Звіт з лабораторної роботи
По предмету «Інформаційні системи та технології»

Тема: Система моніторингу роботи бази даних з використанням PostgreSQL, Prometheus, Grafana та серверної авторизації на Python

1. Загальна ідея та архітектура

Мета проєкту:

Моніторити роботу СУБД через Prometheus → зберігати метрики → візуалізувати їх у Grafana. Доступ до всього (мінімум — до Grafana/Prometheus) мають тільки авторизовані користувачі. Усі компоненти запускаються в Docker.

Сервіси:

1. `db` – СУБД (наприклад, PostgreSQL).
2. `db-loader` – Python-скрипт, який генерує навантаження на БД (insert/update/select) і, за бажання, віддає свої метрики.
3. `postgres_exporter` – експортер метрик БД для Prometheus.
4. `prometheus` – збирач і БД часових рядів.
5. `grafana` – візуалізація метрик.
6. `auth-server` – сервер авторизації (Python + SQLite, наприклад Flask/FastAPI).
7. (Опційно) `rabbitmq` або `kafka` + сервіс-споживач для метрик/подій.
8. `reverse-proxy` (наприклад, Nginx) – щоб закрити доступ до Prometheus/Grafana через авторизацію.

Усі сервіси піднімаються через `docker-compose`.

2. Що це за система

Проєкт db-monitoring-project — це мікросервісна система моніторингу роботи бази даних:

- **PostgreSQL** зберігає тестові дані.
- **Python-сервіс db-loader** постійно:
  - генерує та записує випадкові значення в таблицю test_data;
  - читає дані з БД;
  - експортує власні метрики у форматі Prometheus (наприклад db_operations_total).
- **Prometheus**:
  - періодично опитує db-loader і postgres_exporter;
  - зберігає всі метрики у власне time-series сховище.
- **Grafana**:
  - підключається до Prometheus як до джерела даних;
  - будує графіки та дашборди по метриках.
- **Auth-server (Python + SQLite)**:
  - простий сервер авторизації з формою логіна;
  - видає cookie після успішного входу;

    - використовується Nginx для перевірки доступу.
- **Nginx**:
    - є єдиною точкою входу (http://localhost);
    - прокидує /auth/ на сервер авторизації;
    - пропускає до /grafana/ тільки авторизованих користувачів.
- Усе це запущено в **Docker-контейнерах** через docker-compose.

3. Де що реалізовано (по пунктах завдання)

3.1 Запуск СУБД у Docker-контейнері

Файли/місце реалізації:

docker-compose.yml, сервіс **db**:

```
db:
  image: postgres:15
  container_name: db
  environment:
    POSTGRES_USER: appuser
    POSTGRES_PASSWORD: apppass
    POSTGRES_DB: appdb
  ports:
    - "5432:5432"
  volumes:
    - db_data:/var/lib/postgresql/data
```

Перевірка:

docker compose ps

У списку контейнерів має бути:

db postgres:15 Up 0.0.0.0:5432->5432/tcp

Підключення до БД зсередини контейнера:

docker exec -it db psql -U appuser -d appdb

3.2 Генерація актуалізації даних у БД (Python 3)

Файли/місце реалізації:

- Каталог ./db-loader
    - loader.py (або app.py) — основний Python-скрипт, який:
        - робить підключення до Postgres;
        - створює таблицю test_data (якщо її немає);
        - у циклі вставляє випадкові значення та робить вибірки;
        - оновлює Prometheus-метрики.

Сервіс у docker-compose.yml:

```
db-loader:
 build: ./db-loader
 container_name: db-loader
 depends_on:
  - db
 networks:
  - monitor-net
```

Перевірка генерації даних:

Зайти в БД:

```
docker exec -it db psql -U appuser -d appdb
```

1. Подивитися таблиці: \dt

Має бути: public | test_data | table | appuser

2. Перевірити, що дані реально з'являються:
   SELECT * FROM test_data LIMIT 10;

   Або підрахувати кількість рядків:
   SELECT COUNT(*) FROM test_data;
3. Якщо лічильник росте — db-loader працює й постійно "актуалізує" базу.

3.3 Збір даних: використання Prometheus

Файли/місце реалізації:

Сервіс **prometheus** в docker-compose.yml:

```
prometheus:
 image: prom/prometheus
 container_name: prometheus
 volumes:
  - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
  - prometheus_data:/prometheus
 ports:
  - "9090:9090"
```

Конфіг ./prometheus/prometheus.yml (там описані scrape_configs, наприклад job db-loader та postgres_exporter).

