****

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 ІТ-41ф |  | Перевірив: |
|  |  |  |
| Новиков Д. М. |  |  |
|  |  |  |
|  |  | ас. Цимбал С. І. |

Київ 2024

*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:
2. Додамо необхідну NuGet-залежність:

prometheus-net.AspNetCore

1. 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();

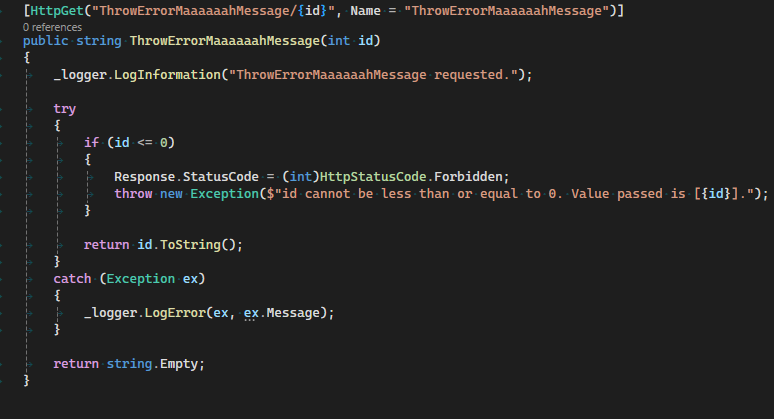
}

…

}

}

1. Модифікуємо існуючий контролер MaaaahController, зокрема метод ThrowErrorMaaaaaahMessage, щоб він повертав код 403 Forbidden у випадку невалідного інпуту:



1. 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"]

1. 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

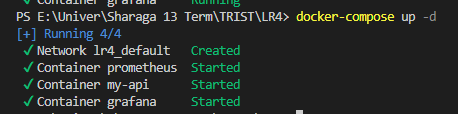
scrape\_configs:

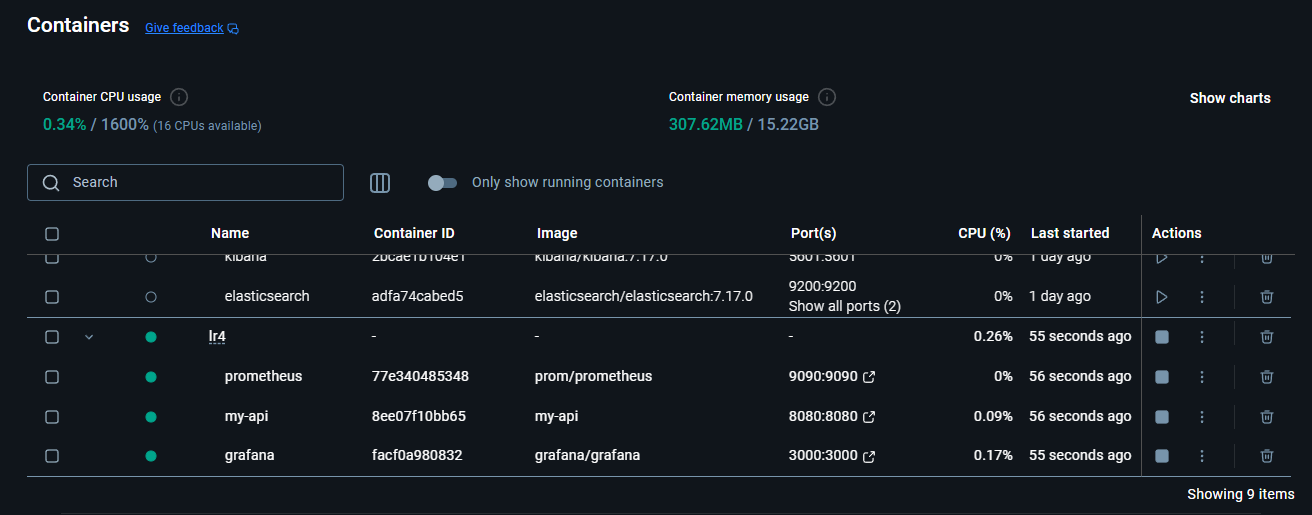
  - job\_name: 'my-api-read-prometheus'

    static\_configs:

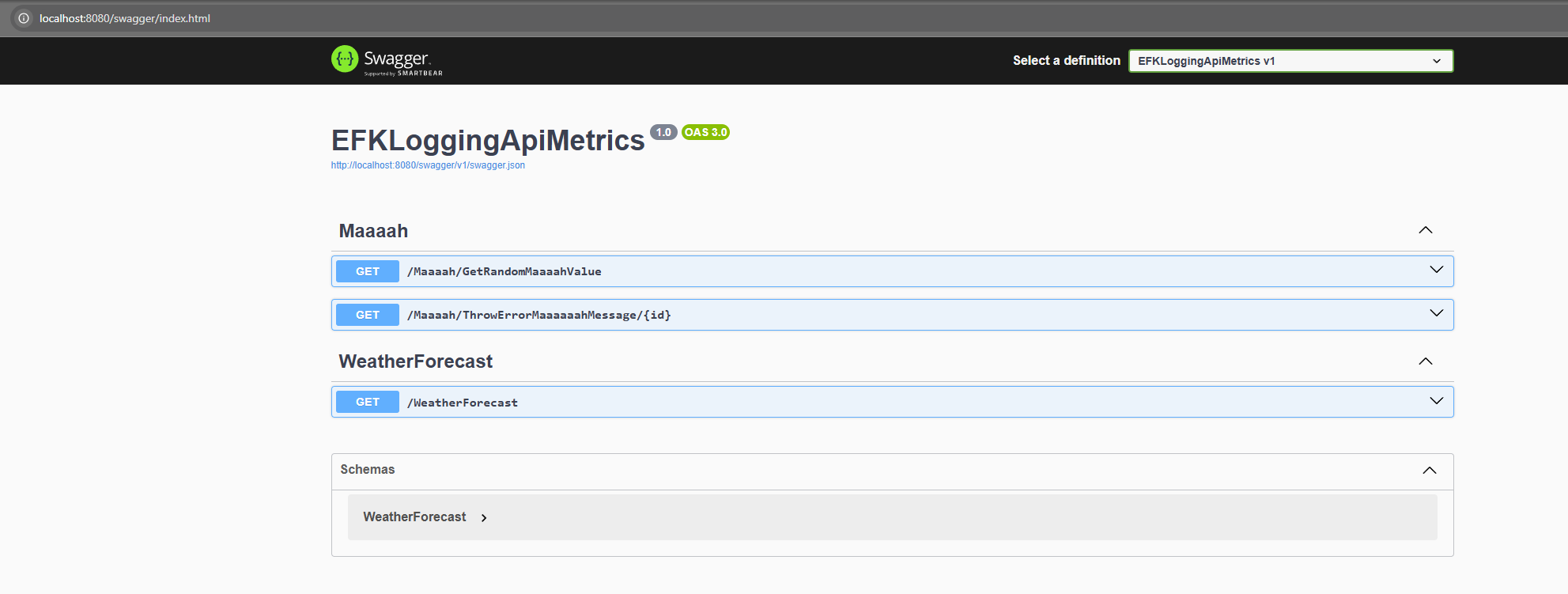
    - targets: ['my-api:8080']

1. Перевіримо працездатність:
2. Запустимо стек за допомогою команди docker-compose up:

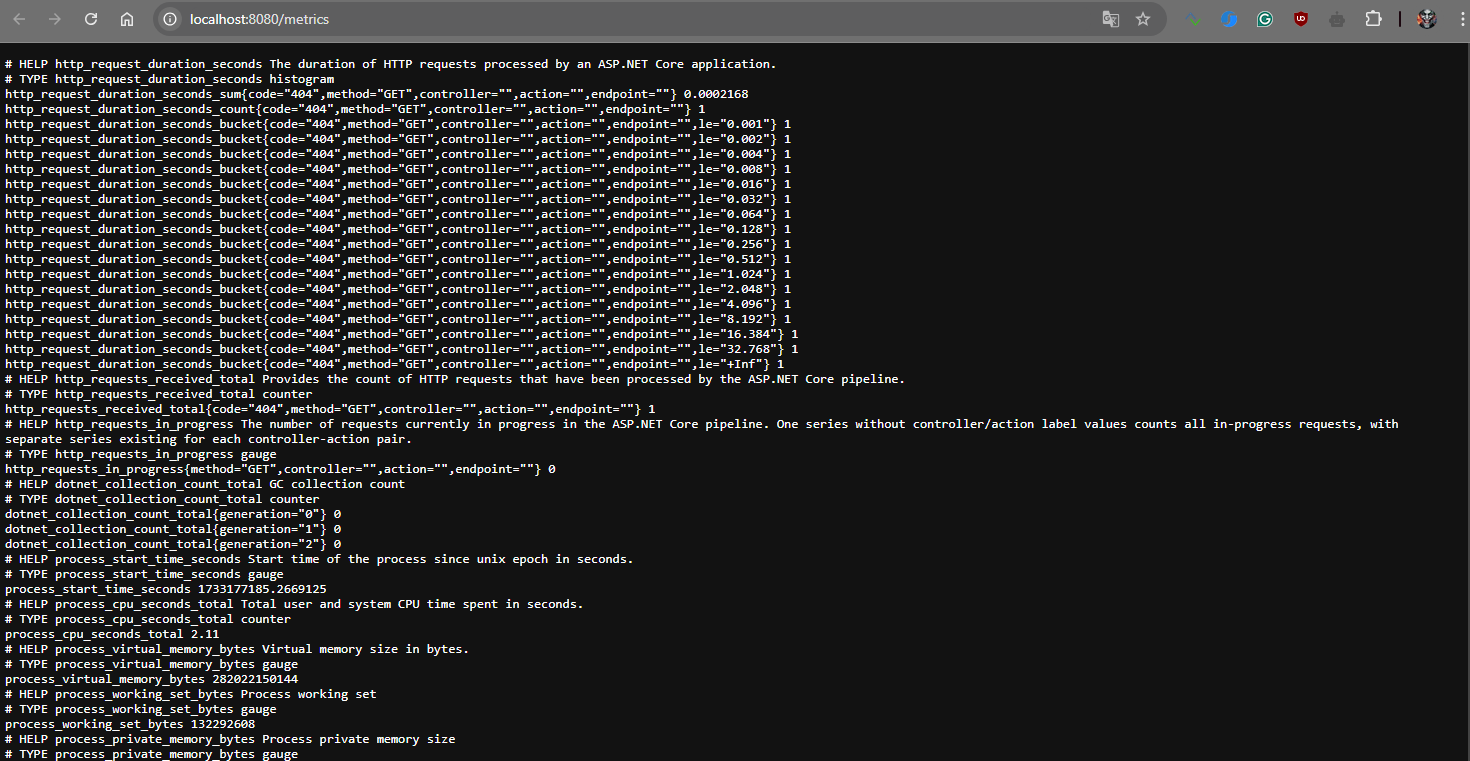




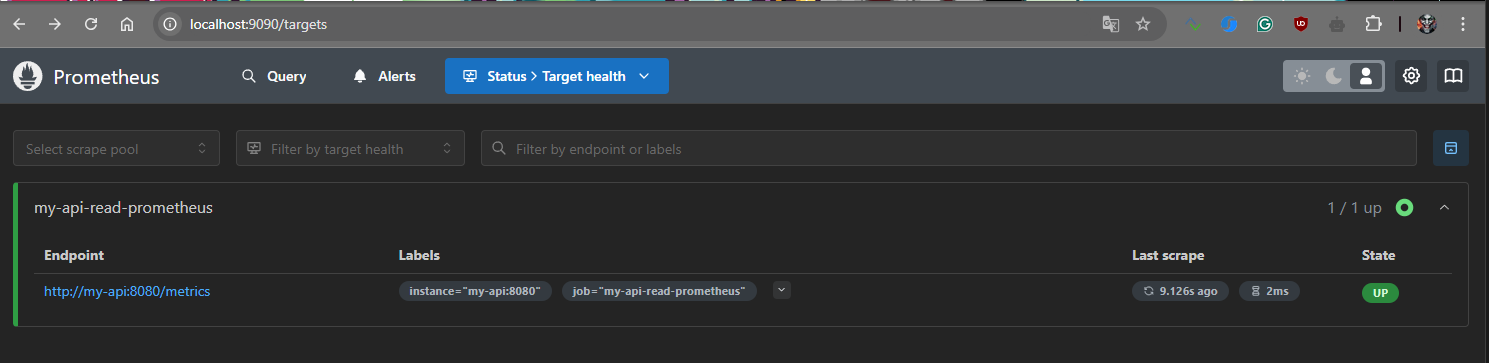
1. Перевіримо роботу застосунку, перейшовши за відповідним посиланням <http://localhost:8080/swagger/index.html> (для доступу до swagger’a):



