриманих метрик та визначення загального рівня навантаження системи. Аналіз даних дозволяє виявляти аномальні зміни у роботі серверів, прогнозувати перевантаження та визначати потенційні проблеми до моменту виникнення критичних збоїв. Для цього можуть використовуватися як статистичні методи, так і алгоритми інтелектуального аналізу даних.

Однією з функціональних вимог є авторизація користувачів. Система повинна забезпечувати контроль доступу до інформації та підтримувати механізми автентифікації користувачів. Це необхідно для захисту конфіденційних даних моніторингу та обмеження доступу до адміністративних функцій системи. Користувачі повинні мати різні рівні доступу залежно від їх ролі в системі.

Для більш структурованого представлення основних можливостей інформаційно-аналітичної системи моніторингу хмарної інфраструктури доцільно сформувати таблицю функціональних вимог. Вона дозволяє визначити ключові функції системи, їх призначення та роль у забезпеченні ефективного моніторингу хмарного середовища (табл. 2.1).

Таблиця 2.1 – Функціональні вимоги до системи моніторингу хмарної інфраструктури

| № | Функціональна вимога | Опис функції |
|---|---|---|
| 1 | Збір метрик | Автоматичне отримання інформації про стан серверів, контейнерів, баз даних та мережевих сервісів |
| 2 | Моніторинг CPU | Контроль навантаження на процесор та аналіз продуктивності серверів |
| 3 | Моніторинг RAM | Відстеження використання оперативної пам’яті системи |
| 4 | Моніторинг Disk | Контроль дискового простору, швидкості читання та запису даних |
| 5 | Моніторинг Network | Аналіз мережевої активності, затримок та пропускної здатності |
| 6 | Зберігання даних | Централізоване збереження метрик, логів та історичних даних |
| 7 | Візуалізація даних | Відображення інформації у вигляді графіків, таблиць та дашбордів |
| 8 | Формування алертів | Автоматичне створення сповіщень у разі виникнення критичних ситуацій |
| 9 | Аналіз стану інфраструктури | Оцінка продуктивності системи та виявлення аномалій |
| 10 | Авторизація користувачів | Забезпечення автентифікації та розмежування прав доступу |
| 11 | Формування звітів | Створення аналітичних звітів про роботу інфраструктури |
| 12 | Підтримка масштабування | Забезпечення роботи системи при збільшенні кількості компонентів інфраструктури |
| 13 | Централізований моніторинг | Об’єднання даних із різних джерел в єдину систему контролю |
| 14 | Інтеграція з Prometheus | Підключення системи збору та аналізу метрик |
| 15 | Інтеграція з Grafana | Підключення системи візуалізації та дашбордів |

Сформовані функціональні вимоги визначають основні можливості інформаційно-аналітичної системи моніторингу та забезпечують основу для подальшого проєктування архітектури системи. Реалізація наведених функцій дозволить забезпечити централізований контроль за станом хмарної

інфраструктури, автоматизувати процес аналізу метрик та підвищити ефективність адміністрування інформаційного середовища.

Поряд із функціональними вимогами важливу роль у процесі проєктування інформаційно-аналітичної системи моніторингу відіграють нефункціональні вимоги. Вони визначають характеристики якості системи, рівень її продуктивності, стабільності, безпеки та зручності використання. Нефункціональні вимоги безпосередньо впливають на ефективність роботи системи в умовах реальної експлуатації та забезпечують можливість її використання у масштабних хмарних інфраструктурах (табл. 2.2).

Таблиця 2.2 – Нефункціональні вимоги до системи моніторингу хмарної інфраструктури

| № | **Нефункціональна вимога** | **Опис вимоги** |
|---|---|---|
| 1 | Швидкодія | Система повинна забезпечувати швидкий збір, обробку та відображення метрик у режимі реального часу |
| 2 | Продуктивність | Обробка великої кількості метрик та запитів без значного зниження швидкості роботи |
| 3 | Масштабованість | Можливість збільшення кількості серверів, сервісів та джерел даних без втрати продуктивності |
| 4 | Гнучкість | Підтримка інтеграції нових компонентів та сервісів моніторингу |
| 5 | Безпека | Захист даних моніторингу, використання механізмів автентифікації та контролю доступу |
| 6 | Конфіденційність | Забезпечення обмеження доступу до службової та аналітичної інформації |
| 7 | Надійність | Стабільна робота системи навіть у разі високого навантаження або часткових збоїв |
| 8 | Відмовостійкість | Забезпечення безперервної роботи системи при виникненні помилок окремих компонентів |
| 9 | Доступність | Постійна доступність системи моніторингу для адміністраторів та користувачів |
| 10 | UX/UI | Забезпечення зручного та інтуїтивно зрозумілого інтерфейсу користувача |

Продовження таблиці 2.2

| | | |
|---|---|---|
| 11 | Зручність візуалізації | Наявність графіків, дашбордів та аналітичних панелей для швидкого аналізу даних |
| 12 | Сумісність | Підтримка роботи з Prometheus, Grafana, PostgreSQL та іншими сервісами |
| 13 | Автоматизація | Автоматичне формування алертів та оновлення даних моніторингу |
| 14 | Розширюваність | Можливість подальшого вдосконалення та додавання нових функцій |
| 15 | Централізоване адміністрування | Можливість керування всіма компонентами системи через єдиний інтерфейс |

Наведені нефункціональні вимоги визначають основні характеристики якості інформаційно-аналітичної системи моніторингу та забезпечують її ефективне функціонування у хмарному середовищі. Дотримання цих вимог дозволить створити продуктивну, надійну та масштабовану систему, здатну забезпечувати стабільний моніторинг інфраструктури, своєчасне виявлення проблем та зручну взаємодію користувачів із сервісом.

## 2.2 Розробка архітектури системи

Архітектура інформаційно-аналітичної системи моніторингу хмарної інфраструктури передбачає побудову багаторівневої структури, у якій кожен компонент виконує окрему функцію: збір метрик, передавання даних, їх зберігання, обробку, візуалізацію та надання користувачу аналітичної інформації. Такий підхід дозволяє забезпечити централізований контроль за станом хмарних ресурсів, своєчасне виявлення відхилень у роботі системи та оперативне реагування на можливі збої.

Загальна архітектура системи складається з декількох взаємопов’язаних рівнів. Перший рівень представлений об’єктами моніторингу, до яких належать сервери, віртуальні машини, контейнери, бази даних та мережеві сервіси. Саме з цих компонентів система отримує інформацію про поточний стан інфраструктури.

Другий рівень формують exporters, які збирають технічні показники з окремих компонентів і передають їх у форматі, придатному для подальшої обробки. Третій рівень представлений Prometheus, який виконує централізований збір метрик, їх попереднє збереження та обробку за допомогою запитів PromQL.

Окреме місце в архітектурі займає PostgreSQL, яка використовується для збереження структурованих даних, інформації про користувачів, налаштування системи, історію алертів, звіти та результати аналітичної обробки. Використання PostgreSQL дозволяє забезпечити цілісність даних, зручність доступу до історичної інформації та можливість подальшого формування звітності.

Для візуалізації результатів моніторингу використовується Grafana. Вона підключається до джерел даних, зокрема Prometheus і PostgreSQL, та забезпечує відображення інформації у вигляді графіків, таблиць, панелей і дашбордів. Завдяки цьому адміністратор або користувач системи може швидко оцінити стан хмарної інфраструктури, виявити перевантажені компоненти та проаналізувати динаміку зміни показників.

Backend-частина системи відповідає за обробку запитів користувача, взаємодію з базою даних, авторизацію, формування звітів та реалізацію бізнес-логіки. Вона виступає проміжним шаром між frontend-інтерфейсом, базою даних та сервісами моніторингу. Frontend забезпечує зручну взаємодію користувача із системою, надає доступ до дашбордів, звітів, журналів подій та налаштувань моніторингу.

Збір метрик у системі відбувається автоматично. Exporters встановлюються на відповідні об’єкти моніторингу або інтегруються з окремими сервісами. Вони отримують інформацію про навантаження CPU, використання RAM, стан дисків, мережеву активність, доступність сервісів та інші показники. Prometheus через задані проміжки часу звертається до exporters, отримує актуальні метрики та зберігає їх у вигляді часових рядів. Надалі ці дані можуть використовуватися для побудови графіків, аналізу стану системи та формування алертів.

Потоки даних у системі організовані таким чином, щоб забезпечити безперервне надходження інформації від об’єктів моніторингу до засобів аналізу

та візуалізації. Спочатку дані формуються на рівні інфраструктури, потім збираються exporters, після чого передаються до Prometheus. Далі інформація може надходити до Grafana для відображення або до backend-модуля для додаткової обробки, збереження в PostgreSQL і формування звітів.

API-взаємодія у системі використовується для обміну даними між frontend, backend, базою даних та сервісами моніторингу. Frontend надсилає запити до backend для отримання інформації про користувачів, звіти, налаштування або результати аналізу. Backend, у свою чергу, звертається до PostgreSQL для роботи з даними та може отримувати інформацію з Prometheus API для аналізу поточних метрик. Такий підхід забезпечує модульність системи та спрощує її подальше розширення.

Алгоритм роботи системи можна описати як послідовність етапів. Спочатку об’єкти хмарної інфраструктури генерують технічні показники. Далі exporters збирають ці показники та надають їх Prometheus. Prometheus періодично отримує метрики, зберігає їх і перевіряє відповідність заданим правилам. Якщо показники перевищують допустимі межі, формується алерт. Паралельно Grafana отримує дані з Prometheus і відображає їх у вигляді дашбордів. Backend забезпечує обробку користувацьких запитів, роботу з базою даних та формування звітності. У результаті користувач отримує актуальну інформацію про стан хмарної інфраструктури в зручному аналітичному вигляді.

Для наочного представлення загальної структури інформаційно-аналітичної системи моніторингу доцільно подати архітектурну схему, яка відображає основні компоненти системи та взаємозв’язки між ними (рис. 2.1).

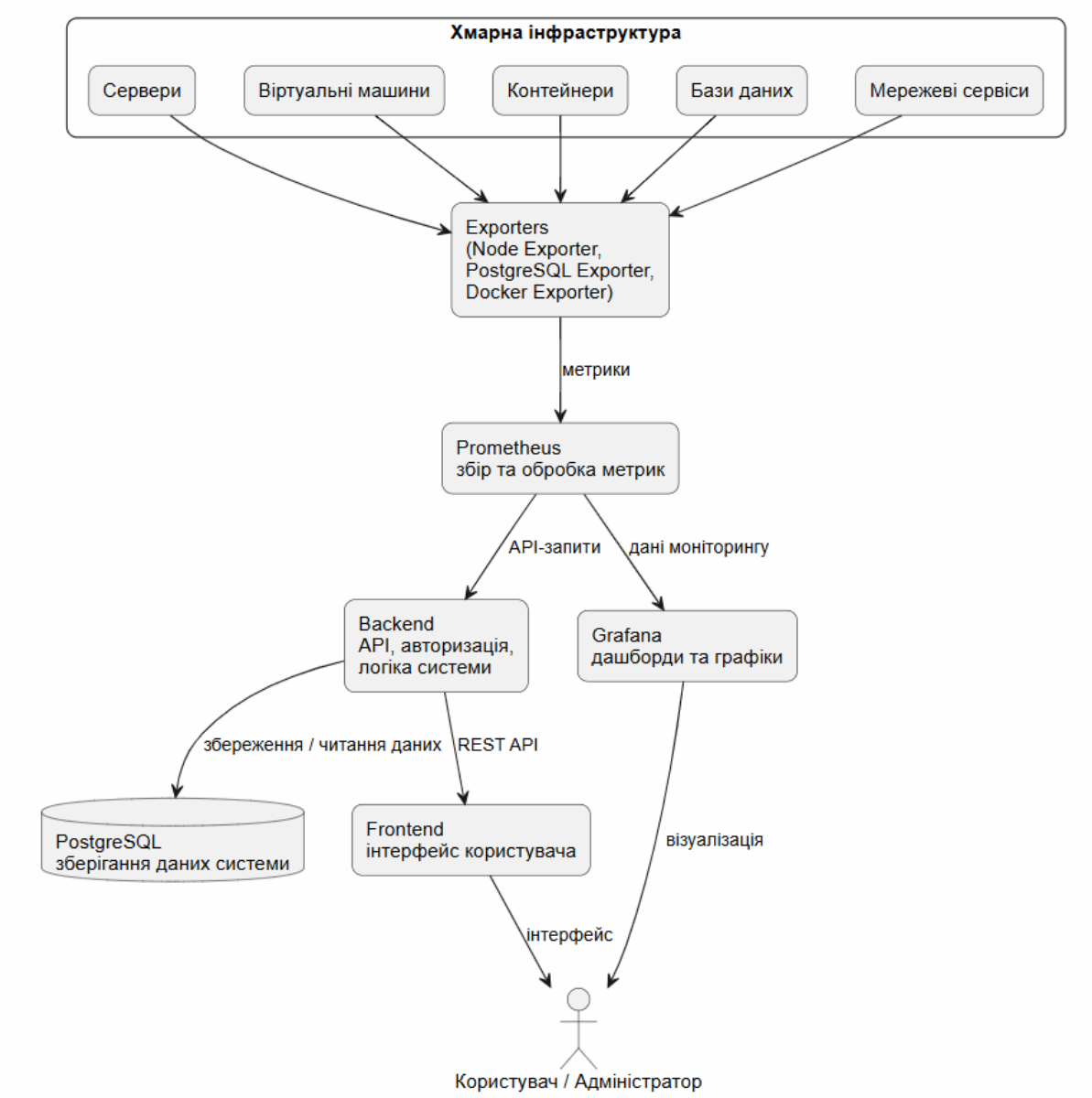


Рисунок 2.1 – Архітектурна схема інформаційно-аналітичної системи моніторингу хмарної інфраструктури

Для деталізації внутрішньої структури системи доцільно використати UML-діаграму компонентів, яка демонструє функціональні модулі системи та принципи їх взаємодії (рис. 2.2).

