



Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №4
Тенденції розвитку інформаційних систем та технологій
Системи моніторингу. Prometheus+Cragana.

Виконав
студент групи IT-41ф

Новиков Д. М.

Перевірив:

ас. Цимбал С. І.

Мета роботи: ознайомлення із централізованими системами моніторингу на прикладі Prometheus та Grafana.

Завдання:

1. Запустити стек з Prometheus/Grafana.
2. Модифікувати використовуваний раніше застосунок таким чином, щоб він давав доступ до обраних метрик. Також, створити Grafana дашборд для відображення даних метрик.
3. Скласти звіт відповідно до виконаних завдань. У звіті вказати посилання на власний репозиторій з кодом самостійно створеного застосунку.

Хід роботи:

1. Модифікуємо застосунок із лабораторної роботи №3 для підтримки збору метрик. Створимо його копію з назвою проекту EFKLoggingApiMetrics:
 - а) Додамо необхідну NuGet-залежність:
prometheus-net.AspNetCore
 - б) Модифікуємо файл Program.cs, щоб налаштувати збір метрик HTTP-сервісу. Це дозволить конфігурувати ASP.NET Core request pipeline для збору метрик Prometheus щодо оброблених HTTP-запитів:

```
public class Program
{
    public static void Main(string[] args)
    {
        try
        {
            var builder = WebApplication.CreateBuilder(args);

            ...
            builder.Services.UseHttpClientMetrics();
            ...

            var app = builder.Build();
            ...
            app.UseMetricServer();
            app.UseHttpMetrics();
            ...

            app.Run();
        }
        ...
    }
}
```

- в) Модифікуємо існуючий контролер MaaaaahController, зокрема метод ThrowErrorMaaaaaahMessage, щоб він повертав код 403 Forbidden у випадку невалідного інпуту:

```

[HttpGet("ThrowErrorMaaaaaahMessage/{id}", Name = "ThrowErrorMaaaaaahMessage")]
0 references
public string ThrowErrorMaaaaaahMessage(int id)
{
    _logger.LogInformation("ThrowErrorMaaaaaahMessage requested.");

    try
    {
        if (id <= 0)
        {
            Response.StatusCode = (int)HttpStatusCode.Forbidden;
            throw new Exception($"id cannot be less than or equal to 0. Value passed is [{id}].");
        }

        return id.ToString();
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, ex.Message);
    }

    return string.Empty;
}

```

- г) Деплой буде виконано за допомогою єдиного сценарію docker-compose. Для цього створимо файл Compose.dockerfile, який буде використано в сценарії деплоя для розгортання застосунку EFKLoggingApiMetrics:

```

# See https://aka.ms/customizecontainer to learn how to customize your debug container and
how Visual Studio uses this Dockerfile to build your images for faster debugging.

# This stage is used when running from VS in fast mode (Default for Debug configuration)
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER $APP_UID
WORKDIR /app
EXPOSE 8080

# This stage is used to build the service project
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["EFKLoggingApiMetrics/EFKLoggingApiMetrics.csproj", "EFKLoggingApiMetrics/"]
RUN dotnet restore "./EFKLoggingApiMetrics/EFKLoggingApiMetrics.csproj"
COPY . .
WORKDIR "/src/EFKLoggingApiMetrics"
RUN dotnet build "./EFKLoggingApiMetrics.csproj" -c $BUILD_CONFIGURATION -o /app/build

# This stage is used to publish the service project to be copied to the final stage
FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./EFKLoggingApiMetrics.csproj" -c $BUILD_CONFIGURATION -o /app/publish
/p:UseAppHost=false

# This stage is used in production or when running from VS in regular mode (Default when
not using the Debug configuration)
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "EFKLoggingApiMetrics.dll"]