Що збирається:

- системні метрики PostgreSQL через postgres_exporter;
- кастомні метрики сервісу db-loader, зокрема:

- db_operations_total{type="insert"};
- db_operations_total{type="select"}.

Перевірка Prometheus:

1. Відкрити веб-інтерфейс: http://localhost:9090

У вкладці **Query** ввести: db_operations_total - метрика яка є **лічильником (counter)** і показує, **скільки разів виконувалися операції з БД**

2. і натиснути **Execute**.
   - db_operations_total{job="db-loader", type="insert"}
   - db_operations_total{job="db-loader", type="select"}

3. Щоб побачити **зростання / динаміку**, перейти у вкладку **Graph** і виконати: rate(db_operations_total[1m]) – так видно швидкість зростання лічильника операцій за хвилину.

3.4 Збереження даних у Time Series Storage Prometheus

Реалізація:

Це робиться самим Prometheus, шлях до сховища вказано в docker-compose.yml:

```
volumes:
 - prometheus_data:/prometheus
command:
 - --config.file=/etc/prometheus/prometheus.yml
 - --storage.tsdb.path=/prometheus
```

Як переконатися, що метрики зберігаються:

1. Запусти систему, почекай кілька хвилин.
2. У Prometheus на http://localhost:9090:

вибераємо будь-яку метрику, наприклад:
db_operations_total{type="insert"}

- У верхньому меню ставимо **Evaluation time** більш ранній час (наприклад, на 10 хвилин назад) і подивись графік — будуть історичні значення.

Зупини і знову запусти Prometheus через Docker:
docker compose restart prometheus

3. Метрики не зникнуть, бо вони лежать у volume prometheus_data.

3.5 Відображення даних: Grafana

Файли/місце реалізації:

Сервіс **grafana** у docker-compose.yml:

```
grafana:
```

image: grafana/grafana
container_name: grafana
depends_on:
- prometheus
environment:
- GF_SECURITY_ADMIN_USER=admin
- GF_SECURITY_ADMIN_PASSWORD=admin
- GF_SERVER_ROOT_URL=/grafana/
- GF_SERVER_SERVE_FROM_SUB_PATH=true
ports:
- "3000:3000"
volumes:
- grafana_data:/var/lib/grafana
- ./grafana/provisioning:/etc/grafana/provisioning

-
- В ./grafana/provisioning/datasources/ — конфігурація data source типу **Prometheus**, який посилається на http://prometheus:9090.

Вхід в Grafana:

- через Nginx, після авторизації: http://localhost/grafana/
- логін/пароль Grafana за замовчуванням (з env):
  - **admin / admin**

Перевірка зв'язку з Prometheus:

1. В Grafana зайти в **Connections** → **Data sources** → **prometheus-1**.
2. Натиснути **Save & test** — має з'явитися повідомлення, що Prometheus успішно запитується.
3. Для візуалізації: Explore → обрати data source prometheus-1;

Ввести: db_operations_total - подивитися **Graph**.

3.6 Авторизація: сервер авторизації (Python + SQLite)

Файли/місце реалізації:

- Каталог ./auth-server:
  - app.py — Flask (або інший фреймворк), де є:

        - маршрут / або /auth/ — сторінка логіну (форма HTML);
        - маршрут /login (POST) — перевірка логіну/пароля;
        - маршрут /verify — API для Nginx auth_request (повертає 200/401);
        - маршрут /success — сторінка “Login successful…”.
    - auth.db — SQLite-база з користувачами (або хардкод в коді).

Сервіс **auth-server** в docker-compose.yml:

```
auth-server:
  build: ./auth-server
  container_name: auth-server
  ports:
    - "5000:5000"
```

Типовий логін (як ми використовували):

- **username:** admin
- **password:** admin

(якщо в auth-server/app.py зроблено інакше — підставити свої).

Перевірка авторизації:

1. В браузері: http://localhost/
   Nginx зробить редирект на /auth/. З’являється форма логіну.
2. Ввести логін/пароль і натиснути **Login**.
3. Після цього відкриється сторінка: http://localhost/auth/success
   текст: **“Login successful. You can now access /grafana/ and /prometheus/ via Nginx.”**
4. Тепер можна зайти на:
    - http://localhost/grafana/ — працює;
    - неавторизованого користувача туди не пустить (буде 401 або редирект на /auth/).