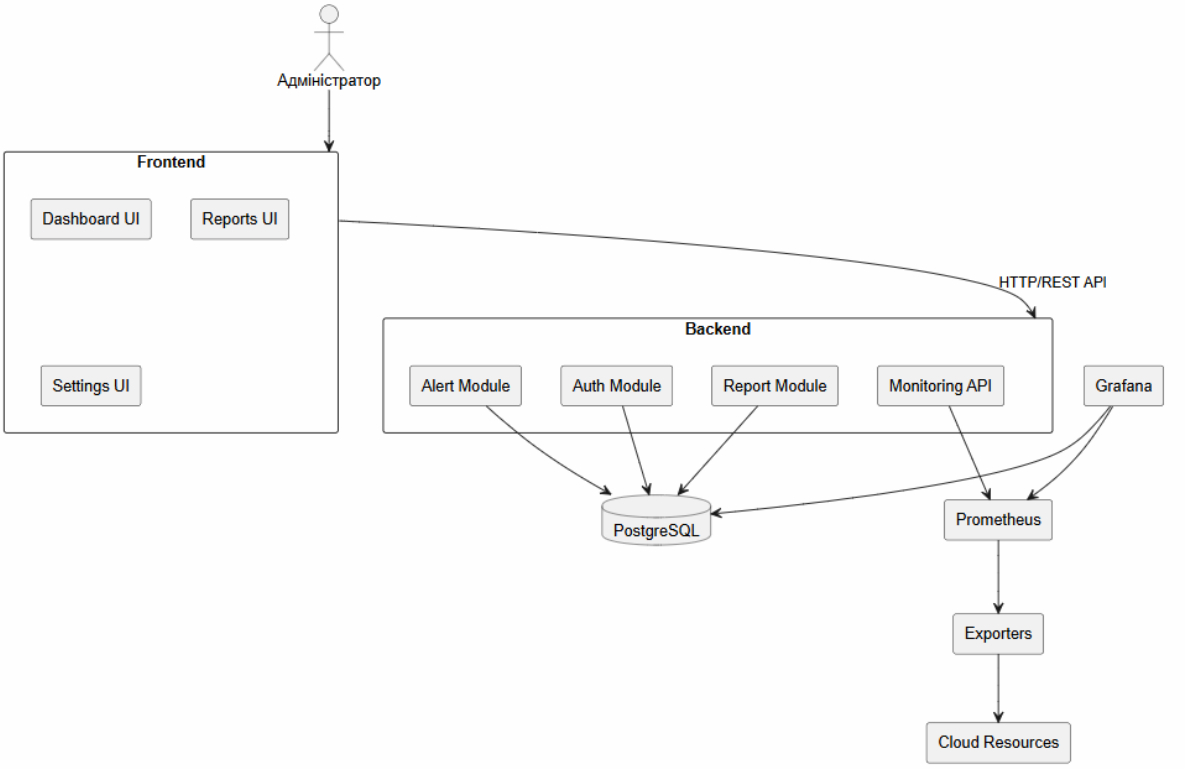


Рисунок 2.2 – UML-діаграма компонентів інформаційно-аналітичної системи моніторингу

Для відображення логіки проходження даних у системі моніторингу доцільно представити блок-схему, яка показує послідовність збору, обробки, збереження та візуалізації метрик хмарної інфраструктури (рис. 2.3).


Рисунок 2.3 – Блок-схема потоку даних у системі моніторингу хмарної інфраструктури

## 2.3 Проєктування бази даних та інтеграція PostgreSQL (Railway)

У цьому підрозділі виконано проєктування структури бази даних для інформаційної системи з урахуванням подальшого розгортання в середовищі PostgreSQL на платформі Railway. Основною метою є побудова узгодженої моделі даних, що забезпечує цілісність, масштабованість і зручність подальшої інтеграції прикладного застосунку [16].

На етапі концептуального проєктування виокремлено ключові сутності предметної області: користувачі, ролі, проєкти, задачі, коментарі, вкладення, журнали активності та системні сповіщення. Між сутностями визначено зв’язки типу «один-до-багатьох» і «багато-до-багатьох», що дозволяє описати реальні сценарії взаємодії користувачів із системою. Для усунення надлишковості та запобігання аномаліям оновлення модель приведено до 3НФ.

На логічному рівні структура бази даних орієнтована на PostgreSQL і передбачає використання первинних ключів (UUID), зовнішніх ключів, обмежень NOT NULL, UNIQUE, CHECK, а також каскадних правил цілісності (ON DELETE CASCADE / ON DELETE SET NULL) залежно від бізнес-правил. Додатково передбачено індексацію полів фільтрації та сортування (дата створення, статус задачі, ідентифікатор користувача), що підвищує продуктивність запитів у робочих сценаріях.

Фізичне проєктування враховує особливості експлуатації на Railway: централізоване зберігання, захищений доступ, підтримка резервного копіювання та прогнозоване горизонтальне зростання навантаження. Таким чином, база даних розробляється як окремий стабільний шар системи, а прикладна логіка взаємодіє з нею через чітко визначені CRUD-операції та транзакційні механізми PostgreSQL.

На рисунку 2.4 представлено концептуальну ER-модель бази даних, що відображає основні сутності та їх зв’язки.

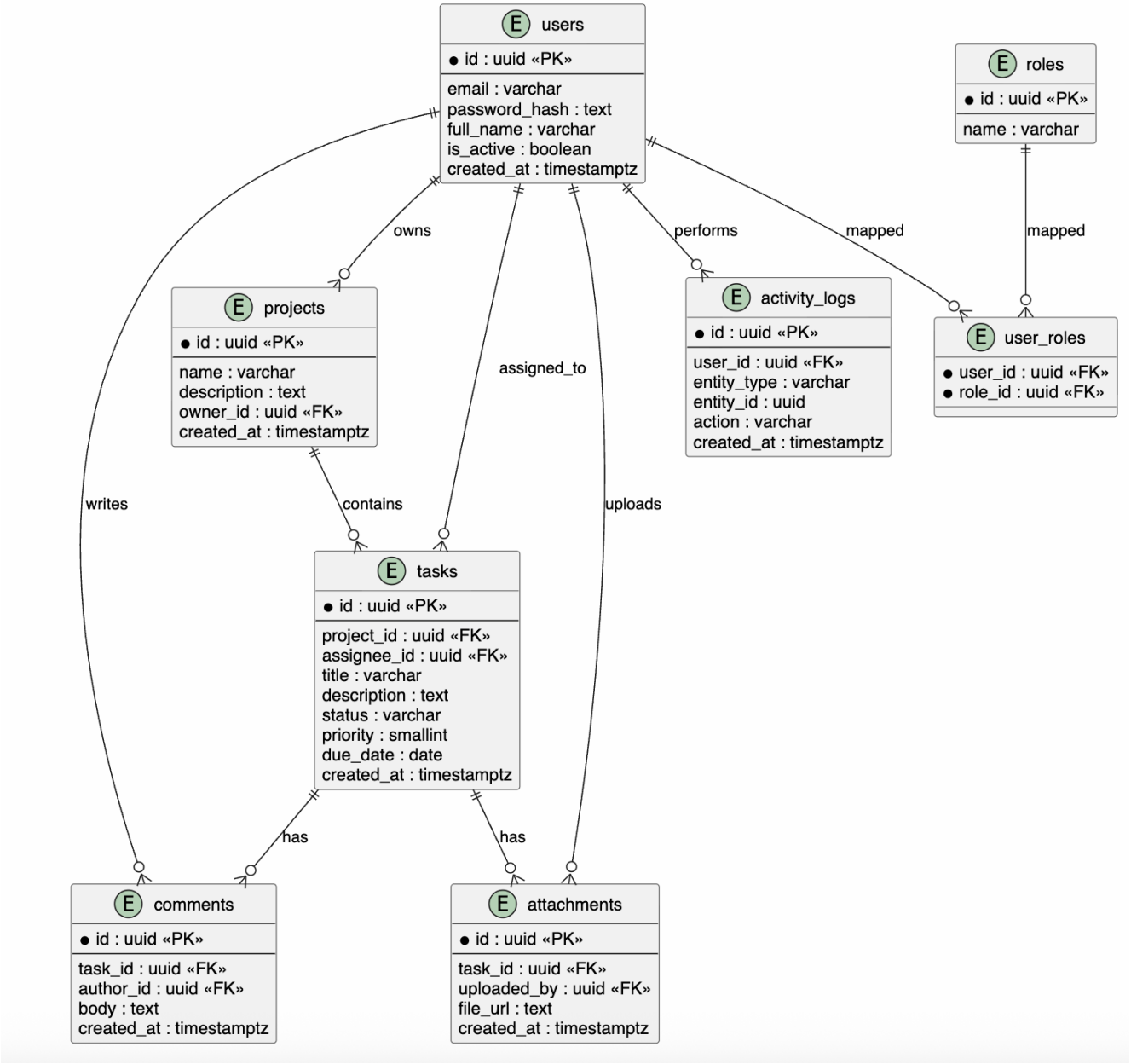


Рисунок 2.4 – ER-модель бази даних інформаційної системи (концептуальний рівень)

Отже, рисунок 2.4 демонструє основу структури даних, яка забезпечує цілісне зберігання інформації про користувачів, проєкти, задачі та журнал дій [18, 19].

Для забезпечення ефективного зберігання інформації в інформаційно-аналітичній системі моніторингу хмарної інфраструктури було спроєктовано структуру бази даних, яка включає набір взаємопов’язаних таблиць. Кожна таблиця відповідає за збереження окремого типу даних, зокрема інформації про користувачів, ролі доступу, метрики моніторингу, журнали подій, алерти та результати аналітичної обробки. Основні таблиці бази даних та їх призначення наведено у табл. 2.3.

Таблиця 2.3

Основні таблиці та їх призначення

| **Таблиця** | **Призначення** | **Ключові поля** |
|---|---|---|
| *users* | зберігання облікових записів користувачів | id, email, password_hash |
| *roles* | ролі доступу в системі | id, name |
| *user_roles* | зв’язок користувачів із ролями | user_id, role_id |
| *projects* | дані про проєкти | id, owner_id, name |
| *tasks* | облік задач у межах проєктів | id, project_id, assignee_id, status |
| *comments* | історія обговорень задач | id, task_id, author_id |
| *attachments* | файли, пов’язані із задачами | id, task_id, file_url |
| *activity_logs* | аудит дій користувачів | id, user_id, action, created_at |

На рисунку 2.5 показано логічну взаємодію прикладного рівня з базою даних PostgreSQL.

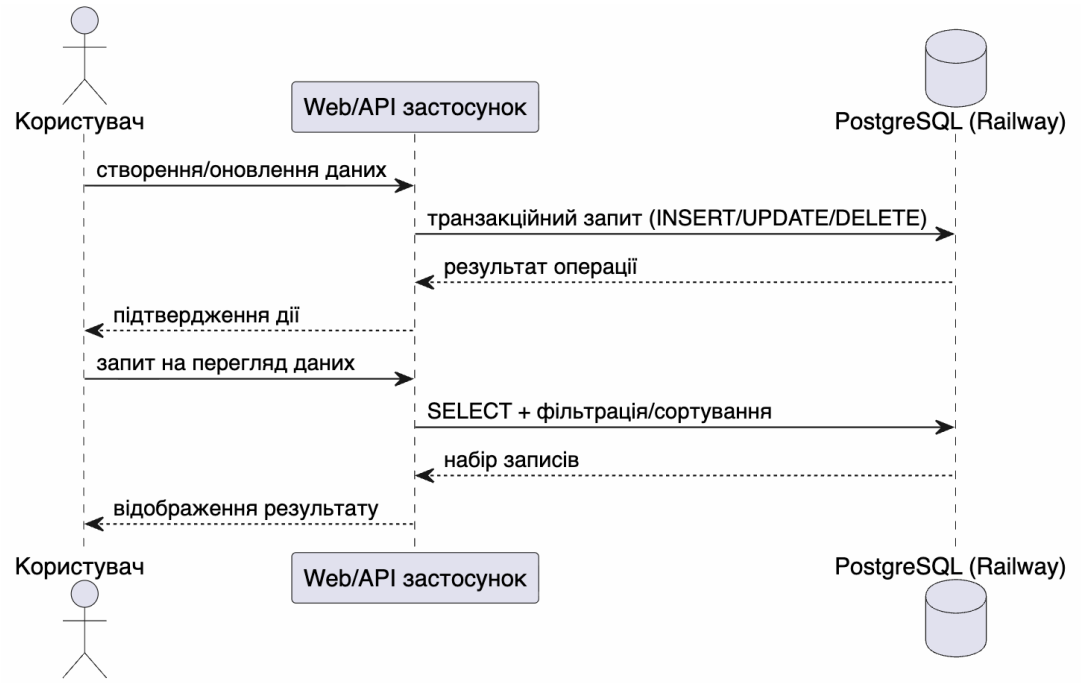


Рисунок 2.5 – Логічна схема обробки даних між застосунком і PostgreSQL

Таким чином, рисунок 2.5 ілюструє транзакційний характер доступу до даних та відокремлення клієнтської й серверної відповідальності.