```

2. Підготуємо файл docker-compose.yml для налаштування роботи нашого застосунку, а також сервісів Prometheus та Grafana:

```
services:
  my-api:
    image: ${DOCKER_REGISTRY-}my-api
    container_name: my-api
    ports:
      - 8080:8080
    build:
      context: .
      dockerfile: EFKLoggingApiMetrics/Compose.Dockerfile
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_HTTP_PORTS=8080

  prometheus:
    image: prom/prometheus
    container_name: prometheus
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml:ro
      - prometheus_data:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--web.console.libraries=/usr/share/prometheus/console_libraries'
      - '--web.console.templates=/usr/share/prometheus/consoles'
    restart: always

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana_data:/var/lib/grafana
    depends_on:
      - prometheus
    restart: always

volumes:
  prometheus_data: {}
  grafana_data: {}
```

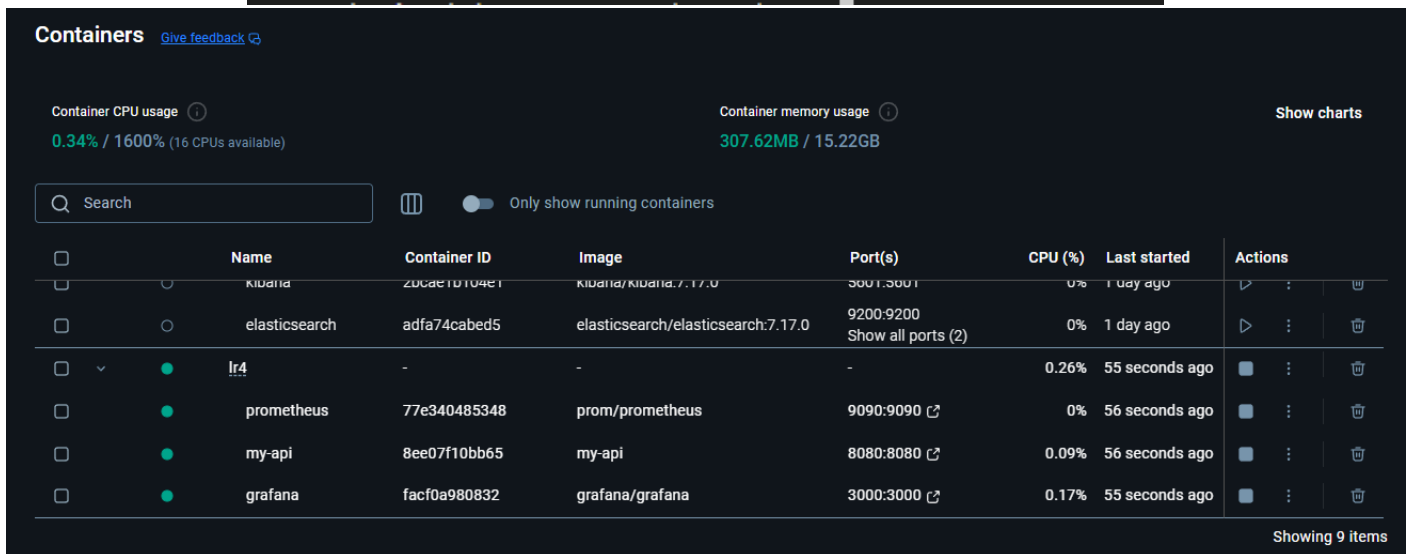
Prometheus.yml:

```
global:
  scrape_interval: 10s
scrape_configs:
  - job_name: 'my-api-read-prometheus'
    static_configs:
      - targets: ['my-api:8080']
```