3.7 Контейнеризація: Docker

Головний файл:

- docker-compose.yml в корені проєкту.

Що описано:

- Сервіси: db (PostgreSQL); db-loader (Python-навантажувач); postgres_exporter; prometheus; grafana; auth-server; nginx
- Спільна мережа monitor-net.

- Волюми: db_data, prometheus_data, grafana_data.

Команди для запуску всієї системи:

із кореня проєкту

docker compose up -d

Перевірка, що всі сервіси встали:

docker compose ps

3.8 Обмін повідомленнями (RabbitMQ / Kafka)

У завданні є пункт:

> Допускається реалізація проекту без брокерів повідомлень…

У нашому варіанті **RabbitMQ / Kafka не використовуються**. Моніторинг роботи СУБД реалізований “програмними методами”:

- db-loader безпосередньо працює з БД через Python (psycopg2 / asyncpg тощо);
- Prometheus опитує db-loader та postgres_exporter по HTTP.

4. Повний сценарій запуску системи

1. **Переконатися, що Docker і Docker Compose працюють.**
2. Перейти в папку проєкту: cd A:\уник\4_курс\инф_сис\db-monitoring-project
3. Запустити всі сервіси: docker compose up -d
4. Перевірити статус: docker compose ps
5. Усі потрібні сервіси мають бути в статусі Up.
6. (За потреби) перезапустити Nginx / Grafana / auth-server після зміни конфігів: docker compose restart nginx grafana auth-server

5. Перевірка роботи системи крок за кроком

4.1. Перевірка БД та db-loader

docker exec -it db psql -U appuser -d appdb
Виведе - appdb=#

У psql:

\dt -- подивитися таблиці

```
(.venv) A:\уник\4_курс\инф_сис\db-monitoring-project>docker exec -it db psql -U appuser -d appdb

psql (15.15 (Debian 15.15-1.pgdg13+1))
Type "help" for help.

appdb=# \dt
          List of relations
 Schema |   Name    | Type  |  Owner
--------+-----------+-------+---------
 public | test_data | table | appuser
(1 row)

appdb=#
```

Наприклад: SELECT * FROM test_data LIMIT 10;
Щоб швидко подивитися:

- які дані взагалі є в таблиці
- які є поля (стовпці)
- які значення вносить твій **db-loader**
- чи робить програма вставку коректно
- чи зберігається час (created_at)
- чи не дублюються дані

Це найшвидший спосіб перевірити, що **генерація даних працює**.

```
appdb=# SELECT * FROM test_data LIMIT 10;
 id | value |         created_at
----+-------+----------------------------
  1 |   171 | 2025-11-22 19:08:57.089287
  2 |   450 | 2025-11-22 19:08:59.093552
  3 |    37 | 2025-11-22 19:09:01.096187
  4 |   995 | 2025-11-22 19:09:03.099239
  5 |   509 | 2025-11-22 19:09:05.101531
  6 |   411 | 2025-11-22 19:09:07.104116
  7 |   272 | 2025-11-22 19:09:09.119738
  8 |   238 | 2025-11-22 19:09:11.135415
  9 |   772 | 2025-11-22 19:09:15.121433
 10 |    10 | 2025-11-22 19:09:17.12434
(10 rows)
```

Порахувати кількість рядків: SELECT COUNT(*) FROM test_data;

Якщо COUNT(*) з часом росте — **запис відбувається**.

\q - вихід з psql

4.2. Перевірка, що метрики db-loader забирає Prometheus

1. Відкрити Prometheus: http://localhost:9090

В поле запиту ввести:

db_operations_total

2. Натиснути **Execute** → у таблиці будуть два рядки:

    - db_operations_total{type="insert", ...}
    - db_operations_total{type="select", ...}

Щоб побачити **динаміку росту**, перейти на вкладку **Graph** і виконати:
rate(db_operations_total[1m])
Ти одразу побачиш лінію:

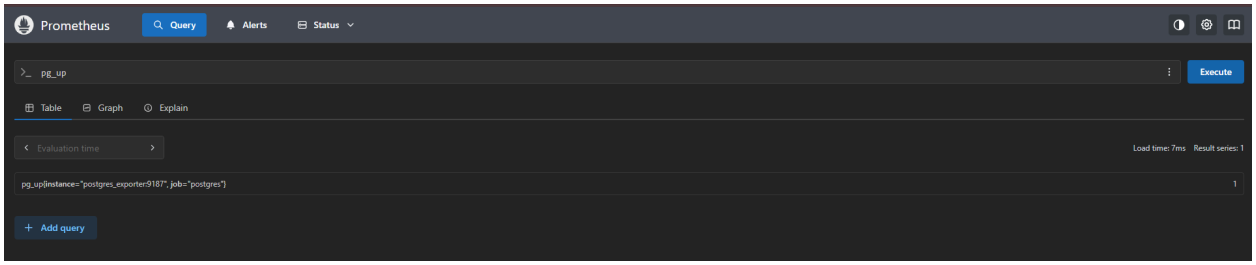
- яка зростає, коли db-loader робить вставки/вибірки,
- падає до нуля, якщо припиняється потік даних.

4.3. Перевірка PostgreSQL-експортера

У Prometheus (http://localhost:9090): **pg_up**

- Показує, чи PostgreSQL доступний для експортеру.
- Значення:
    - **1** → все працює

- **0** → проблеми з доступом або Prometheus не бачить exporter



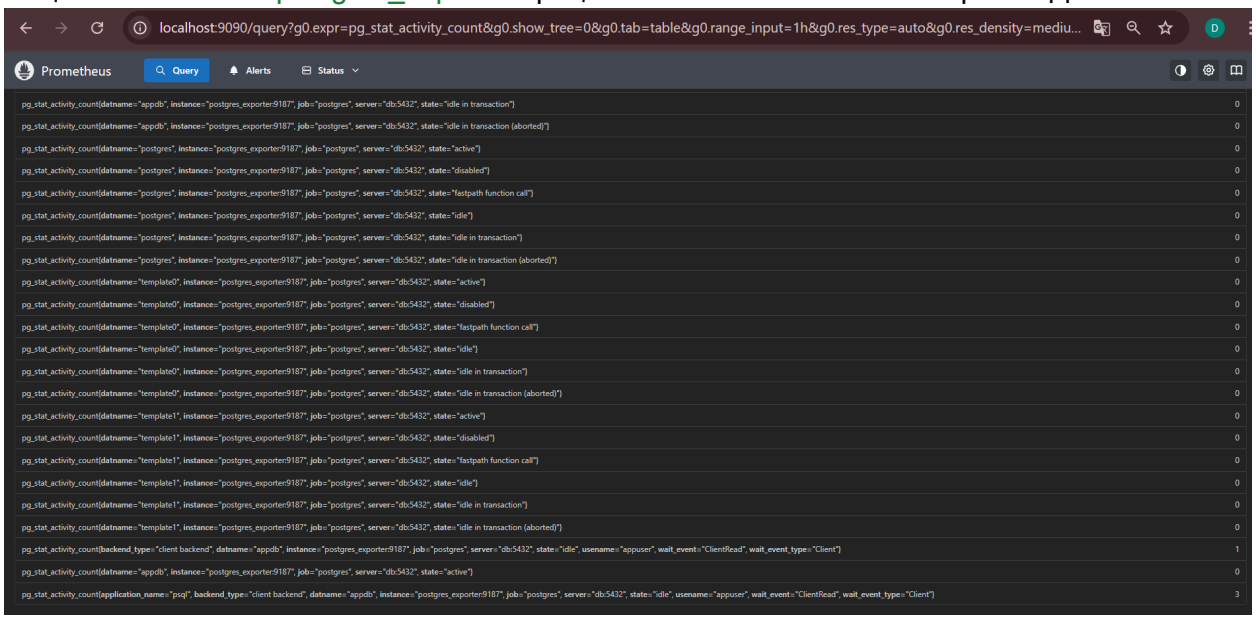
pg_stat_activity_count

- Показує кількість активних з'єднань до PostgreSQL.
- Якщо число є → експортер працює і Prometheus отримує метрики.

(У метриці pg_stat_activity_count більшість значень дорівнюють 0, оскільки в системі є лише два активні джерела підключень: модуль db-loader та сам postgres_exporter. Інші можливі стани активності (наприклад, fastpath function call, idle in transaction, aborted, disabled) у локальному проєкті не використовуються, тому їхній лічильник нульовий.