Під час фізичного проєктування бази даних також особливу увагу приділено забезпеченню цілісності та продуктивності. Для цього на рівні схеми визначено обмеження PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, а також правила каскадної обробки пов’язаних записів. Наприклад, при видаленні проєкту доцільно автоматично видаляти пов’язані задачі (ON DELETE CASCADE), тоді як для історичних журналів дій можна зберігати запис із зануленням посилання на користувача (ON DELETE SET NULL), щоб не втрачати аудит [19 ,20].

Для оптимізації читання даних у типових сценаріях (фільтрація задач за статусом, виконавцем, датою створення) передбачено систему індексів. Зокрема, індекси створюються для полів tasks(project_id), tasks(assignee_id), tasks(status), tasks(created_at), а також для activity_logs(user_id, created_at). Це скорочує час виконання запитів у робочих інтерфейсах, де активно використовуються списки, дашборди та журнали подій. Окремо враховано транзакційність операцій. Критичні бізнес-процеси (створення задачі з початковим коментарем, зміна статусу з логуванням події, масове оновлення атрибутів) виконуються в межах однієї

транзакції, що гарантує узгоджений стан даних навіть у разі часткового збою. Такий підхід мінімізує ризик появи “частково записаних” сутностей і підвищує надійність системи в умовах конкурентного доступу.

Додатково для експлуатації в Railway доцільно застосувати керування конфігурацією через змінні середовища (DATABASE_URL, PGHOST, PGPORT, PGUSER, PGPASSWORD) та механізм міграцій схеми. Використання міграцій забезпечує контрольовану еволюцію структури БД між середовищами розробки, тестування й продакшн, а також спрощує відтворюваність розгортання для всієї команди.

***Приклад DDL-опису таблиць та індексації в PostgreSQL***

```
CREATE TABLE users (
    id UUID PRIMARY KEY,
    email VARCHAR(255) NOT NULL UNIQUE,
    full_name VARCHAR(255) NOT NULL,
    password_hash TEXT NOT NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE TABLE projects (
    id UUID PRIMARY KEY,
    owner_id UUID REFERENCES users(id) ON DELETE SET NULL,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE TABLE tasks (
    id UUID PRIMARY KEY,
    project_id UUID NOT NULL REFERENCES projects(id) ON DELETE
CASCADE,
```

```
    assignee_id UUID REFERENCES users(id) ON DELETE SET NULL,
    title VARCHAR(255) NOT NULL,
    status VARCHAR(32) NOT NULL CHECK (status IN ('new','in_progress','done')),
    priority SMALLINT NOT NULL CHECK (priority BETWEEN 1 AND 5),
    due_date DATE,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE TABLE activity_logs (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(id) ON DELETE SET NULL,
    action VARCHAR(64) NOT NULL,
    entity_type VARCHAR(64) NOT NULL,
    entity_id UUID,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE INDEX idx_tasks_project_id ON tasks(project_id);
CREATE INDEX idx_tasks_assignee_id ON tasks(assignee_id);
CREATE INDEX idx_tasks_status_created_at ON tasks(status, created_at);
CREATE INDEX idx_activity_logs_user_created_at ON activity_logs(user_id, created_at);
```

Це демонструє практичний підхід до побудови структури БД, де одночасно забезпечуються узгодженість даних (через ключі та CHECK-обмеження) і продуктивність вибірок (через цільову індексацію) [21, 22].

## Висновок до розділу 2

У другому розділі розглянуто теоретико-прикладні засади побудови системи моніторингу активності користувачів, орієнтованої на виявлення інсайдерських загроз. Узагальнено підходи до збору та обробки подій безпеки, а також обґрунтовано доцільність використання поведінкового аналізу для виявлення нетипових дій. Показано, що поєднання аудиту подій із аналітичними методами створює основу для своєчасної ідентифікації ризикової активності.

Окрему увагу приділено проєктуванню структури даних: визначено ключові сутності предметної області, їхні зв’язки та правила цілісності. Сформовано логічну й фізичну модель бази даних, орієнтовану на PostgreSQL, із використанням первинних/зовнішніх ключів, перевірок коректності значень і механізмів каскадної обробки пов’язаних записів. Додатково обґрунтовано індексацію як інструмент підвищення продуктивності в типових сценаріях фільтрації, сортування та аналітичного перегляду даних.

Результати другого розділу формують завершену технічну базу для практичної реалізації системи в наступному розділі, зокрема для налаштування підключення PostgreSQL у Railway, виконання міграцій і запуску прикладної логіки. Отримані напрацювання забезпечують узгодженість архітектури, масштабованість рішення та готовність до подальшого експериментального оцінювання ефективності системи моніторингу.

# 3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ

## 3.1 Реалізація системи збору метрик (Prometheus, exporters)

У межах кваліфікаційної роботи вебзастосунок використовується як прикладне середовище для перевірки працездатності системи моніторингу серверної інфраструктури. Його призначення полягає не лише у наданні користувацького функціоналу, а й у формуванні реального навантаження, на якому можна оцінювати повноту збору метрик, стабільність телеметрії та коректність подальшого аналізу. Для розгортання серверної частини обрано хмарну платформу AWS, що дозволяє відтворити умови, наближені до промислової експлуатації, та отримати репрезентативні результати моніторингу [23, 24].

Розроблений вебзастосунок реалізує функції керування інвентарними об’єктами користувача і підтримує повний цикл взаємодії з даними через вебінтерфейс та API-рівень. Архітектура рішення побудована за клієнт-серверною моделлю: клієнтська частина ініціює запити, серверна логіка на Flask обробляє бізнес-операції, а PostgreSQL відповідає за надійне збереження структурованих даних. Такий підхід забезпечує контрольований потік подій, який є необхідним для коректного зняття метрик продуктивності, доступності та відгуку сервісу.

Важливим компонентом є підсистема автентифікації та авторизації користувачів. Під час реєстрації облікових записів застосовується хешування паролів, що відповідає базовим практикам захисту облікових даних. У поєднанні з розмежуванням доступу це знижує ризик компрометації даних і дозволяє оцінювати безпекові аспекти системи в контексті моніторингу подій доступу. Додатково всі ключові операції (створення, редагування, перегляд сутностей) виконуються через REST API, що спрощує трасування запитів і корелювання прикладних подій із системними метриками.

На рівні збереження даних застосовано PostgreSQL з ORM SQLAlchemy, що забезпечує узгоджену роботу з моделями, транзакційність і контроль цілісності. Серверна частина реалізована на Flask, який обрано завдяки простоті

налаштування, прозорій маршрутизації та зручній інтеграції з інструментами спостережуваності. У підсумку отримано технологічну основу, на якій можна коректно реалізувати збір метрик через Prometheus та exporters у наступних підпунктах.

На рисунку 3.1 представлено початковий інтерфейс застосунку, що використовується як функціональне середовище для генерації прикладної активності та подальшого моніторингу.

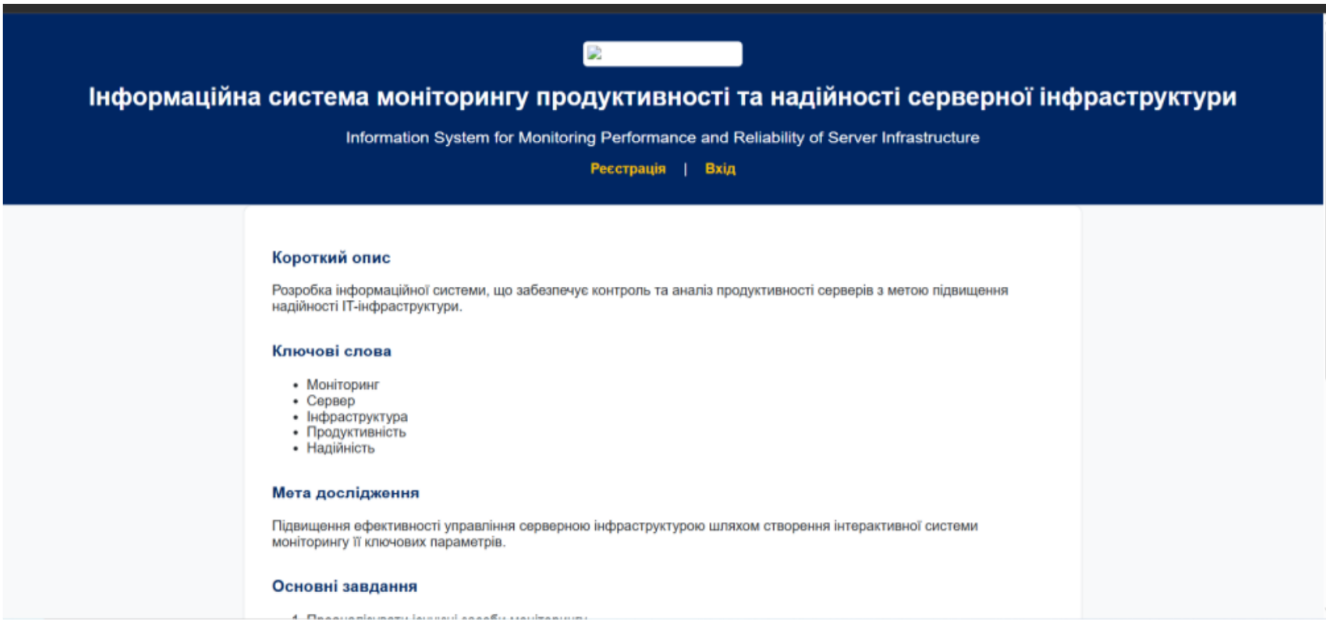


Рисунок 3.1 – Головна сторінка вебзастосунку системи моніторингу серверної інфраструктури

Рисунок 3.1 ілюструє стартову сторінку інформаційної системи, яка в межах дослідження виконує подвійну роль: прикладного сервісу для користувача та тестового джерела подій для підсистеми збору метрик.

Для розгортання серверної частини застосунку обрано AWS EC2 (Elastic Compute Cloud), оскільки цей сервіс надає повноцінний віртуальний сервер із доступом до операційної системи та гнучкими параметрами конфігурації. Такий формат розміщення дозволяє працювати в середовищі, яке за характеристиками близьке до реального production-хостингу, що є важливим для достовірного тестування системи моніторингу.

Суттєвою перевагою EC2 є підтримка різних ОС, зокрема Ubuntu Server 22.04 LTS, яка є стабільною платформою для Python/Flask-рішень. Додатково платформа дозволяє керувати мережевими налаштуваннями на рівні інстансу та відкривати необхідні порти (80, 443, 5000) для роботи вебінтерфейсу, API та сервісних компонентів моніторингу.

Окремо слід відзначити масштабованість ресурсу: у разі зміни навантаження можна оперативно змінювати тип інстансу, обсяг оперативної пам’яті, процесорні характеристики та параметри сховища. Це забезпечує раціональне використання інфраструктури та спрощує адаптацію системи до нових експлуатаційних вимог.

Важливою є й інтеграція EC2 з іншими сервісами AWS, що дає змогу формувати єдиний контур розгортання, безпеки та спостережуваності. Контроль доступу реалізується через SSH-ключі та security groups, де правила з’єднання задаються за IP-адресами й портами, що підвищує загальний рівень захисту серверного середовища.

У межах практичної реалізації було створено інстанс на базі Ubuntu Server 22.04 LTS з призначеною публічною IP-адресою 13.61.16.177, який використано як хост для серверної частини вебзастосунку [25].