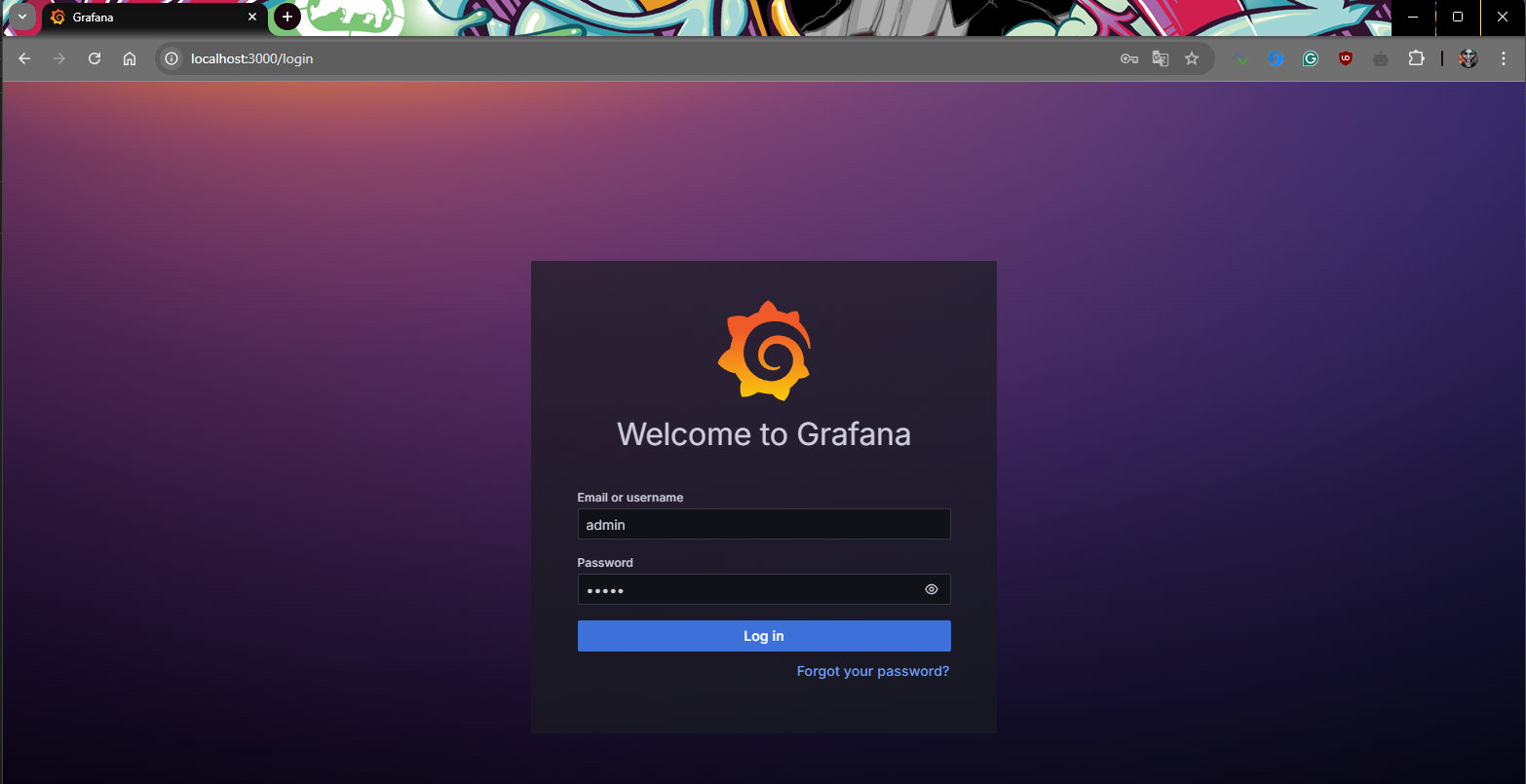
1. Перевіримо, чи застосунок збирає метрики, перейшовши за посиланням <http://localhost:8080/metrics> (адрес API + /metrics endpoint):