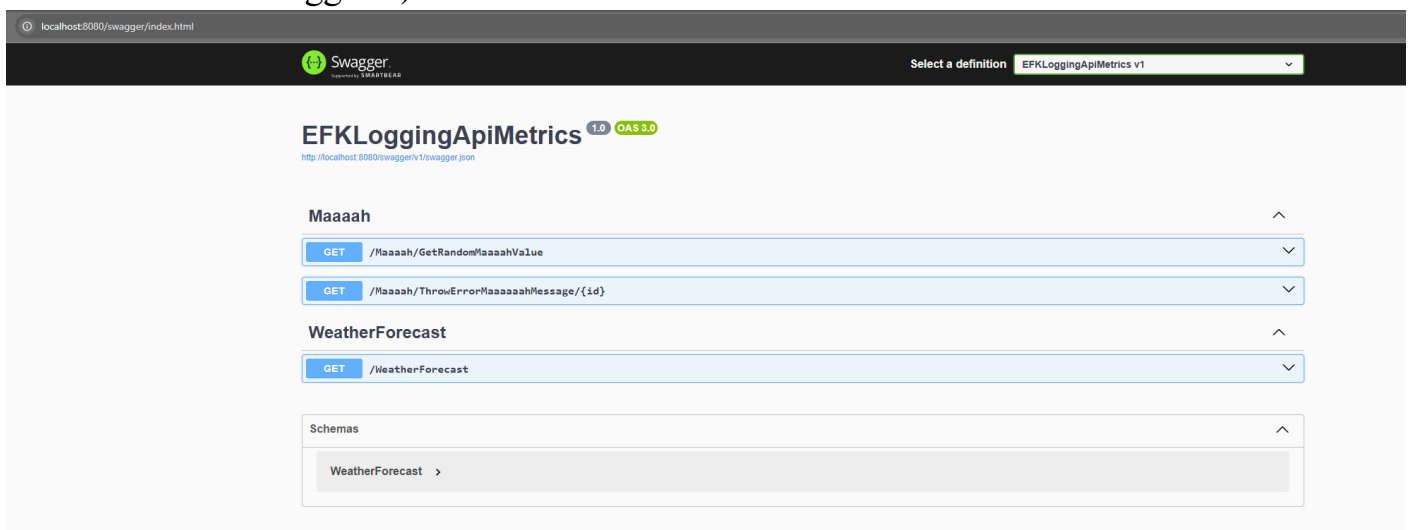
3. Перевіримо працездатність:

а) Запустимо стек за допомогою команди docker-compose up:

```
[+] Running 4/4
✓ Network lr4_default Created
✓ Container prometheus Started
✓ Container my-api Started
✓ Container grafana Started
```



б) Перевіримо роботу застосунку, перейшовши за відповідним посиланням <http://localhost:8080/swagger/index.html> (для доступу до swagger'a):



в) Перевіримо, чи застосунок збирає метрики, перейшовши за посиланням <http://localhost:8080/metrics> (адрес API + /metrics endpoint):

```
localhost:8080/metrics

# HELP http_request_duration_seconds The duration of HTTP requests processed by an ASP.NET Core application.
# TYPE http_request_duration_seconds histogram
http_request_duration_seconds_sum{code="404",method="GET",controller="",action="",endpoint=""} 0.0002168
http_request_duration_seconds_count{code="404",method="GET",controller="",action="",endpoint=""} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="0.001"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="0.002"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="0.004"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="0.008"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="0.016"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="0.032"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="0.064"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="0.128"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="0.256"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="0.512"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="1.024"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="2.048"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="4.096"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="8.192"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="16.384"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="32.768"} 1
http_request_duration_seconds_bucket{code="404",method="GET",controller="",action="",endpoint="",le="+Inf"} 1
# HELP http_requests_received_total Provides the count of HTTP requests that have been processed by the ASP.NET Core pipeline.
# TYPE http_requests_received_total counter
http_requests_received_total{code="404",method="GET",controller="",action="",endpoint=""} 1
# HELP http_requests_in_progress The number of requests currently in progress in the ASP.NET Core pipeline. One series without controller/action label values counts all in-progress requests, with
separate series existing for each controller-action pair.
# TYPE http_requests_in_progress gauge
http_requests_in_progress{method="GET",controller="",action="",endpoint=""} 0
# HELP dotnet_collection_count_total GC collection count
# TYPE dotnet_collection_count_total counter
dotnet_collection_count_total{generation="0"} 0
dotnet_collection_count_total{generation="1"} 0
dotnet_collection_count_total{generation="2"} 0
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1733177185.2669125
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 2.11
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 2828022150144
# HELP process_working_set_bytes Process working set
# TYPE process_working_set_bytes gauge
process_working_set_bytes 132292608
# HELP process_private_memory_bytes Process private memory size
# TYPE process_private_memory_bytes gauge
```

г) Перевіримо, чи Prometheus працює коректно і чи має підключення до нашого API. Для цього відкриємо сторінку <http://localhost:9090/targets>:

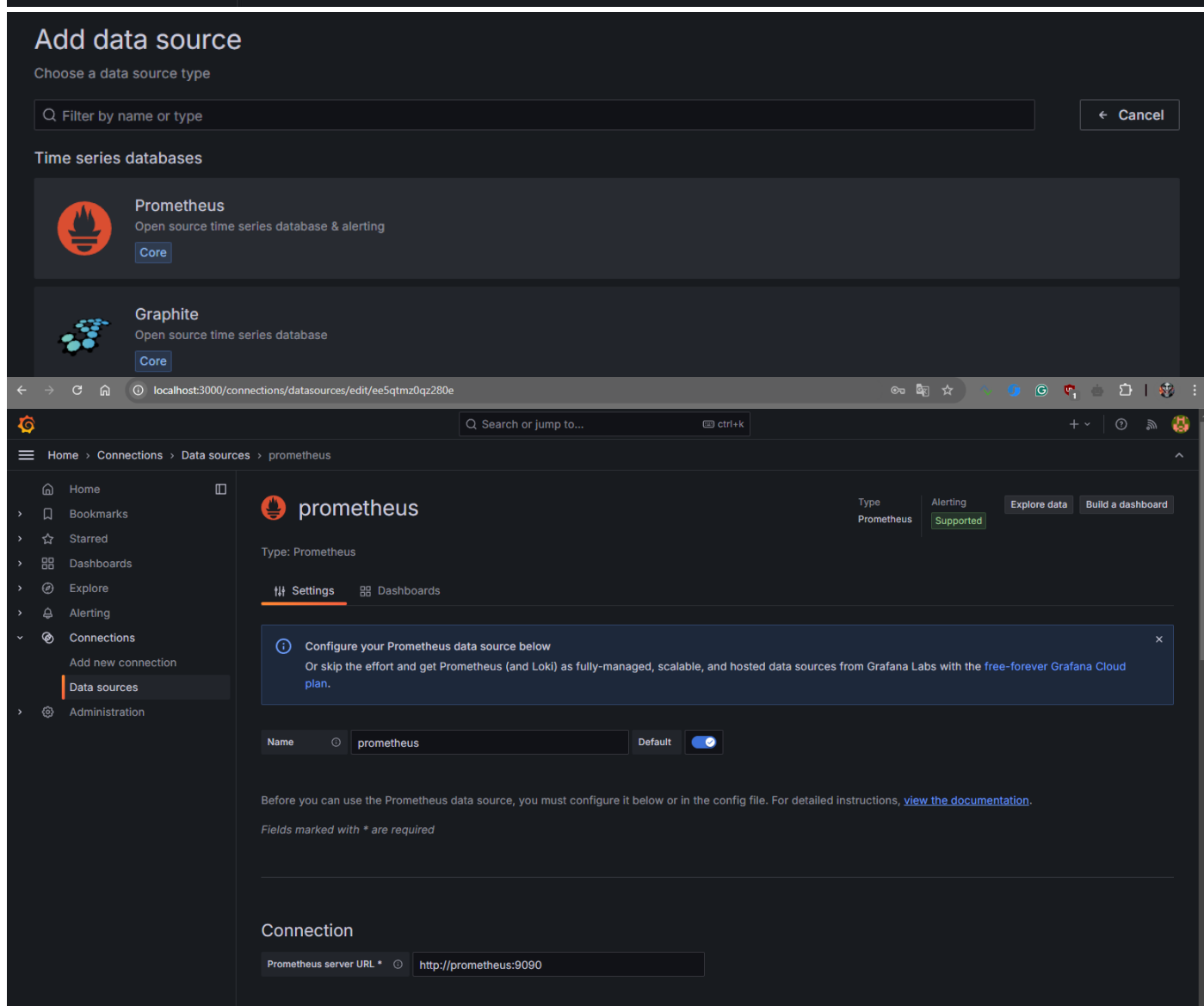
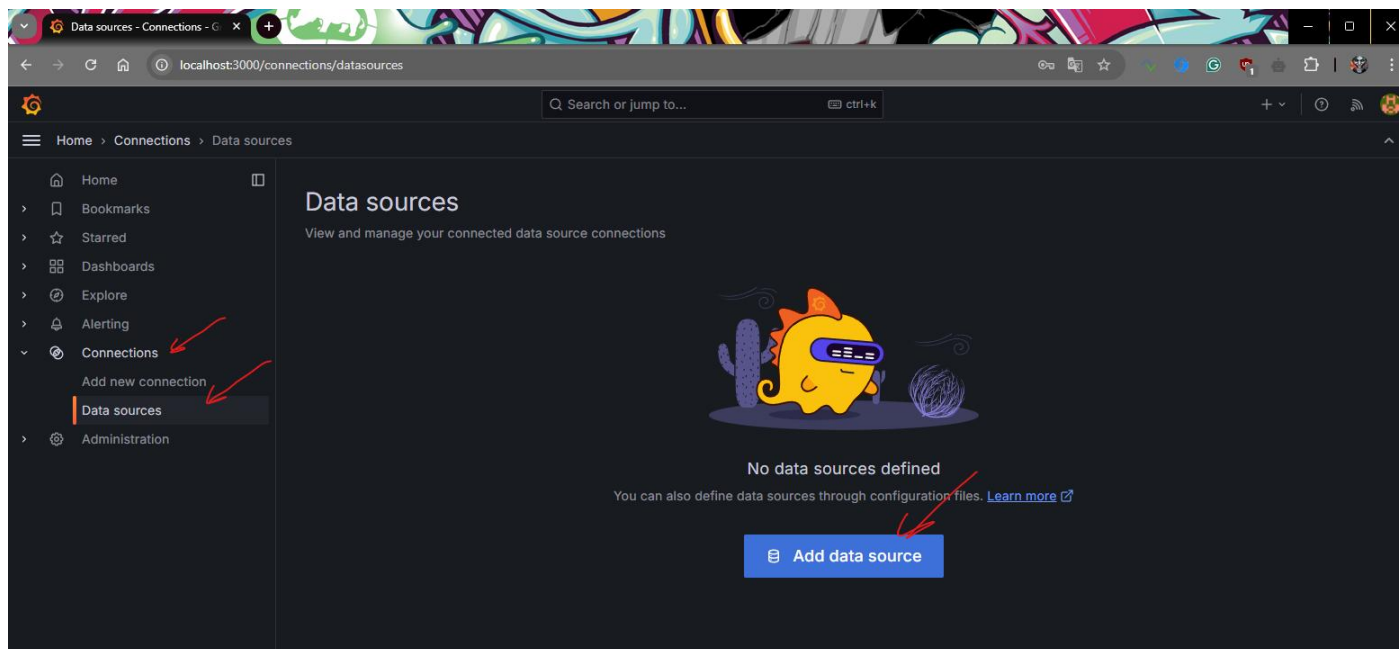
The screenshot shows the Prometheus web interface at localhost:9090/targets. The 'Status > Target health' tab is selected. A table lists the targets:

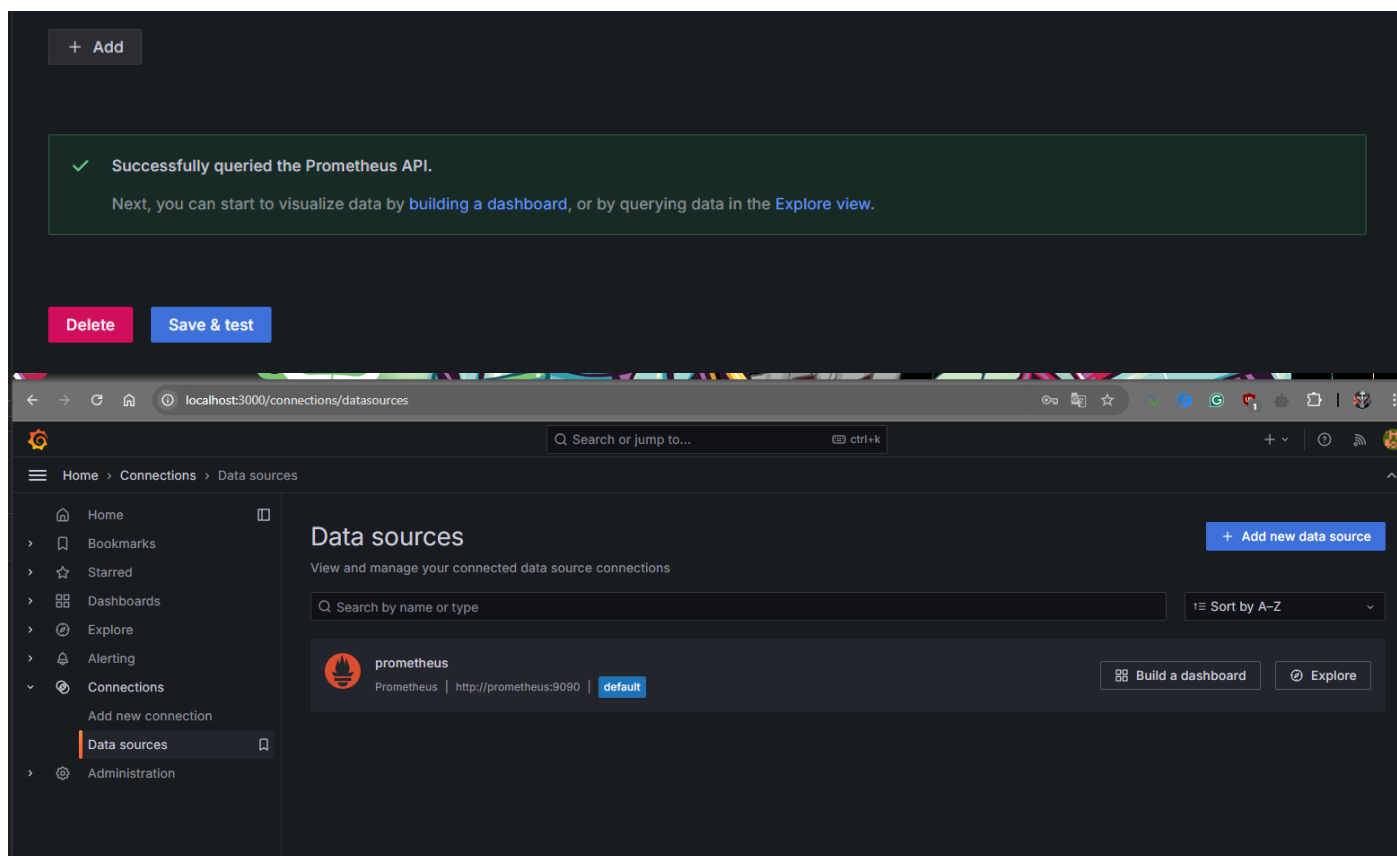
Endpoint	Labels	Last scrape	State
http://my-api:8080/metrics	instance="my-api:8080" job="my-api-read-prometheus"	9.126s ago	UP