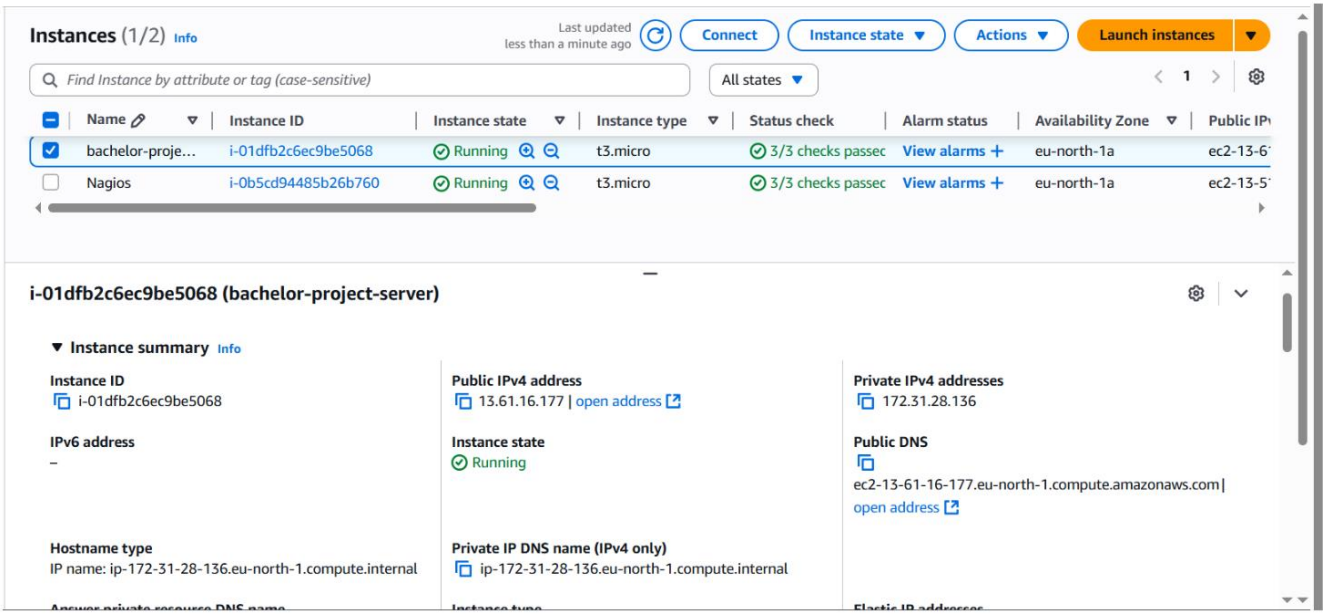


Рисунок 3.2 – Активний EC2-інстанс AWS, розгорнутий для хостингу серверного модуля вебзастосунку

Після створення EC2-інстансу виконано поетапне розгортання серверної частини застосунку. Спочатку встановлено SSH-з’єднання з віртуальним сервером за допомогою приватного ключа:

***ssh -i "aws-key.pem" ubuntu@13.61.16.177***

Далі підготовлено програмне середовище: оновлено пакети, інстальовано python3-pip та необхідні залежності застосунку (Flask, Flask-SQLAlchemy, psycopg2). Після цього вихідний код завантажено з GitHub-репозиторію, а в системі налаштовано змінні середовища, зокрема DATABASE_URL, для підключення до PostgreSQL (Railway). На завершальному етапі застосунок запущено командою python3 app.py з доступом через зовнішню IP-адресу сервера.

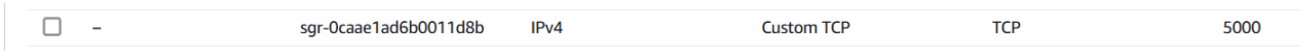


Рисунок 3.3 – Налаштування правила доступу до порту 5000 у Security Group для Flask-сервісу

Окремо на рівні AWS Security Group відкрито порт 5000, що забезпечило зовнішній доступ до Flask-сервісу. Після розгортання виконано перевірку через браузер за адресою http://13.61.16.177:5000/: сторінки реєстрації та авторизації відображалися коректно, а базовий функціонал інвентаризації працював стабільно. Надалі цей інстанс використовувався як цільовий вузол для збору метрик у Prometheus [26].

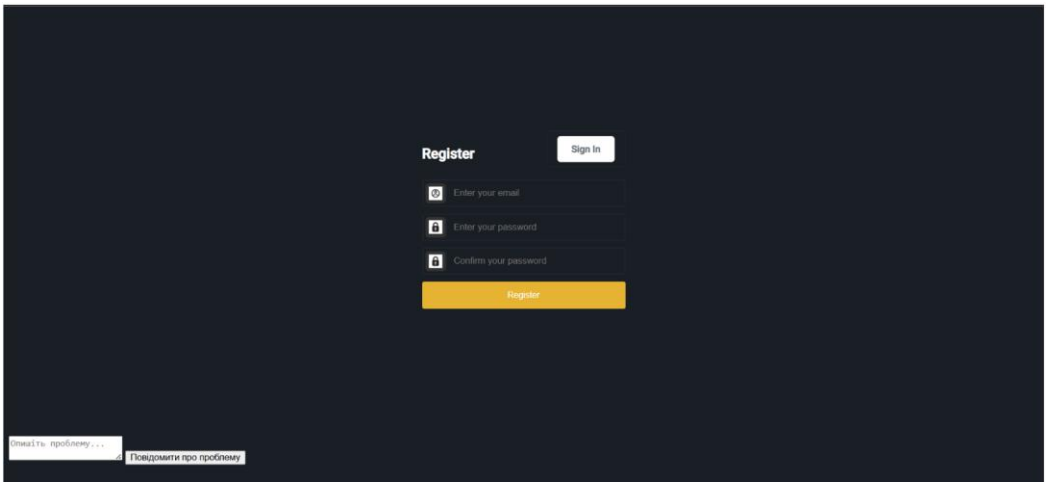


Рисунок 3.4 – Форма реєстрації вебзастосунку після успішного розгортання на EC2

Для організації безперервного контролю стану серверної інфраструктури у проєкті впроваджено Prometheus як базову платформу збору метрик. Це рішення забезпечує регулярне опитування цільових вузлів, збереження часових рядів і подальший аналітичний перегляд показників у реальному часі. Основна мета його використання в роботі полягає у своєчасному виявленні деградації сервісу: перевантаження CPU, нестачі оперативної пам’яті, зростання затримок відповіді та інших відхилень, що впливають на стабільність вебзастосунку.

Перевагою Prometheus є модульний підхід до інтеграції через exporters, завдяки якому можна підключати як системні, так і прикладні джерела телеметрії. Для аналізу метрик застосовується мова запитів PromQL, що дає змогу формувати як прості, так і комбіновані вирази для діагностики стану інфраструктури. Додатково система підтримує інтеграцію з Grafana для побудови дашбордів і з Alertmanager для автоматизованого сповіщення про інциденти, що підвищує оперативність реагування.

У практичній реалізації моніторинговий контур розгорнуто на окремому EC2-інстансі з IP-адресою 13.51.161.180, тоді як вебзастосунок працює на 13.61.16.177. Таке розділення дозволило ізолювати моніторингове навантаження від прикладного, зменшити ризик взаємного впливу компонентів і відтворити типовий сценарій віддаленого спостереження за продуктивним вузлом.

До основних кроків розгортання належали: створення інстансу Ubuntu Server 22.04, підключення через SSH, завантаження дистрибутива Prometheus, налаштування prometheus.yml із ціллю 13.61.16.177:9100, запуск сервера моніторингу та відкриття порту 9090 для доступу до вебінтерфейсу. Після цього інтерфейс Prometheus став доступним за адресою http://13.51.161.180:9090, що підтвердило коректність базової конфігурації.

На рисунку 3.5 показано окремий EC2-вузол, призначений для розміщення компонентів моніторингу.

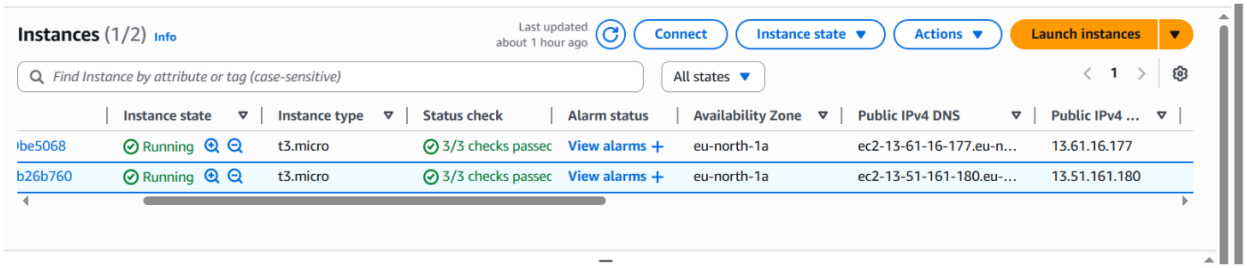


Рисунок 3.5 – Виділений EC2-інстанс AWS для розгортання підсистеми моніторингу Prometheus

Рисунок 3.5 підтверджує реалізацію архітектурного підходу з розділенням прикладного та моніторингового контурів.

На рисунку 3.6 представлено вебінтерфейс Prometheus, доступний після завершення налаштування сервісу.

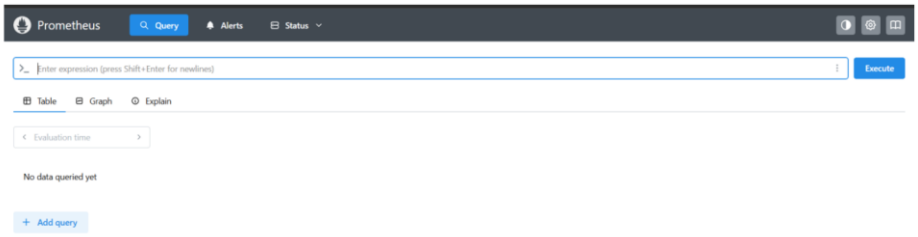


Рисунок 3.6 – Вебінтерфейс Prometheus після успішного запуску на моніторинговому сервері

Таким чином, рисунок 3.6 демонструє готовність системи Prometheus до виконання запитів і подальшого збору метрик із цільового сервера.

Щоб Prometheus міг регулярно отримувати системні показники з вузла, де працює вебзастосунок, на сервері 13.61.16.177 було розгорнуто Node Exporter. Цей компонент надає метрики у форматі, сумісному з Prometheus, і охоплює ключові параметри стану хоста: завантаження процесора, використання оперативної пам’яті та swap, дискову активність, мережевий трафік і тривалість безперервної роботи системи (uptime). Саме ці дані формують базу для оцінки стабільності інфраструктури та раннього виявлення деградації продуктивності.

Встановлення виконано стандартним сценарієм: завантаження архіву з офіційного репозиторію, розпакування, запуск node_exporter як окремого процесу та перевірка endpoint http://13.61.16.177:9100/metrics. Після запуску сервіс почав

експортувати часові ряди, доступні для зчитування Prometheus через механізм scrape. Це підтверджує коректність інтеграції між прикладним сервером і підсистемою моніторингу.

У вебінтерфейсі Prometheus (http://13.51.161.180:9090) на вкладці Targets вузол 13.61.16.177:9100 відображається у стані UP, що означає стабільне підключення та успішний регулярний збір метрик. Отримані показники використовуються як основа для подальшого аналізу динаміки навантаження, порівняння режимів роботи застосунку та формування умов сповіщення про інциденти.

З практичної точки зору обрані метрики є найбільш репрезентативними для оцінки серверної надійності під час тестування: вони дозволяють фіксувати як короткочасні піки навантаження, так і довготривалі тренди виснаження ресурсів. Це створює технічне підґрунтя для обґрунтованих висновків щодо працездатності та масштабованості розгорнутої системи [26, 27].

На рисунку 3.7 наведено фрагмент метрик, що експортуються Node Exporter із сервера застосунку.

```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 3.2682e-05
go_gc_duration_seconds{quantile="0.25"} 3.7978e-05
go_gc_duration_seconds{quantile="0.5"} 4.1189e-05
go_gc_duration_seconds{quantile="0.75"} 4.5835e-05
go_gc_duration_seconds{quantile="1"} 0.000104532
go_gc_duration_seconds_sum 2.118012175
go_gc_duration_seconds_count 48730
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 9
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.22.2"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 2.089064e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 9.1916954552e+10
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 2.075466e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 1.173216692e+09
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 3.227376e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 2.089064e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 4.087808e+06
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 3.825664e+06
# HELP go_memstats_heap_objects Number of allocated objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 12138
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.
# TYPE go_memstats_heap_released_bytes gauge
go_memstats_heap_released_bytes 3.145728e+06
```

Рисунок 3.7 – Вивід системних метрик Node Exporter на endpoint /metrics сервера застосунку

Рисунок 3.7 підтверджує доступність системних метрик для подальшого збирання та аналізу.

На рисунку 3.8 показано результат перевірки підключення цільового вузла у вебінтерфейсі Prometheus.

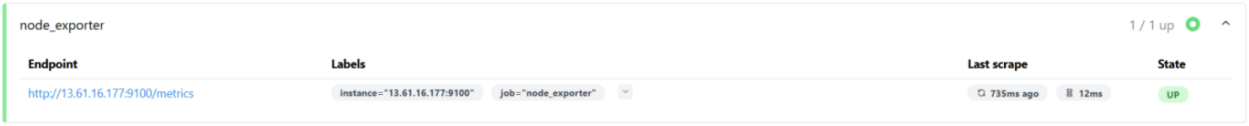


Рисунок 3.8 – Стан цільового вузла в Prometheus Targets після підключення Node Exporter

Рисунок 3.8 демонструє успішне встановлення з’єднання між Prometheus і сервером застосунку в режимі регулярного моніторингу.

## 3.2 Реалізація зберігання та обробки даних (PostgreSQL, Railway)

У межах практичної реалізації вебзастосунку одним із ключових завдань стало побудувати надійний механізм збереження та обробки користувацьких даних. Для цього використано керований хмарний сервіс Railway із розгортанням PostgreSQL, що забезпечило швидке підключення бази даних до серверної частини застосунку без складного ручного адміністрування інфраструктури. Такий підхід дозволив зосередитися на прикладній логіці системи, зберігаючи при цьому стабільність та керованість середовища даних.

Вибір Railway обґрунтований тим, що платформа автоматизує створення PostgreSQL-інстансу, формує параметри підключення та надає їх через інтерфейс змінних середовища. У результаті відпадає потреба в окремому налаштуванні серверів БД, мережевих маршрутів і базових операцій безпеки на стартовому етапі. Додатковими перевагами є централізоване керування конфігурацією, вбудовані механізми резервування та можливість масштабування, що важливо для безперервної роботи вебсервісу при зміні навантаження [27, 28].