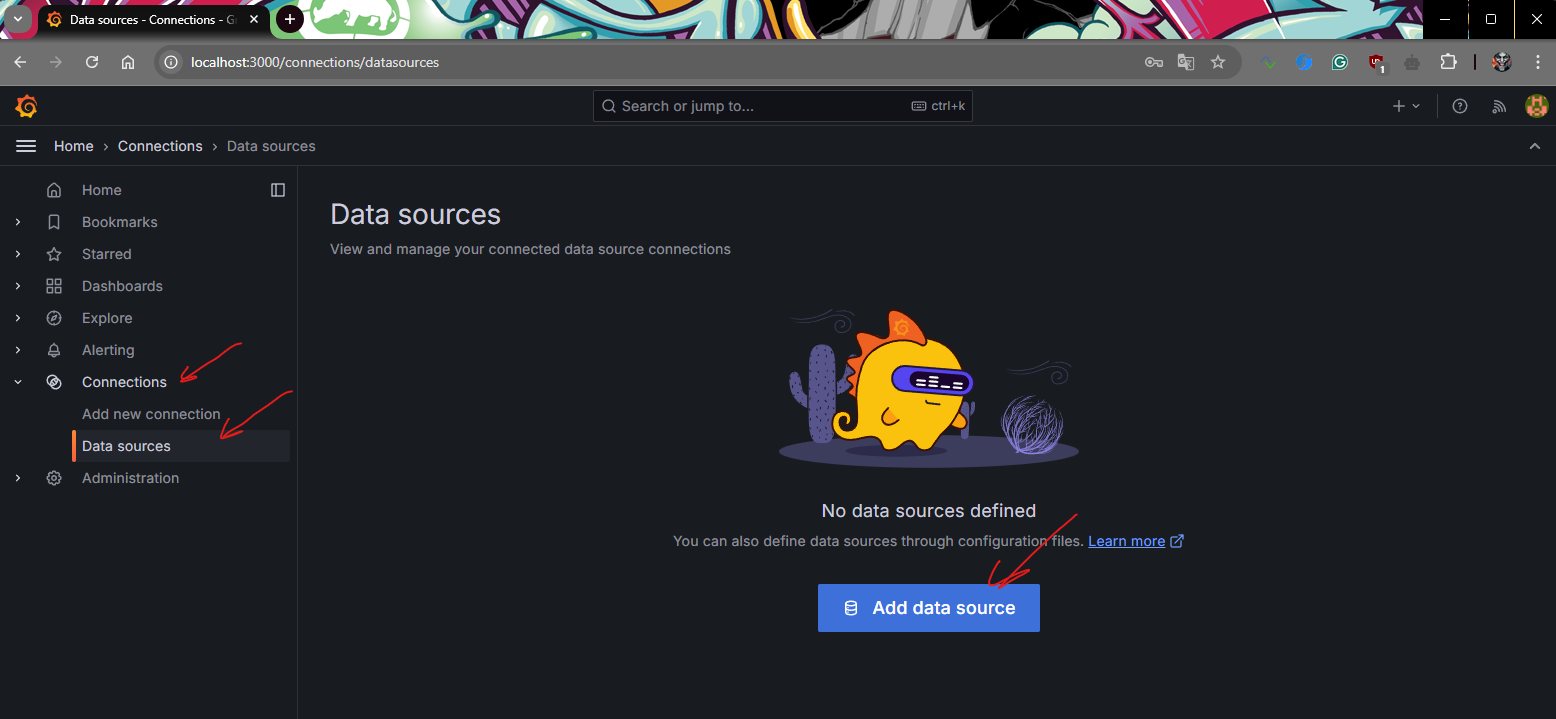


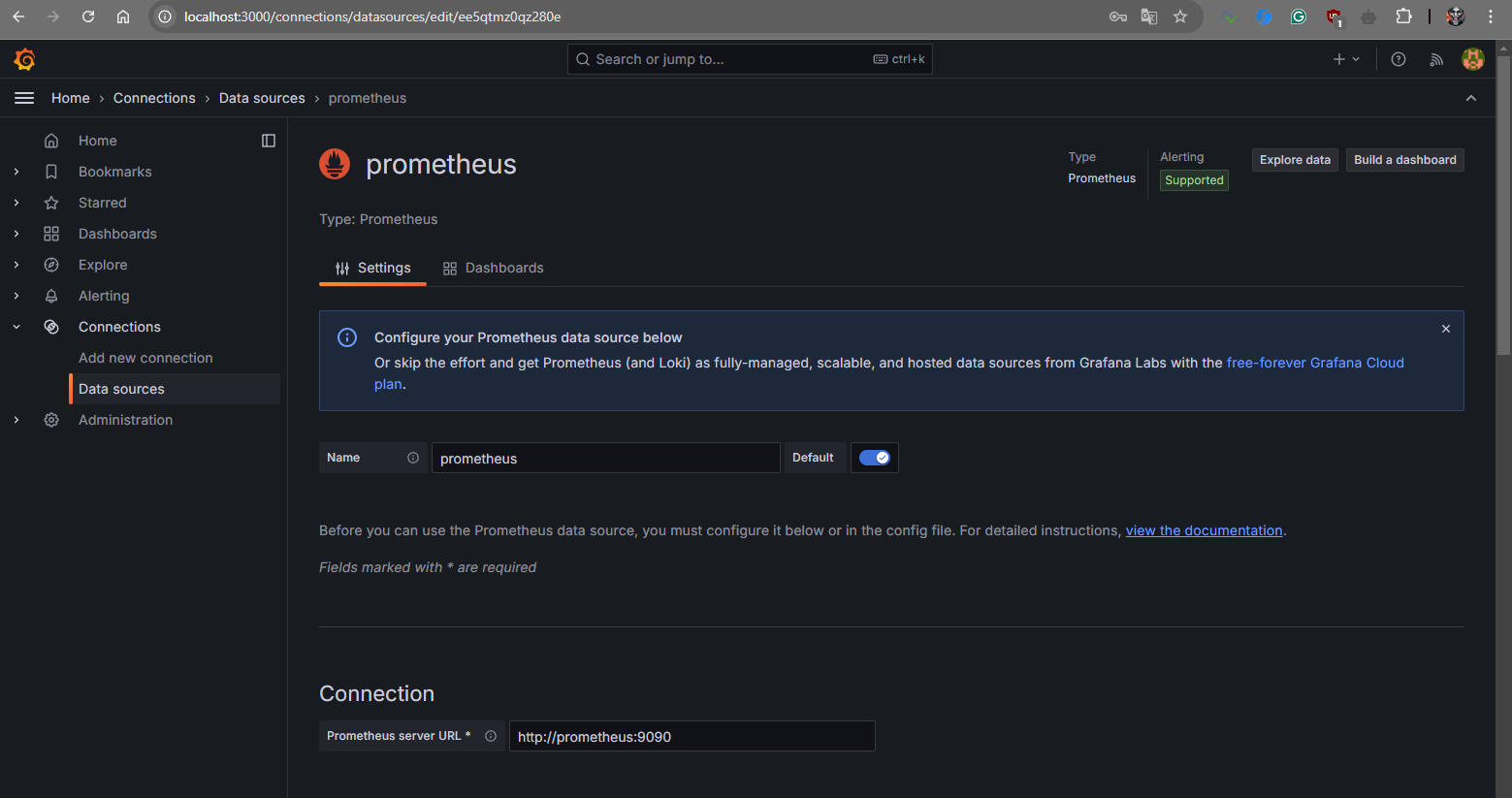
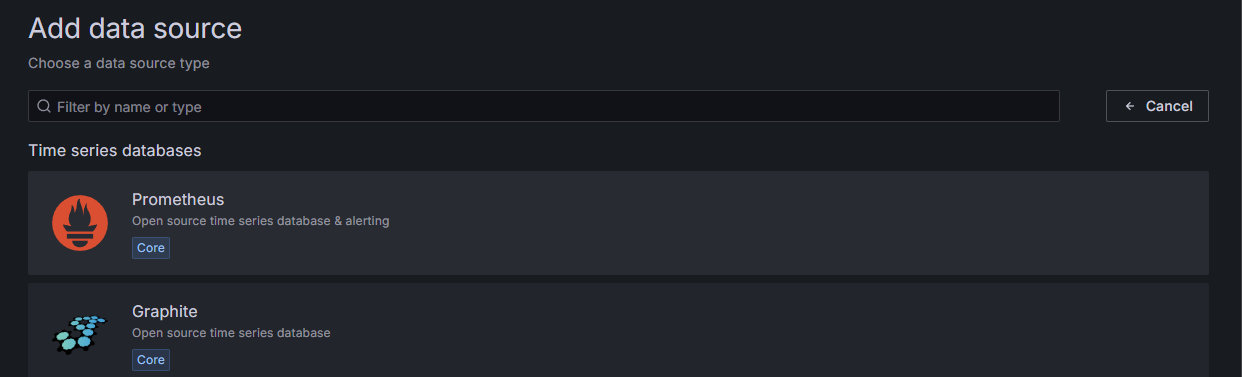
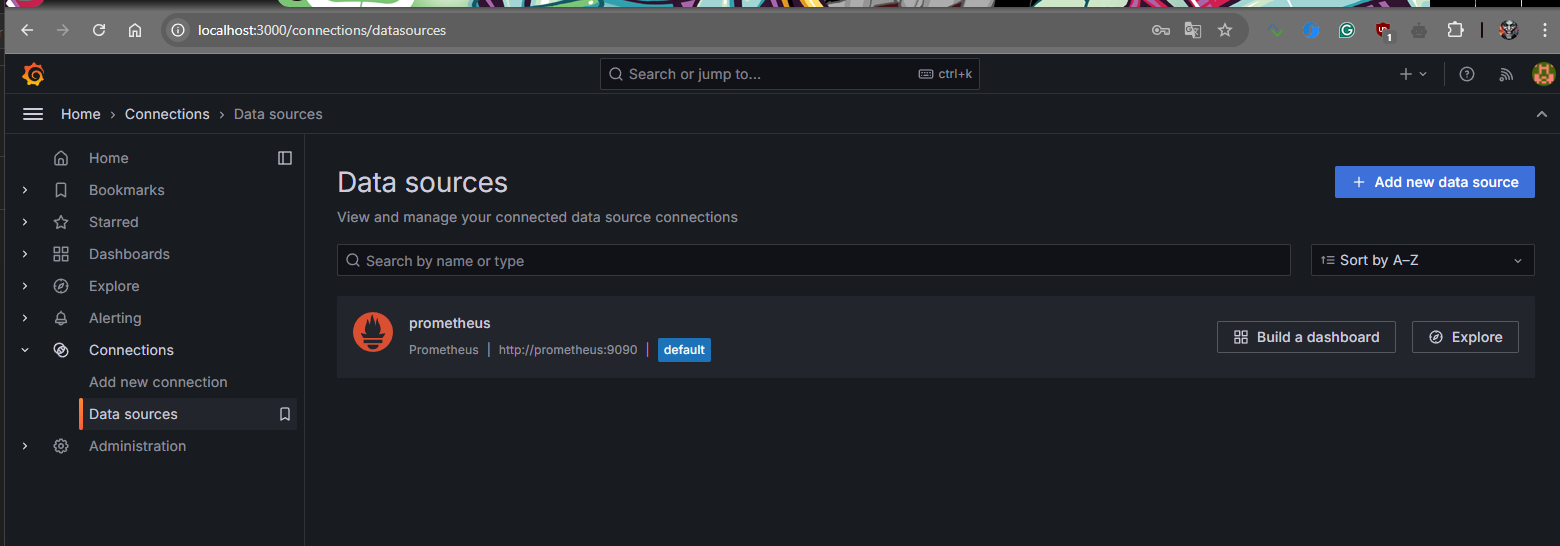