Це свідчить про правильну роботу експортера та про відсутність зайвих або завислих транзакцій у системі.)

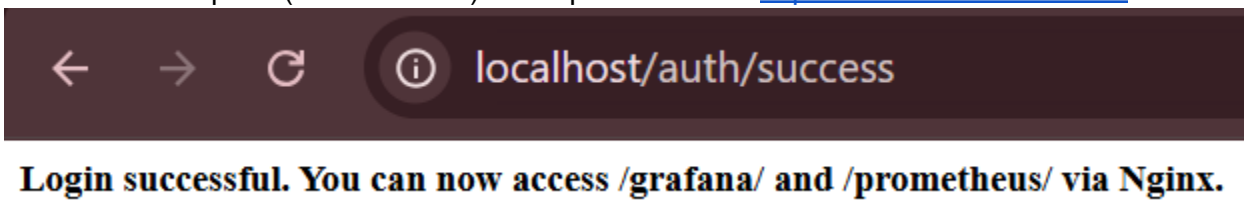
Якщо значення є — postgres_exporter працює і Prometheus бачить метрики БД.



4.4. Перевірка авторизації та доступу через Nginx

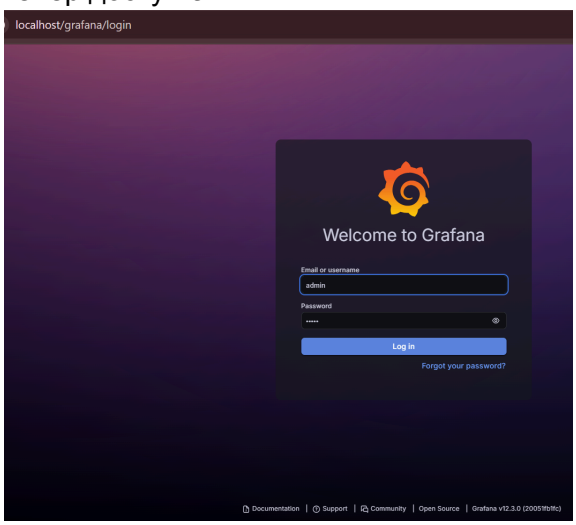
1. Зайти на головну: http://localhost/
   (має перекинути на http://localhost/auth/).

2. Ввести логін/пароль (admin/admin) → потрапляємо на: http://localhost/auth/success

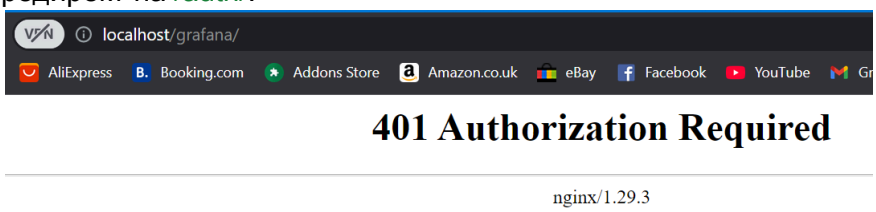


3. Відкрити: http://localhost/grafana/

Тепер доступ є.

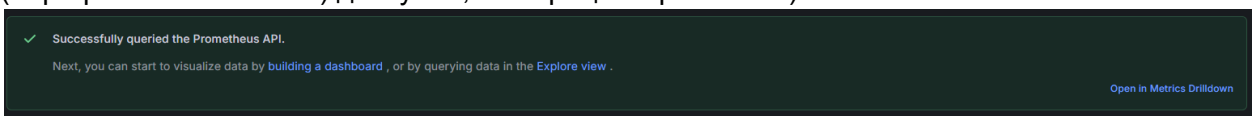


Якщо відкрити цю ж адресу в іншому інкогніто-вікні **без логіну** — отримаємо 401 / редирект на /auth/.