Початкове підключення виконується через створення облікового запису Railway (у роботі використано авторизацію через GitHub), після чого створюється проєкт і додається сервіс PostgreSQL. Після активації інстансу платформа автоматично генерує дані доступу (host, port, database, user, password) і рядок з’єднання, який далі використовується у Flask-застосунку через змінну DATABASE_URL.

На рисунку 3.9 показано приклад успішного розгортання PostgreSQL-сервісу в Railway та готовність середовища до інтеграції із серверною частиною застосунку.

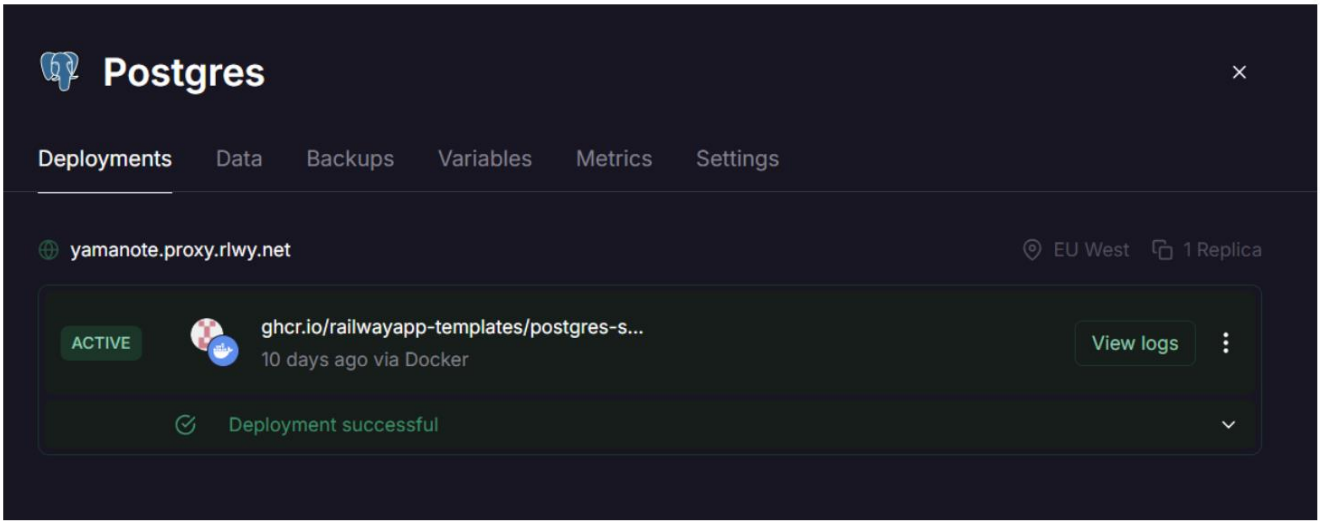


Рисунок 3.9 – Успішне розгортання PostgreSQL-сервісу в Railway із автоматично сформованими параметрами підключення

Після ініціалізації PostgreSQL-інстансу платформа Railway автоматично формує рядок підключення, який включає всі критичні параметри доступу: ім’я користувача, пароль, адресу хоста, порт і назву бази. Узагальнений формат такого рядка має вигляд: postgres://username:password@hostname:port/dbname

На прикладному рівні інтеграція реалізується через SQLAlchemy у Flask-застосунку. Джерелом конфігурації виступає змінна середовища DATABASE_URL, що дозволяє не зберігати чутливі дані безпосередньо в кодовій базі та спрощує перенесення між середовищами (локальне, тестове, production). У конфігурації застосунку використовується читання значення через os.environ.get(...) і передача цього рядка до драйвера підключення.

Перед запуском сервісу змінну потрібно явно визначити в термінальному середовищі, наприклад через команду export DATABASE_URL=.... Після цього серверна частина може встановити з’єднання з PostgreSQL і виконувати штатні операції читання та запису даних у межах транзакційної моделі.

На рисунку 3.10 показано приклад практичного налаштування параметра DATABASE_URL та ініціалізації підключення до PostgreSQL (Railway) у серверній частині Flask-застосунку.

```
# --- PostgreSQL Railway config ---
DATABASE_URL = "postgresql://postgres:tebiSVutOuZepNyjQqGWFzGUUUQeYfZL@yamanote.proxy.rlwy.net:23898/railway"

def get_db_connection():
    return psycopg2.connect(DATABASE_URL, sslmode='require')
```

Рисунок 3.10 – Налаштування DATABASE_URL і ініціалізація підключення Flask-застосунку до PostgreSQL (Railway)

Для первинної ініціалізації структури бази даних використано скрипти, наведені в Додатку А. У моделі збереження визначено дві базові сутності: User та Item. Сутність User містить атрибути облікового запису (ім’я, e-mail, хеш пароля), а Item описує об’єкти інвентарю (назва, опис, посилання на власника). Така структура дозволяє реалізувати персоналізоване керування ресурсами в межах одного застосунку.

На рівні ORM ці сутності відображаються на таблиці PostgreSQL із логічним зв’язком типу “один-до-багатьох”: один користувач може мати багато інвентарних записів, тоді як кожен запис Item належить одному конкретному користувачеві через зовнішній ключ. Для створення таблиць у БД застосовується стандартний механізм SQLAlchemy:

***db = SQLAlchemy(app)***

***db.create_all()***

Після виконання ініціалізації таблиці автоматично формуються у PostgreSQL відповідно до описаних моделей, що забезпечує коректну основу для операцій додавання, перегляду, редагування та видалення даних у вебзастосунку.

На рисунку 3.11 представлено відображення створених таблиць у середовищі Railway після успішного застосування ORM-моделей.

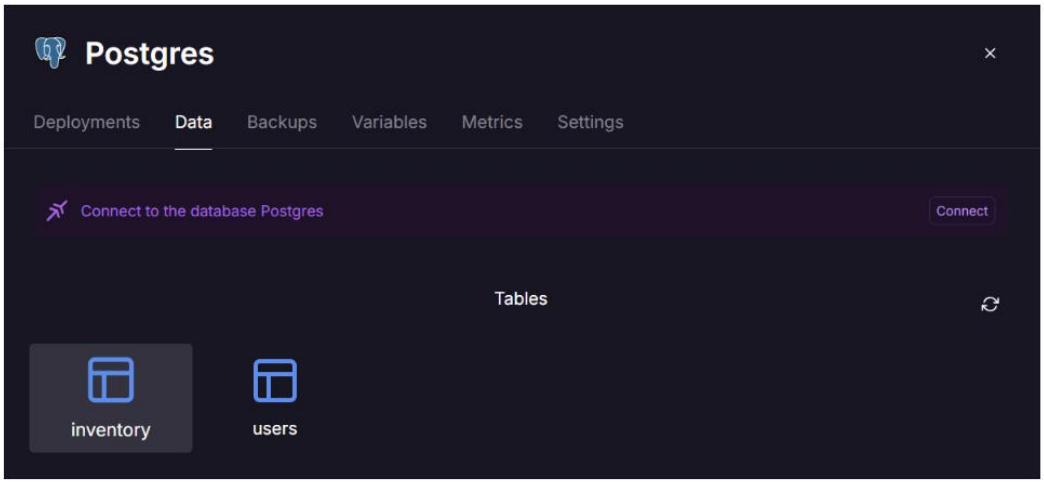


Рисунок 3.11 – Відображення таблиць users та inventory у PostgreSQL (Railway) після ініціалізації моделей SQLAlchemy

Після інтеграції вебзастосунку з PostgreSQL виконано серію базових функціональних перевірок, які підтвердили коректність обміну даними між прикладним і сховищним рівнями. У межах тестування послідовно перевірено реєстрацію нового користувача, автентифікацію, створення інвентарних записів і відображення збережених даних у вебінтерфейсі. Усі запити були опрацьовані без помилок, що свідчить про стабільне з'єднання з БД та правильну конфігурацію SQLAlchemy.

Практичний досвід показав, що використання Railway суттєво спрощує етап розгортання PostgreSQL, оскільки більшість інфраструктурних операцій автоматизовано на рівні платформи. Це скоротило час підготовки середовища, дало змогу зосередитись на прикладній логіці й підвищило відтворюваність конфігурації між тестовими і робочими сценаріями.

На рисунку 3.12 показано результат успішного запису та відображення користувацьких даних у PostgreSQL через інтегрований API-застосунок.

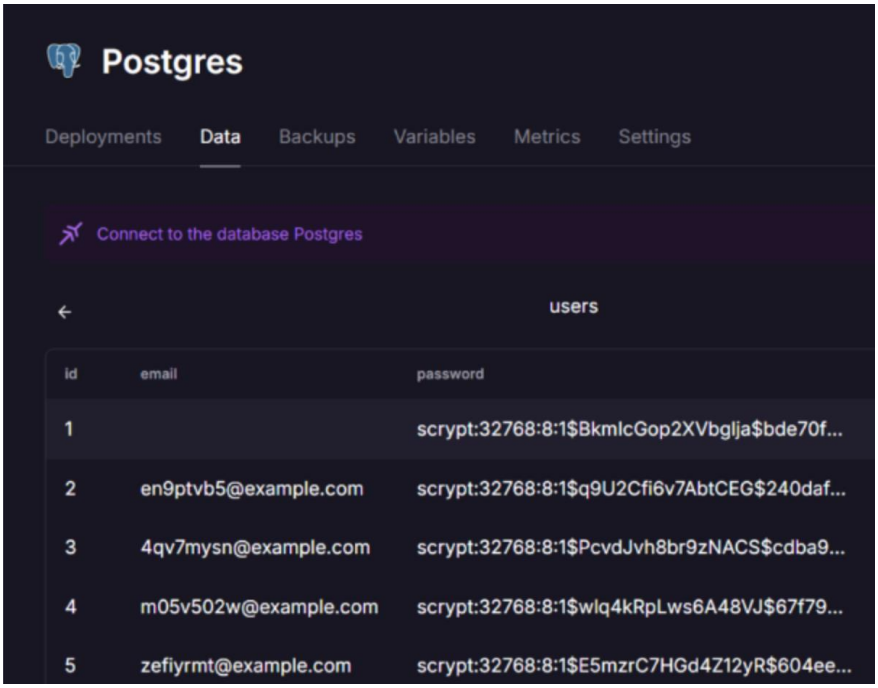


Рисунок 3.12 – Підтвердження успішної обробки API-запитів і запису даних у PostgreSQL (Railway)

Додатковою перевагою стала централізована робота зі змінними середовища, що підвищує керованість параметрів доступу та зменшує ризик помилок при зміні налаштувань підключення. Вбудовані механізми резервного копіювання знижують ризики втрати даних, а можливість масштабування ресурсів забезпечує стабільність БД під час пікового навантаження та стрес-тестування застосунку.

## 3.3 Візуалізація та аналітика (Grafana, дашборди, алерти)

Моніторинговий контур серверної інфраструктури було розширено за рахунок впровадження Grafana — платформи візуалізації часових рядів, що напряму інтегрується з Prometheus. Її використання в межах проєкту спрямоване на перетворення сирих метрик у наочні аналітичні панелі, які дозволяють оперативно відстежувати продуктивність і надійність серверного середовища.

Ключовою причиною вибору Grafana стала вбудована сумісність із Prometheus без додаткових проміжних компонентів. Система підтримує побудову інтерактивних графіків у реальному часі, гнучке фільтрування даних,

конструювання запитів PromQL та налаштування порогових сповіщень. Це суттєво спрощує виявлення відхилень у поведінці інфраструктури, зокрема при зростанні навантаження або деградації окремих сервісів.

Для спрощення адміністрування Grafana розгорнуто на тому самому EC2-вузлі, що й Prometheus (13.51.161.180), тобто в межах єдиного моніторингового хоста. Інсталяція виконувалась через офіційний репозиторій: додано джерело пакетів, імпортовано GPG-ключ, встановлено сервіс grafana, після чого активовано запуск служби та автозапуск через systemctl.

Завершальним кроком стало відкриття порту 3000 у правилах безпеки AWS для забезпечення доступу до вебінтерфейсу Grafana. Після цього система стала готовою до підключення джерела даних Prometheus і побудови дашбордів моніторингу [29].

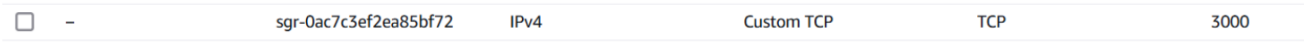


Рисунок 3.13 – Налаштування доступу до порту 3000 для вебінтерфейсу Grafana на моніторинговому EC2-сервері

Після завершення інсталяції Grafana та перевірки доступу за адресою http://13.51.161.180:3000 виконано підключення джерела метрик. Для цього здійснено вхід із початковими обліковими даними (admin/admin), після чого у розділі Connections → Data Sources додано нове джерело типу Prometheus. На етапі конфігурації в полі URL вказано адресу локального сервера метрик http://localhost:9090, оскільки Grafana і Prometheus розгорнуті на одному EC2-вузлі.

Після збереження параметрів виконано валідацію з’єднання через функцію Save & test. Успішний результат перевірки підтвердив, що Grafana коректно отримує метрики з Prometheus і готова до побудови дашбордів, графіків та алертів на основі запитів PromQL.