д) Перейдемо до Grafana, переконаємося, що вона працює, та налаштуємо Prometheus як джерело даних для Grafana:

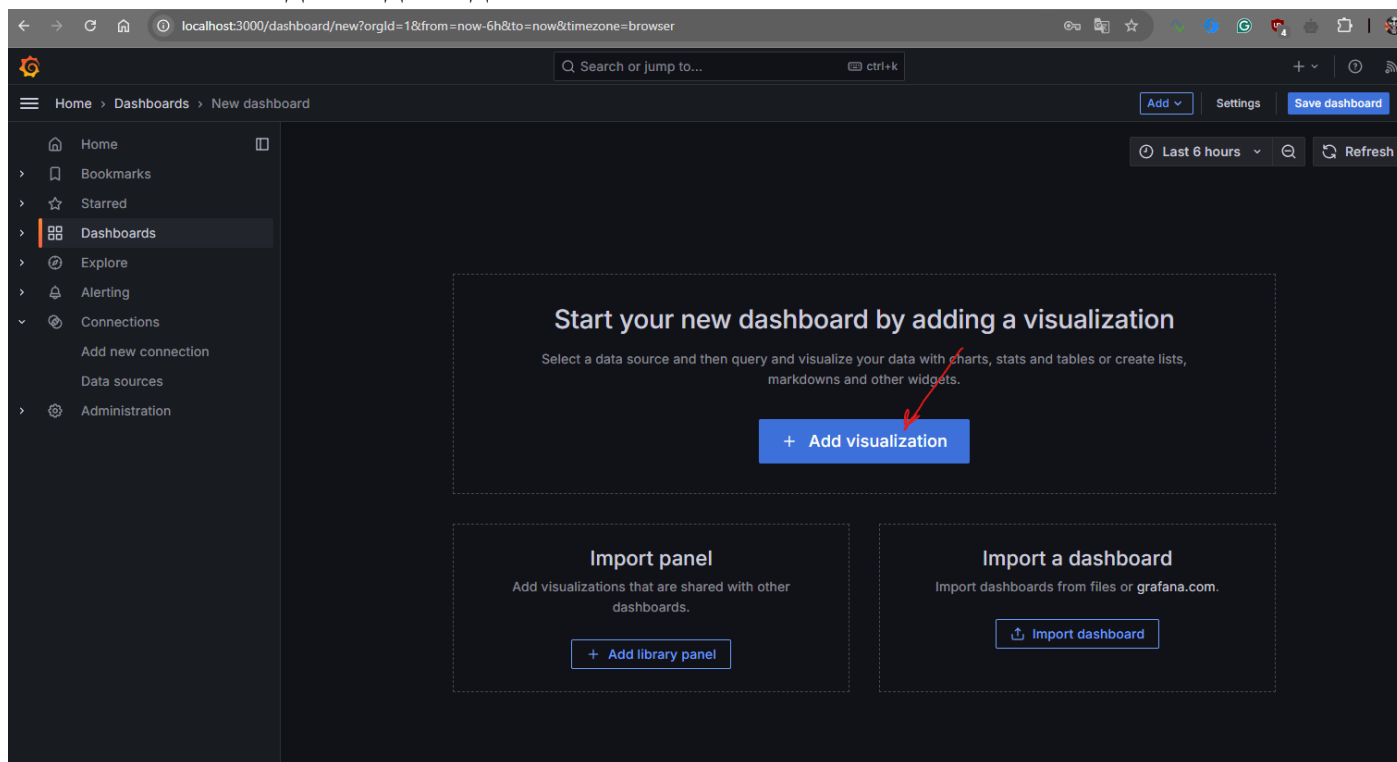
The screenshot shows the Grafana web interface at localhost:3000/login. The page features the Grafana logo and a login form:

- Email or username: admin
- Password: *****
- Log in button
- Forgot your password? link





е) Створимо Dashboard у Grafana для відображення будь-якої метрики. Наприклад, сумарний час виконання HTTP-запитів, згрупований за кодом відповіді:



The image shows two parts of the Grafana interface. The top part is the 'Select data source' dialog, where 'prometheus' is selected as the default data source. The bottom part is the 'Query editor' for a Prometheus query. The query is `sum by(code) (http_request_duration_seconds_bucket)`. The interface includes buttons for 'Run queries', 'Builder', and 'Code'. Red annotations highlight the 'Code' button and the query text.

ж) Виконаємо кілька запитів до нашого застосунку:

- Один успішний (код 200);
- Один із невалідними даними (403 код).

The image shows a REST client interface. The request is a GET to `/Maaaaah/ThrowErrorMaaaaahMessage/{id}` with the parameter `id = -5`. The response is a 403 Forbidden error. The response headers are: `content-length: 0`, `content-type: text/plain; charset=utf-8`, `date: Mon, 02 Dec 2024 22:24:33 GMT`, and `server: Kestrel`.

GET
/Maaaaah/ThrowErrorMaaaaaahMessage/{id}

Parameters
Cancel

Name	Description
id * required integer(\$int32) (path)	1

Execute
Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/Maaaaah/ThrowErrorMaaaaaahMessage/1' \
-H 'accept: text/plain'
```

Request URL