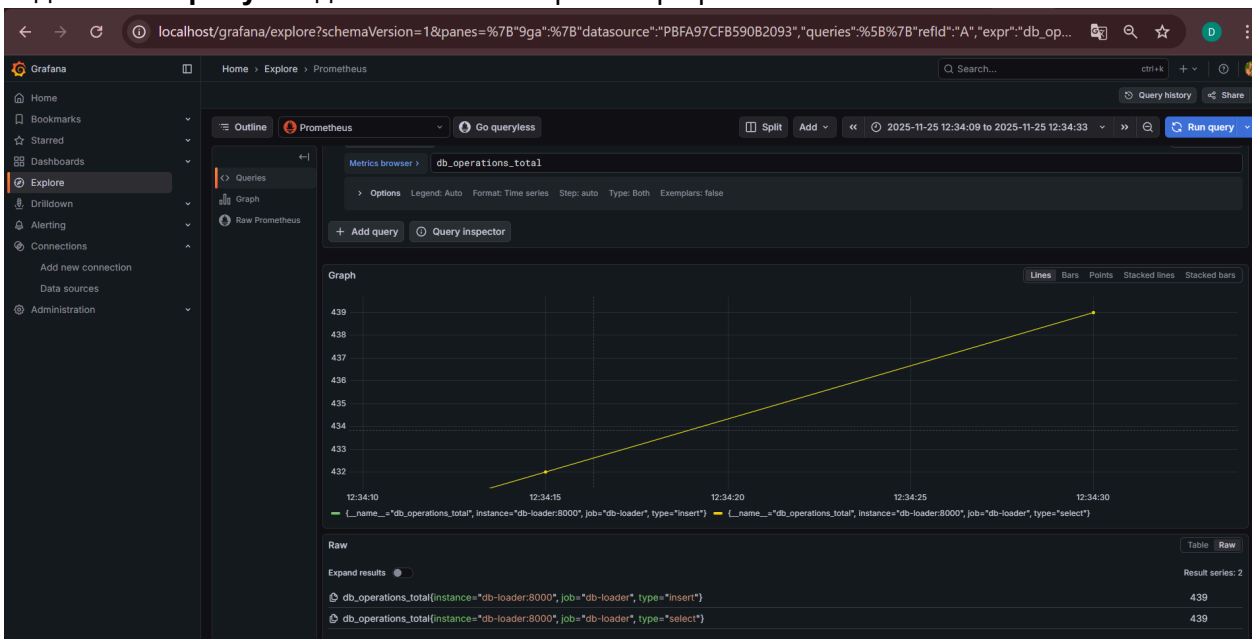
4.5. Перевірка Grafana + Prometheus

1. http://localhost/grafana/ → логін в Grafana (admin / admin).
2. Connections → Data sources → prometheus-1 → Save & test — має бути “Successfully queried the Prometheus API”.
   (Це означає: Grafana підключилася до Prometheus, адреса Prometheus у docker (http://prometheus:9090/) доступна, все працює правильно)



3. Explore → data source prometheus-1→ У панелі запиту натискаєш **Code**:
   Запит: db_operations_total

Водимо **Run query -** видає значення метрик та графік.



Можна створити простий дашборд:

- **New dashboard → Add panel → Query**:
    - ввести rate(db_operations_total[1m]); зберегти панель.

4.6. Перевірка Nginx

Файл конфігурації: ./nginx/nginx.conf

Ключові блоки (скорочено):

```
upstream grafana_upstream { server grafana:3000; }
upstream prometheus_upstream { server prometheus:9090; }
upstream auth_upstream { server auth-server:5000; }
server {
    listen 80;
    location /auth/ {
        proxy_pass http://auth_upstream/;
    }
    location = /auth_verify {
        internal;
        proxy_pass http://auth_upstream/verify;
        proxy_pass_request_body off;
        proxy_set_header Content-Length "";
    }
    location /grafana/ {
```

```
        auth_request /auth_verify;
        proxy_pass http://grafana_upstream;
        ...
    }
    location = / {
        return 302 /auth/;
    }
}
```

Перезапуск після змін:

```
docker compose restart nginx
```

5. Як працюють Prometheus та Grafana в нашій програмі

- **Prometheus**
    - періодично (наприклад, кожні 15 секунд) робить HTTP-запити до:
        - db-loader:8000/metrics;
        - postgres_exporter:9187/metrics;
    - парсить текстовий формат метрик;
    - зберігає їх у time-series базу (/prometheus);
    - дозволяє виконувати PromQL-запити через веб-інтерфейс http://localhost:9090.

- **Grafana**
    - підключається до Prometheus як до джерела даних;
    - через UI (/grafana/) ти створюєш панелі, які виконують PromQL-запити;
    - відображає результати у вигляді графіків, таблиць, alert-панелей;
    - доступ до інтерфейсу йде через Nginx і сервер авторизації.