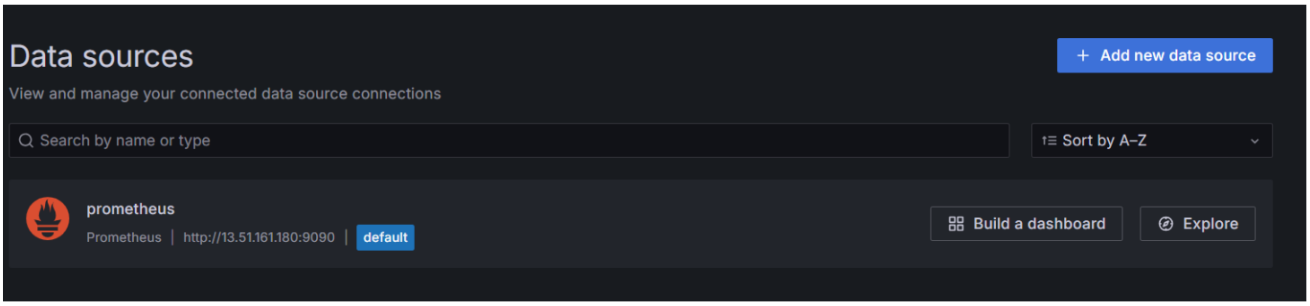


Рисунок 3.14 – Налаштування підключення Prometheus як джерела даних у Grafana

Після інтеграції джерела даних Prometheus було сформовано користувацький дашборд Grafana для оперативного контролю стану сервера застосунку. Структура дашборда охоплює ключові системні індикатори, які найбільш точно відображають рівень навантаження та стабільність інфраструктури під час роботи сервісу [29, 30].

До складу панелей включено:

- ***завантаження CPU***
  100 - (avg by(instance) (irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
- ***використання оперативної пам'яті***
  (node_memory_MemTotal_bytes - node_memory_MemAvailable_bytes) / node_memory_MemTotal_bytes * 100
- ***рівень заповнення дискового простору***
  node_filesystem_avail_bytes{fstype!="tmpfs"} /
  node_filesystem_size_bytes{fstype!="tmpfs"} * 100
- ***мережевий трафік (приймання/передача)***
  rate(node_network_receive_bytes_total[5m])
  rate(node_network_transmit_bytes_total[5m])

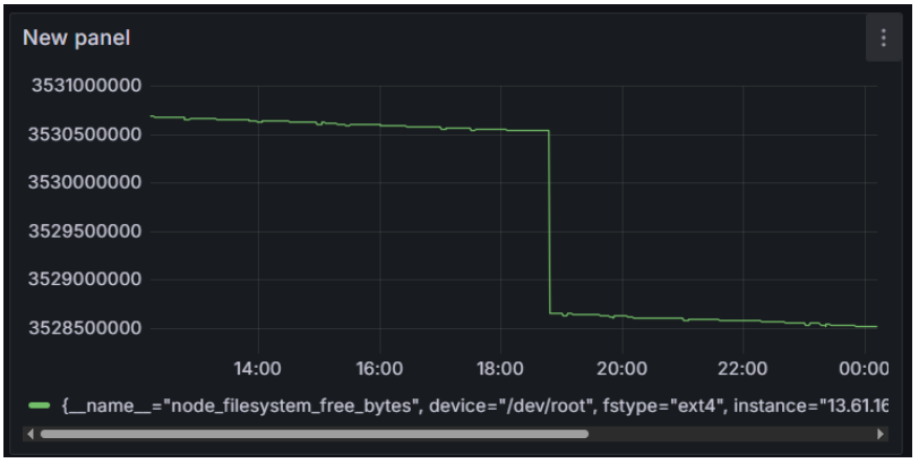


Рисунок 3.15 – Користувацький дашборд Grafana з візуалізацією дискового навантаження сервера застосунку

Такий набір метрик дозволяє одночасно контролювати обчислювальні, пам’яттєві, дискові та мережеві параметри, що є достатнім для первинної діагностики вузла і виявлення аномальної поведінки в роботі серверного середовища.

# Висновок до розділу 3

У третьому розділі реалізовано практичну частину проєкту: виконано розгортання вебзастосунку в хмарному середовищі AWS, налаштовано серверну інфраструктуру та забезпечено її працездатність у режимі реального доступу. Підтверджено коректність базового функціоналу застосунку, зокрема автентифікації користувачів і операцій з інвентарними даними, що створило прикладну основу для подальшого моніторингу.

Окремим результатом розділу стало впровадження підсистеми збору та обробки даних: інтегровано PostgreSQL у Railway, налаштовано підключення через змінні середовища та ORM SQLAlchemy, ініціалізовано структуру таблиць і перевірено повний цикл операцій читання/запису. Отримані результати підтвердили стабільність взаємодії між прикладним рівнем і базою даних, а також доцільність використання керованого хмарного сервісу для підвищення надійності та спрощення адміністрування.

Також у розділі побудовано повноцінний моніторинговий контур на основі Prometheus, Node Exporter і Grafana. Реалізовано збір системних метрик із серверу застосунку, налаштовано візуалізацію ключових показників (CPU, RAM, диск, мережа) та перевірено доступність інструментів спостереження в реальному часі. Таким чином, сформовано працездатну архітектуру, що забезпечує контроль продуктивності інфраструктури, своєчасне виявлення відхилень і технічну готовність до подальшого масштабування системи.

# ВИСНОВКИ

У кваліфікаційній роботі розглянуто підходи до побудови системи моніторингу серверної інфраструктури, орієнтованої на підвищення продуктивності, надійності та керованості вебзастосунків у хмарному середовищі. Обґрунтовано актуальність комплексного контролю стану серверів в умовах зростання навантаження, динамічності ресурсів і потреби в оперативному виявленні інцидентів.

У теоретичній частині проаналізовано сучасні інструменти та практики моніторингу, зокрема моделі збору телеметрії, підходи до централізованого зберігання даних і механізми візуалізації метрик. Виокремлено ключові технічні вимоги до такої системи: масштабованість, безперервність збору показників, надійність зберігання, зручність інтерпретації даних і можливість швидкого реагування на відхилення.

У практичній частині розроблено архітектуру розгортання вебзастосунку в AWS EC2 із відокремленням прикладного та моніторингового контурів. Реалізовано серверну частину на Flask, забезпечено працездатність функцій реєстрації, автентифікації та керування інвентарними записами. Такий підхід дозволив сформувати реальне експлуатаційне середовище для подальшої перевірки інструментів спостережуваності.

Окремо розроблено підсистему збереження та обробки даних на базі PostgreSQL (Railway), інтегровану із застосунком через SQLAlchemy та змінні середовища. Виконано ініціалізацію структури таблиць, перевірено коректність транзакційної взаємодії, а також підтверджено стабільність операцій читання/запису під час тестових сценаріїв. Це забезпечило надійну основу для прикладної логіки та подальшого масштабування.

У межах моніторингового напряму розроблено систему збору метрик за допомогою Prometheus і Node Exporter, а також інтерактивну візуалізацію в Grafana. Налаштовано збір і відображення ключових параметрів (CPU, оперативна пам’ять, дискова підсистема, мережевий трафік), що дало змогу здійснювати

оперативний контроль стану інфраструктури й своєчасно фіксувати потенційні відхилення.

Поставлену мету роботи досягнуто: розглянуто теоретичні засади, проаналізовано інструментальні рішення, розроблено практичну архітектуру та виокремлено ефективні підходи до організації моніторингу в хмарному середовищі. Отримані результати можуть бути використані як готовий приклад впровадження системи спостережуваності для вебпроєктів і як база для подальшого розвитку – зокрема в напрямі автоматизованих алертів, прогнозної аналітики та розширення сценаріїв відмовостійкості.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Turnbull J. The Prometheus Monitoring System: Monitoring and Alerting for Containerized and Distributed Systems. – Portland : Turnbull Press, 2018. – 320 p.
2. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. – Sebastopol : O'Reilly Media, 2017. – 616 p.
3. Burns B., Beda J., Hightower K. Kubernetes: Up and Running. – 3rd ed. – Sebastopol : O'Reilly Media, 2022. – 388 p.
4. Newman S. Building Microservices: Designing Fine-Grained Systems. – 2nd ed. – Sebastopol : O'Reilly Media, 2021. – 617 p.
5. Narkhede N., Shapira G., Palino T. Kafka: The Definitive Guide. – 2nd ed. – Sebastopol : O'Reilly Media, 2021. – 366 p.
6. Grafana Labs. Grafana Documentation [Електронний ресурс]. – Режим доступу: https://grafana.com/docs/ (дата звернення: 21.04.2026).
7. Prometheus Authors. Prometheus Documentation [Електронний ресурс]. – Режим доступу: https://prometheus.io/docs/ (дата звернення: 21.04.2026).
8. Amazon Web Services. Amazon CloudWatch Documentation [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/cloudwatch/ (дата звернення: 21.04.2026).
9. PostgreSQL Global Development Group. PostgreSQL Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/ (дата звернення: 21.04.2026).
10. Docker Inc. Docker Documentation [Електронний ресурс]. – Режим доступу: https://docs.docker.com/ (дата звернення: 21.04.2026).
11. Kubernetes Authors. Kubernetes Documentation [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/ (дата звернення: 21.04.2026).
12. Red Hat. OpenShift Monitoring Documentation [Електронний ресурс]. – Режим доступу: https://docs.openshift.com/ (дата звернення: 21.04.2026).

13. Linux Foundation. OpenTelemetry Documentation [Електронний ресурс]. – Режим доступу: https://opentelemetry.io/docs/ (дата звернення: 21.04.2026).

14. Turnbull J. Monitoring with Prometheus. – 2nd ed. – Berlin : Linux New Media, 2021. – 285 p.

15. Zabbix LLC. Zabbix Documentation [Електронний ресурс]. – Режим доступу: https://www.zabbix.com/documentation/current/en/manual (дата звернення: 21.04.2026).

16. Nagios Enterprises. Nagios Documentation [Електронний ресурс]. – Режим доступу: https://assets.nagios.com/downloads/nagioscore/docs/ (дата звернення: 21.04.2026).

17. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. – Boston : Addison-Wesley Professional, 2015. – 512 p.

18. Prometheus Authors. Getting started [Електронний ресурс]. – 2026. – Режим доступу: https://prometheus.io/docs/prometheus/latest/getting_started/ (дата звернення: 09.05.2026).

19. Prometheus Authors. Configuration [Електронний ресурс]. – 2026. – Режим доступу: https://prometheus.io/docs/prometheus/latest/configuration/configuration/ (дата звернення: 09.05.2026).

20. Prometheus Authors. Querying basics (PromQL) [Електронний ресурс]. – 2026. – Режим доступу: https://prometheus.io/docs/prometheus/latest/querying/basics/ (дата звернення: 09.05.2026).

21. Prometheus Authors. Alertmanager documentation [Електронний ресурс]. – 2026. – Режим доступу: https://prometheus.io/docs/alerting/latest/alertmanager/ (дата звернення: 09.05.2026).

22. Grafana Labs. Prometheus data source [Електронний ресурс]. – 2026. – Режим доступу: https://grafana.com/docs/grafana/latest/datasources/prometheus/ (дата звернення: 09.05.2026).

23. Grafana Labs. Dashboards [Електронний ресурс]. – 2026. – Режим доступу: https://grafana.com/docs/grafana/latest/dashboards/ (дата звернення: 09.05.2026).

24. Railway. PostgreSQL [Електронний ресурс]. – 2026. – Режим доступу: https://docs.railway.com/databases/postgresql (дата звернення: 09.05.2026). Railway.

25. Using Variables [Електронний ресурс]. – 2026. – Режим доступу: https://docs.railway.com/variables (дата звернення: 09.05.2026). Railway. Databases [Електронний ресурс]. – 2026. – Режим доступу: https://docs.railway.com/databases (дата звернення: 09.05.2026).

26. Amazon Web Services. Get started with Amazon EC2 [Електронний ресурс]. – 2026. –

27. Режим доступу: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html (дата звернення: 09.05.2026).

28. Amazon Web Services. Launch an EC2 instance using the launch instance wizard [Електронний ресурс]. – 2026. – Режим доступу: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-launch-instance-wizard.html (дата звернення: 09.05.2026).

29. Amazon Web Services. Security groups for your VPC [Електронний ресурс]. – 2026. – Режим доступу: https://docs.aws.amazon.com/vpc/latest/userguide/vpc-security-groups.html (дата звернення: 09.05.2026).

30. PostgreSQL Global Development Group. PostgreSQL Documentation [Електронний ресурс]. – 2026. – Режим доступу: https://www.postgresql.org/docs/ (дата звернення: 09.05.2026).

31. Bayer M. et al. SQLAlchemy 2.0 Documentation [Електронний ресурс]. – 2026. – Режим доступу: https://docs.sqlalchemy.org/20/ (дата звернення: 09.05.2026).

32. Pallets. Flask Documentation (3.1.x) [Електронний ресурс]. – 2025. – Режим доступу: https://flask.palletsprojects.com/en/stable/ (дата звернення: 09.05.2026).

**ДЕМОНСТРАЦІЙНИЙ МАТЕРІАЛ (Презентація)**

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

**КВАЛІФІКАЦІЙНА РОБОТА**

# «Інформаційно-аналітична система моніторингу хмарної інфраструктури»

на здобуття освітнього ступеня бакалавра
зі спеціальності 124 – Системний аналіз
освітньо-професійної програми Системний аналіз

**Виконав:** здобувач вищої освіти гр. САД-41
Олексій ГОРБУНОВ

**Керівник:** Михайло Кузьміч
доктор філософії, доцент кафедри Інформаціних Систем та Технологій

**Київ - 2026**

# МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

## Мета роботи

Розробка інформаційно-аналітичної системи моніторингу хмарної інфраструктури, яка забезпечує збір метрик, аналіз стану системи, виявлення аномалій та зручну візуалізацію даних для адміністратора.