```
http://localhost:8080/Maaaaah/ThrowErrorMaaaaaahMessage/1
```

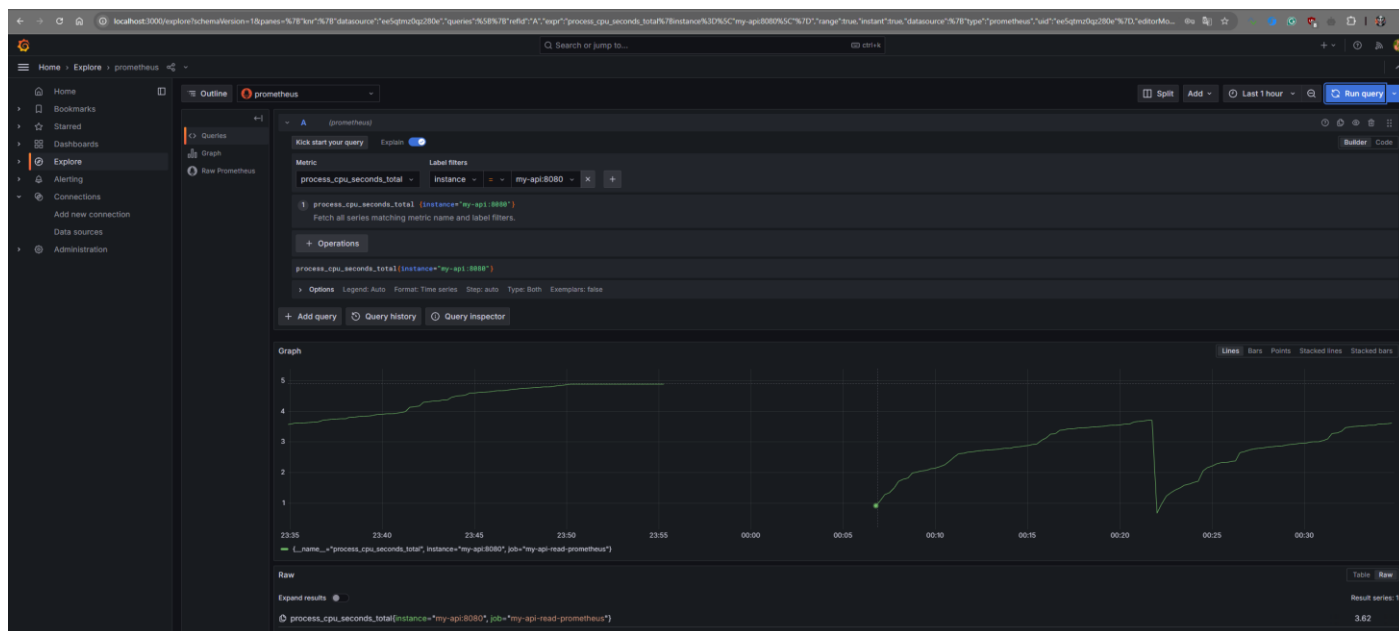
Server response

Code	Details
200	<div>Response body</div> <div>1</div> <div>Download</div> <div>Response headers</div> <pre>content-type: text/plain; charset=utf-8 date: Mon, 02 Dec 2024 22:31:54 GMT server: Kestrel transfer-encoding: chunked</pre>

з) Перевіримо дашборд у Grafana. Він має відображати інформацію про час виконання щойно зроблених запитів, згруповану за кодами відповіді:



и) Перейдімо на вкладку Explore у Grafana, щоб переглянути будь-яку зі зібраних метрик. Наприклад, переглянемо метрику `process_cpu_seconds_total`, яка відображає загальний час використання CPU процесом (на графіку можна побачити перепади, які демонструють, що застосунок було запущено вже втретє: між першим і другим запуском була пауза, пов'язана з редагуванням коду):



Висновки: в результаті виконання цієї лабораторної роботи було ознайомлено з базовими концепціями збору та аналізу метрик на основі технологій Prometheus та Grafana.

На основі отриманих знань було реалізовано практичну частину, яка полягала у:

- модифікації застосунку для підтримки збору метрик;
- налаштуванні стеку, що включає Prometheus та Grafana;
- створенні дашборду для моніторингу метрик застосунку.

У ході виконання роботи було протестовано збір метрик, перевірено їх відображення в Grafana, а також підтверджено працездатність налаштувань через успішні та помилкові HTTP-запити.

Вихідний код застосунку можна знайти за наступним посиланням на [GitHub](#).