## Об'єкт дослідження

Процеси моніторингу та аналізу стану хмарної інфраструктури в інформаційних системах.

## Предмет дослідження

Методи та засоби побудови інформаційно-аналітичних систем для збору, обробки та візуалізації даних про стан хмарних ресурсів.

# НАУКОВІ ЗАВДАННЯ ТА МЕТОДИ ДОСЛІДЖЕННЯ

**Наукові завдання:**

- Дослідження тенденцій розвитку хмарних технологій та інфраструктур
- Огляд методів моніторингу: метрики, логи, трейси, системи алертингу
- Порівняльний аналіз сучасних інструментів (Prometheus, Grafana, Zabbix, Nagios, CloudWatch)
- Формування вимог та проєктування архітектури системи
- Реалізація механізмів збору метрик і інтеграція PostgreSQL
- Розробка дашбордів Grafana та налаштування алертів
- Тестування та оцінка ефективності системи

**Методи дослідження:**

- Аналіз сучасних підходів до хмарних інфраструктур та моніторингу
- Дослідження існуючих систем моніторингу
- Проєктування архітектури інформаційно-аналітичної системи
- Розробка механізмів збору та обробки метрик
- Реалізація алгоритмів аналізу даних та виявлення аномалій
- Створення інтерфейсу для візуалізації результатів

# МОДЕЛІ ТА ТИПИ ХМАРНИХ ІНФРАСТРУКТУР

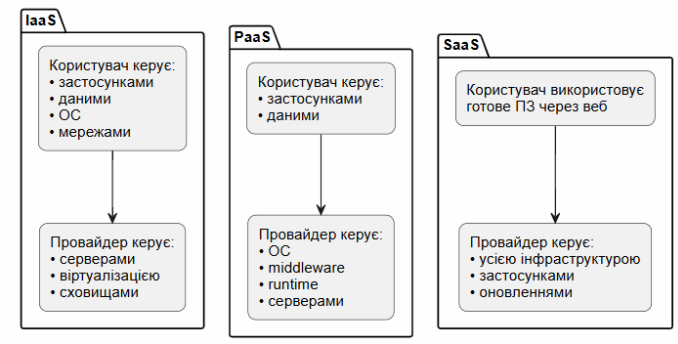


Рисунок 1 – Моделі хмарних обчислень (IaaS, PaaS, SaaS) та розподіл відповідальності

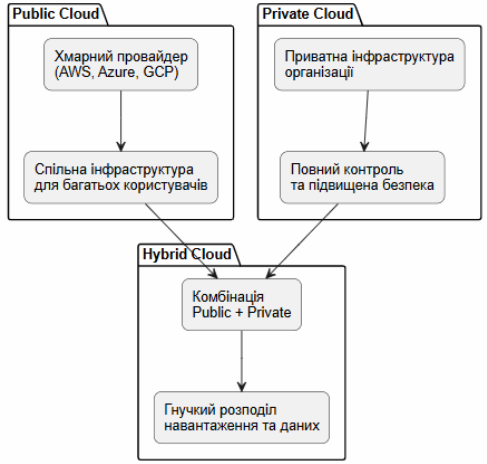


Рисунок 2 – Типи хмарних інфраструктур

Таблиця 1 – Порівняння типів хмарної інфраструктури

| Тип | Переваги | Недоліки | Приклади |
|---|---|---|---|
| Public Cloud | Масштабованість, доступність, низькі витрати | Обмежений контроль, питання приватності | AWS, Azure, GCP |
| Private Cloud | Безпека, повний контроль, ізоляція | Висока вартість, складність адміністрування | OpenStack, VMware |
| Hybrid Cloud | Гнучкість, оптимізація витрат, відмовостійкість | Складна інтеграція різних середовищ | AWS Outposts, Azure Arc |

# ПОРІВНЯЛЬНИЙ АНАЛІЗ СИСТЕМ МОНІТОРИНГУ ХМАРНОЇ ІНФРАСТРУКТУРИ

Таблиця 2 – Порівняльна характеристика сучасних систем моніторингу

| Критерій | Prometheus | Grafana | Zabbix | Nagios | CloudWatch |
|---|---|---|---|---|---|
| Тип | Збір метрик | Візуалізація | Комплексний | Моніторинг серверів | AWS-сервіс |
| Архітектура | Pull-модель | Зовнішні джерела | Agent-based | Plugin-based | Хмарна |
| Контейнери | Висока | Висока | Середня | Обмежена | Висока |
| Kubernetes | Повна | Повна | Часткова | Обмежена | AWS EKS |
| Візуалізація | Базова | Розширена | Вбудована | Мінімальна | AWS Dashboard |
| Масштабованість | Висока | Висока | Середня | Низька | Висока |
| Ліцензія | Open-source | Open-source | Open-source | Open-source | Комерційний |
| Вартість | Безкоштовно | Безкоштовно | Безкоштовно | Безкоштовно | За використання |

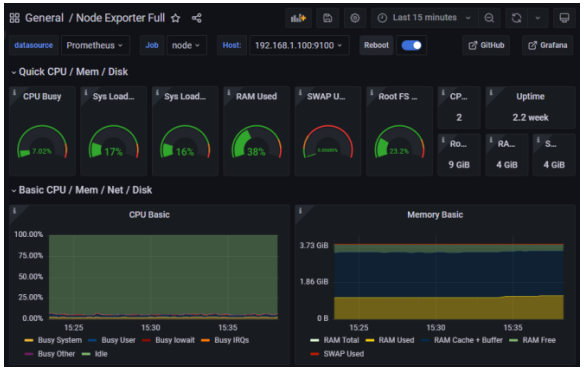


Рисунок 3 – Grafana: Node Exporter Full дашборд

Рисунок 4 – Grafana: Website Performance дашборд

# АРХІТЕКТУРА ІНФОРМАЦІЙНО-АНАЛІТИЧНОЇ СИСТЕМИ МОНІТОРИНГУ

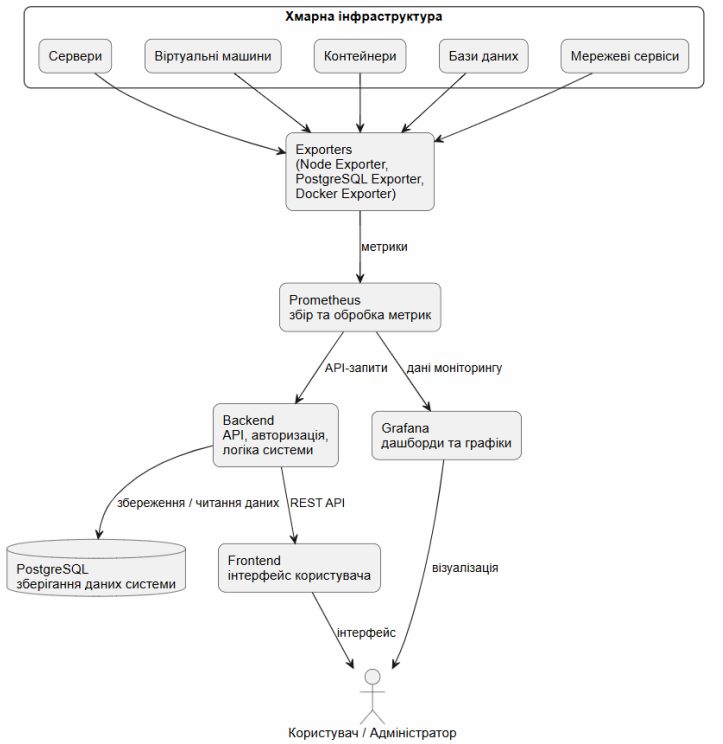


Рисунок 5 – Архітектурна схема системи моніторингу (Prometheus + Grafana + PostgreSQL)

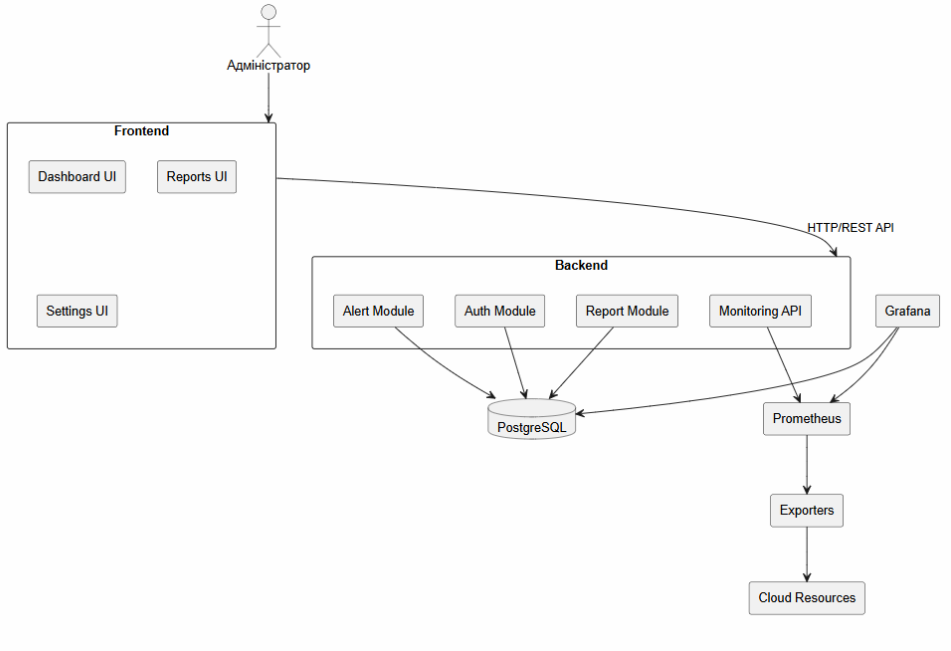


Рисунок 6 – UML-діаграма компонентів системи моніторингу

# ФУНКЦІОНАЛЬНІ ТА НЕФУНКЦІОНАЛЬНІ ВИМОГИ ДО СИСТЕМИ

Таблиця 3 – Функціональні вимоги до системи моніторингу

| Вимога | Опис |
|---|---|
| Збір метрик у реальному часі | Автоматичне отримання даних CPU, RAM, Disk, Network через Prometheus Exporters |
| Зберігання даних | Збереження метрик у PostgreSQL (Railway) та TSDB Prometheus |
| Виявлення аномалій | Генерація алертів при перевищенні порогових значень |
| Візуалізація | Інтерактивні дашборди Grafana з графіками та метриками |
| Авторизація | Реєстрація та автентифікація користувачів системи |
| Звітність | Формування звітів про стан інфраструктури |

Таблиця 4 – Нефункціональні вимоги до системи моніторингу

| Вимога | Опис |
|---|---|
| Продуктивність | Збір метрик кожні 15–60 секунд без значного впливу на сервер |
| Масштабованість | Підтримка підключення нових серверів без зміни архітектури |
| Надійність | Uptime системи не менше 99.5%; резервування даних |
| Безпека | Шифрування з'єднань, захист паролів (scrypt), контроль доступу |
| Зручність використання | Інтуїтивний вебінтерфейс для перегляду метрик і дашбордів |

# ER-МОДЕЛЬ БАЗИ ДАНИХ ТА ЛОГІКА ОБРОБКИ ДАНИХ

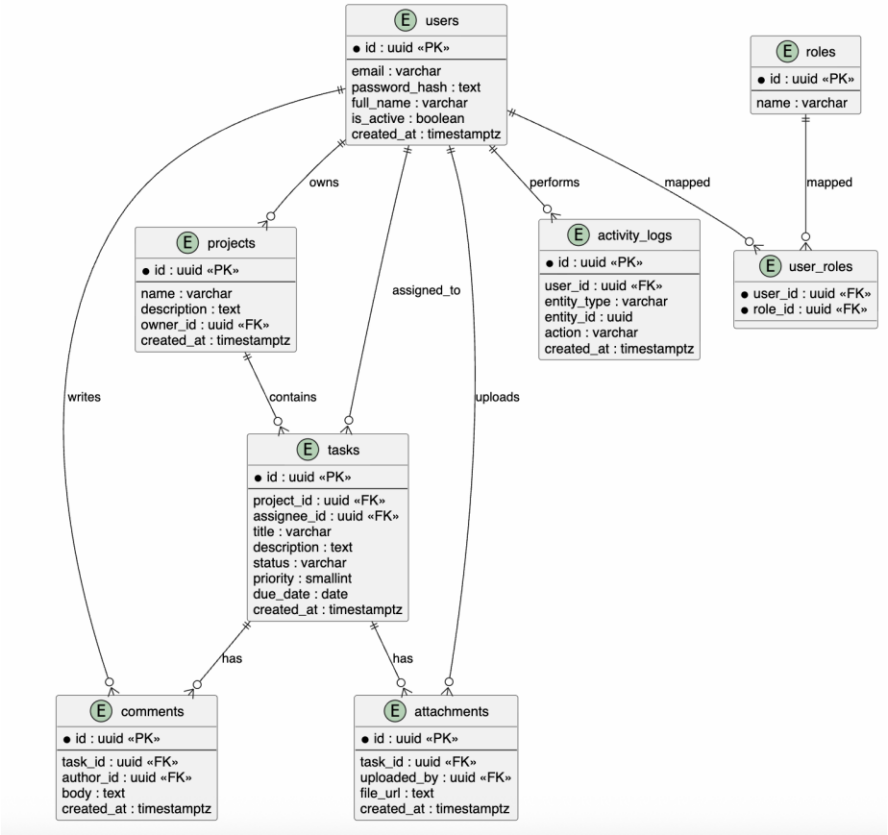


Рисунок 7 – ER-модель бази даних (концептуальний рівень)

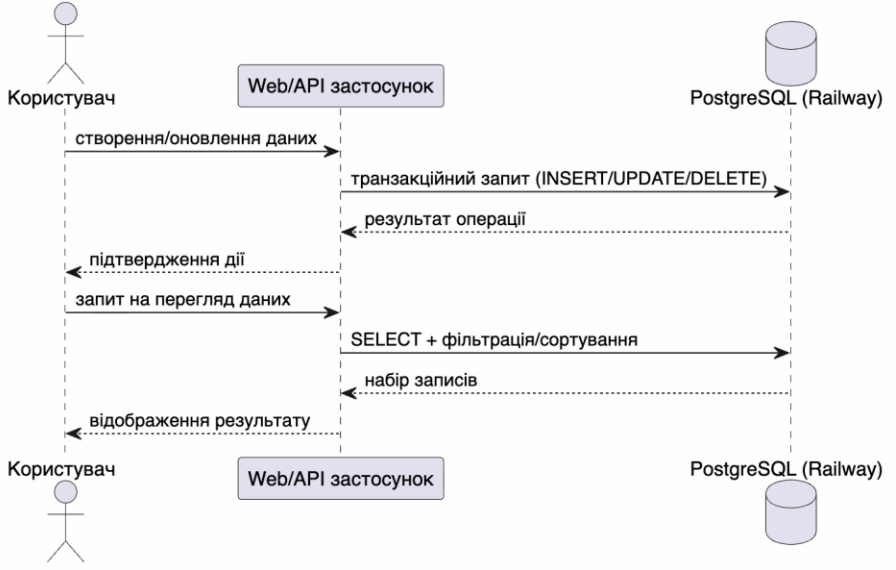


Рисунок 8 – UML-діаграма послідовності взаємодії з PostgreSQL (Railway)

# РЕАЛІЗАЦІЯ: ВЕБЗАСТОСУНОК ТА РОЗГОРТАННЯ НА AWS EC2

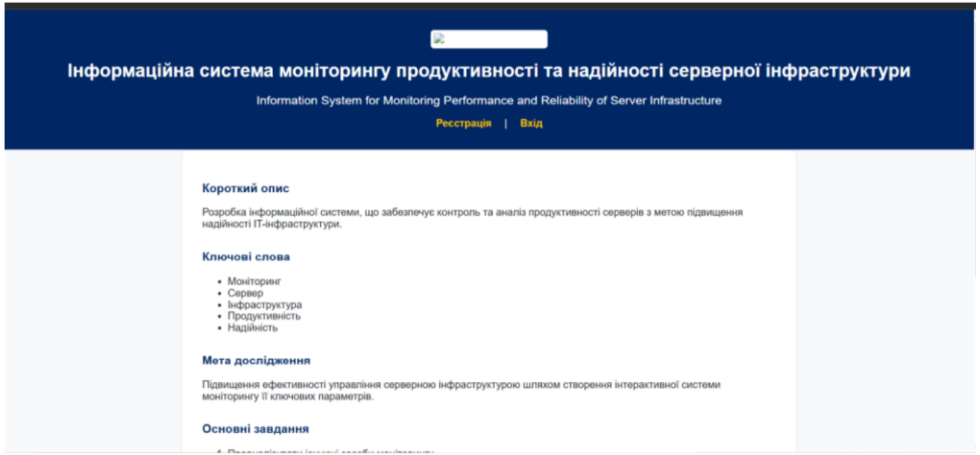


Рисунок 9 – Головна сторінка вебзастосунку системи моніторингу (Flask, AWS EC2)

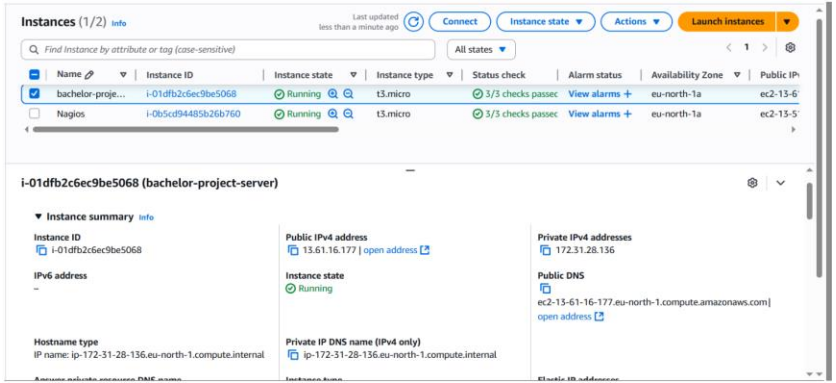


Рисунок 10 – Активні EC2-інстанси AWS (bachelor-project-server + Prometheus-сервер)

# РЕАЛІЗАЦІЯ: PROMETHEUS, NODE EXPORTER ТА POSTGRESQL RAILWAY

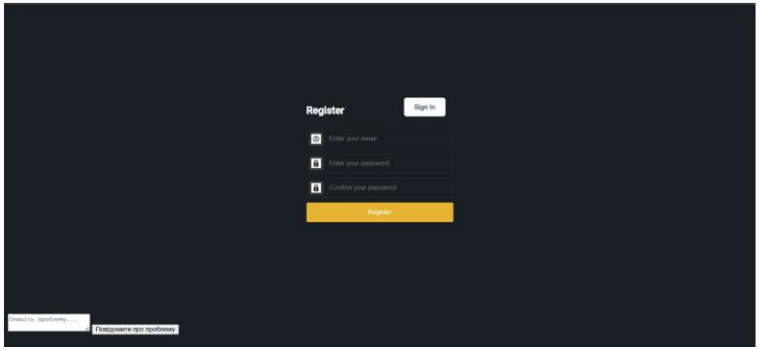


Рисунок 11 – Форма реєстрації вебзастосунку на EC2

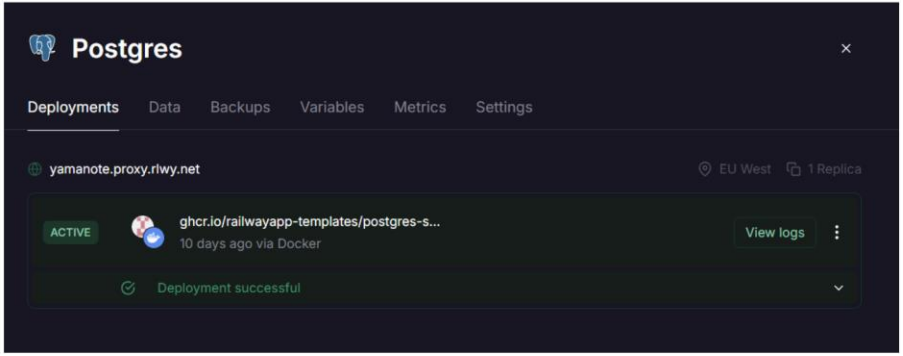


Рисунок 12 – PostgreSQL-сервіс у Railway (Deployment successful)

```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 3.2682e-05
go_gc_duration_seconds{quantile="0.25"} 3.7978e-05
go_gc_duration_seconds{quantile="0.5"} 4.1189e-05
go_gc_duration_seconds{quantile="0.75"} 4.5835e-05
go_gc_duration_seconds{quantile="1"} 0.000184532
go_gc_duration_seconds_sum 2.118012175
go_gc_duration_seconds_count 48730
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 9
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.22.2"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 2.089064e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 9.1916954552e+10
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 2.075466e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 1.173216692e+09
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 3.227376e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 2.089064e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 4.087808e+06
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 3.825664e+06
# HELP go_memstats_heap_objects Number of allocated objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 12138
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.
# TYPE go_memstats_heap_released_bytes gauge
go_memstats_heap_released_bytes 3.145728e+06
```

Рисунок 14 – Системні метрики Node Exporter на endpoint /metrics

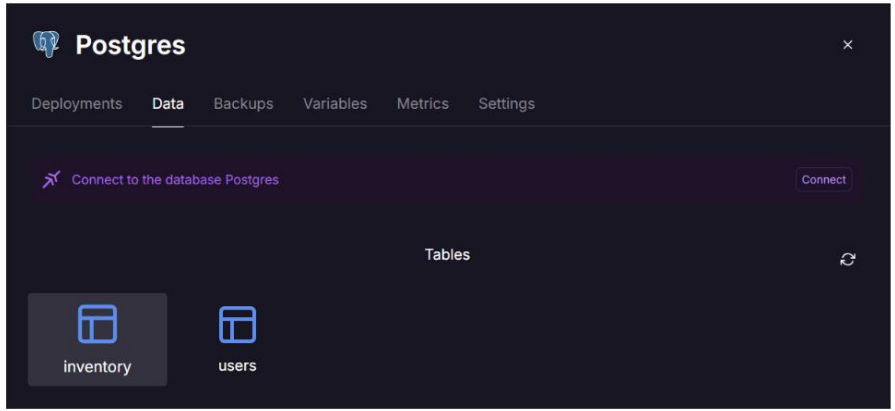


Рисунок 13 – Таблиці users та inventory у Railway PostgreSQL

# ВІЗУАЛІЗАЦІЯ ТА АНАЛІТИКА: GRAFANA ДАШБОРДИ

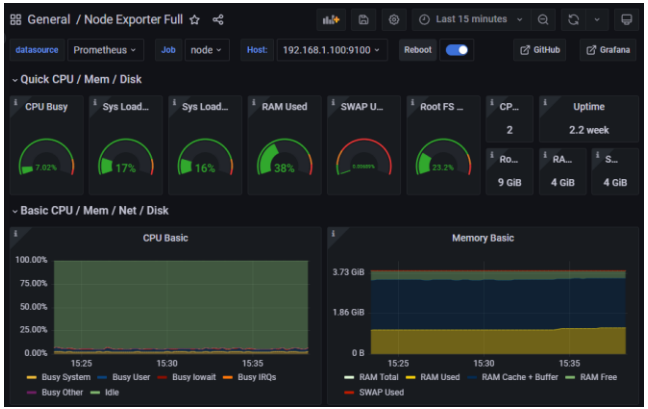


Рисунок 15 – Дашборд Node Exporter Full: CPU, RAM, Disk, Uptime

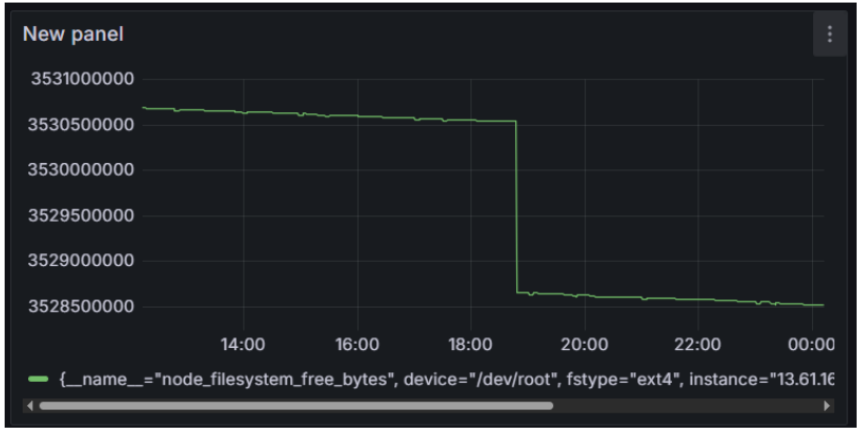


Рисунок 16 – Користувацький дашборд Grafana: дискове навантаження


Рисунок 17 – Grafana: Website Performance - Memory/CPU, server requests, page load time

# ВИСНОВКИ

1. Досліджено теоретичні засади хмарних інфраструктур: розглянуто моделі IaaS, PaaS, SaaS, типи хмар (public, private, hybrid) та основні підходи до моніторингу (метрики, логи, трейси).

2. Проведено порівняльний аналіз систем моніторингу (Prometheus, Grafana, Zabbix, Nagios, CloudWatch): обрано стек Prometheus + Grafana як найбільш ефективне open-source рішення.

3. Сформовано функціональні та нефункціональні вимоги та розроблено архітектуру системи з модулями збору метрик, зберігання, аналізу та візуалізації.

4. Спроєктовано ER-модель бази даних із таблицями users, inventory, projects, tasks, activity_logs; інтегровано PostgreSQL через Railway.

5. Реалізовано підсистему збору метрик: Flask-застосунок розгорнуто на AWS EC2, підключено Node Exporter, налаштовано Prometheus для регулярного scraping.

6. Розроблено дашборди Grafana (Node Exporter Full, Website Performance, дискове навантаження) та налаштовано систему алертингу.

7. Поставлену мету досягнуто: система забезпечує ефективний моніторинг хмарної інфраструктури та є основою для розширення — автоматизованих алертів та прогнозної аналітики.

**Дякую за увагу!**