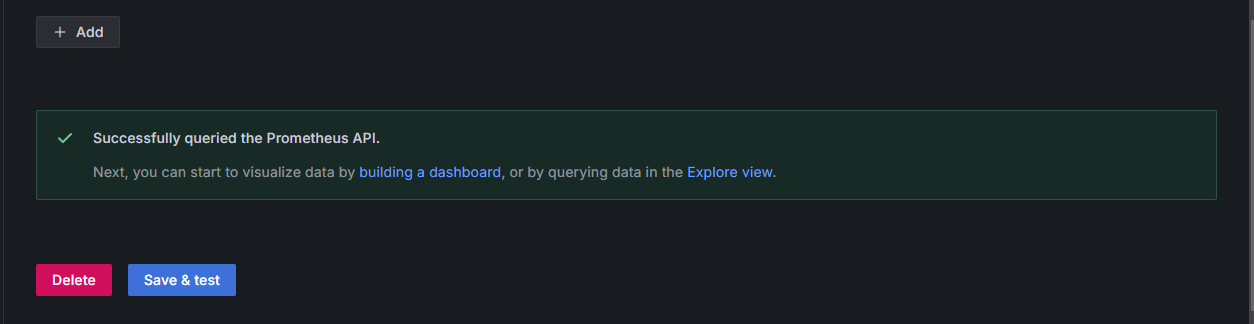
1. Перевіримо, чи Prometheus працює коректно і чи має підключення до нашого API. Для цього відкриємо сторінку <http://localhost:9090/targets>:



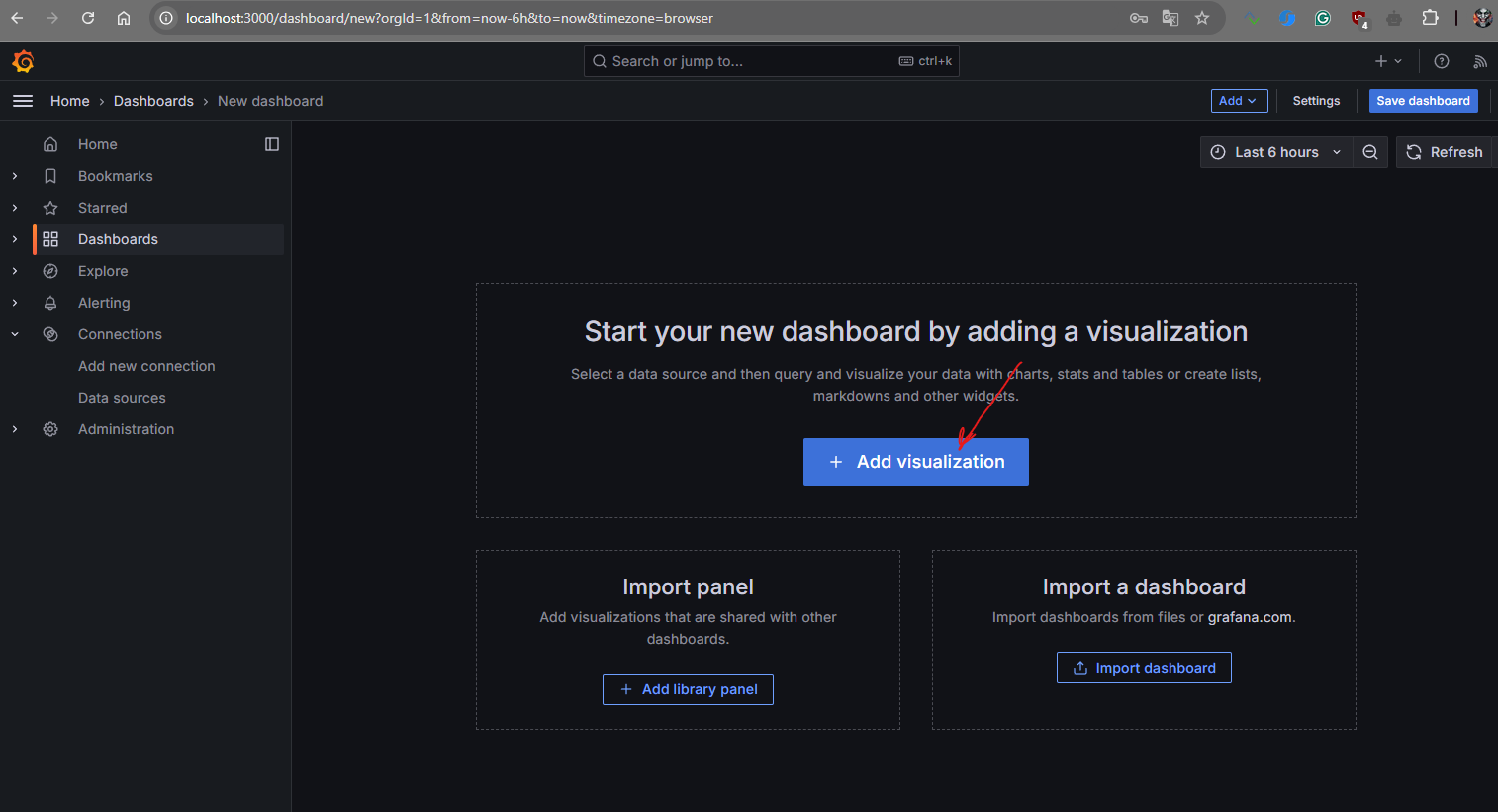
1. Перейдемо до Grafana, переконаємося, що вона працює, та налаштуємо Prometheus як джерело даних для Grafana:

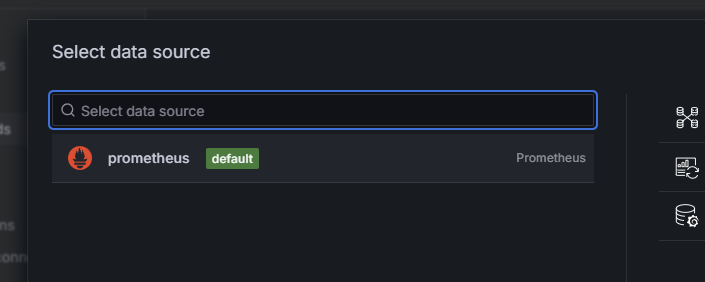


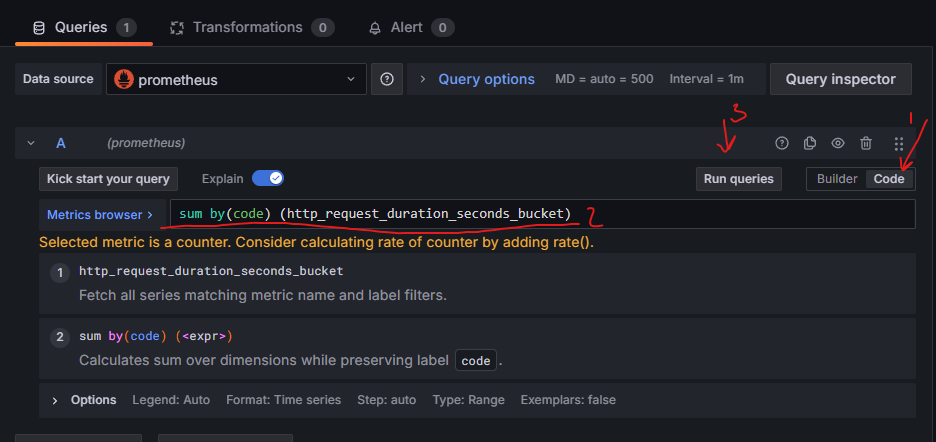


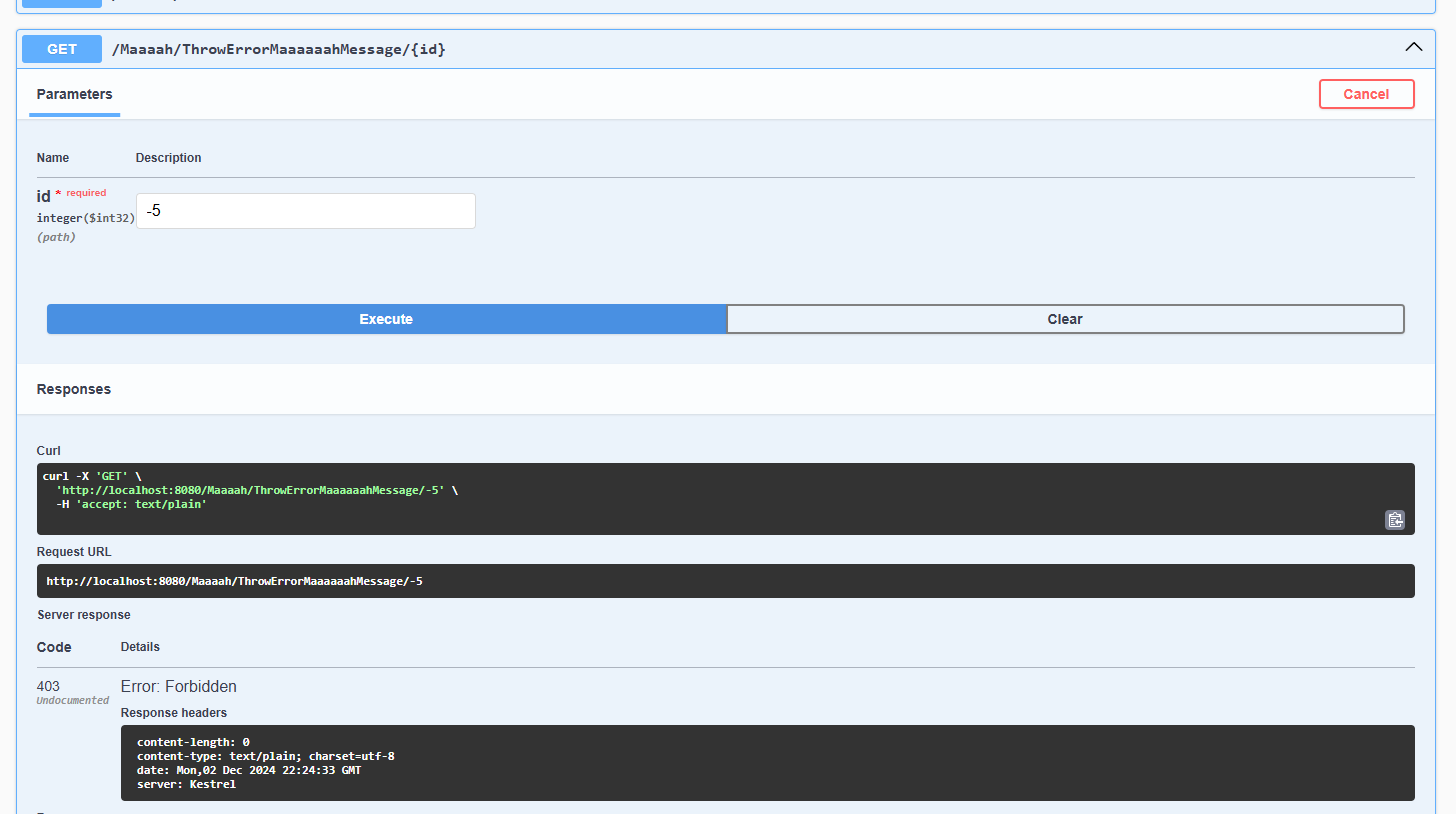
1. Створимо Dashboard у Grafana для відображення будь-якої метрики. Наприклад, сумарний час виконання HTTP-запитів, згрупований за кодом відповіді:

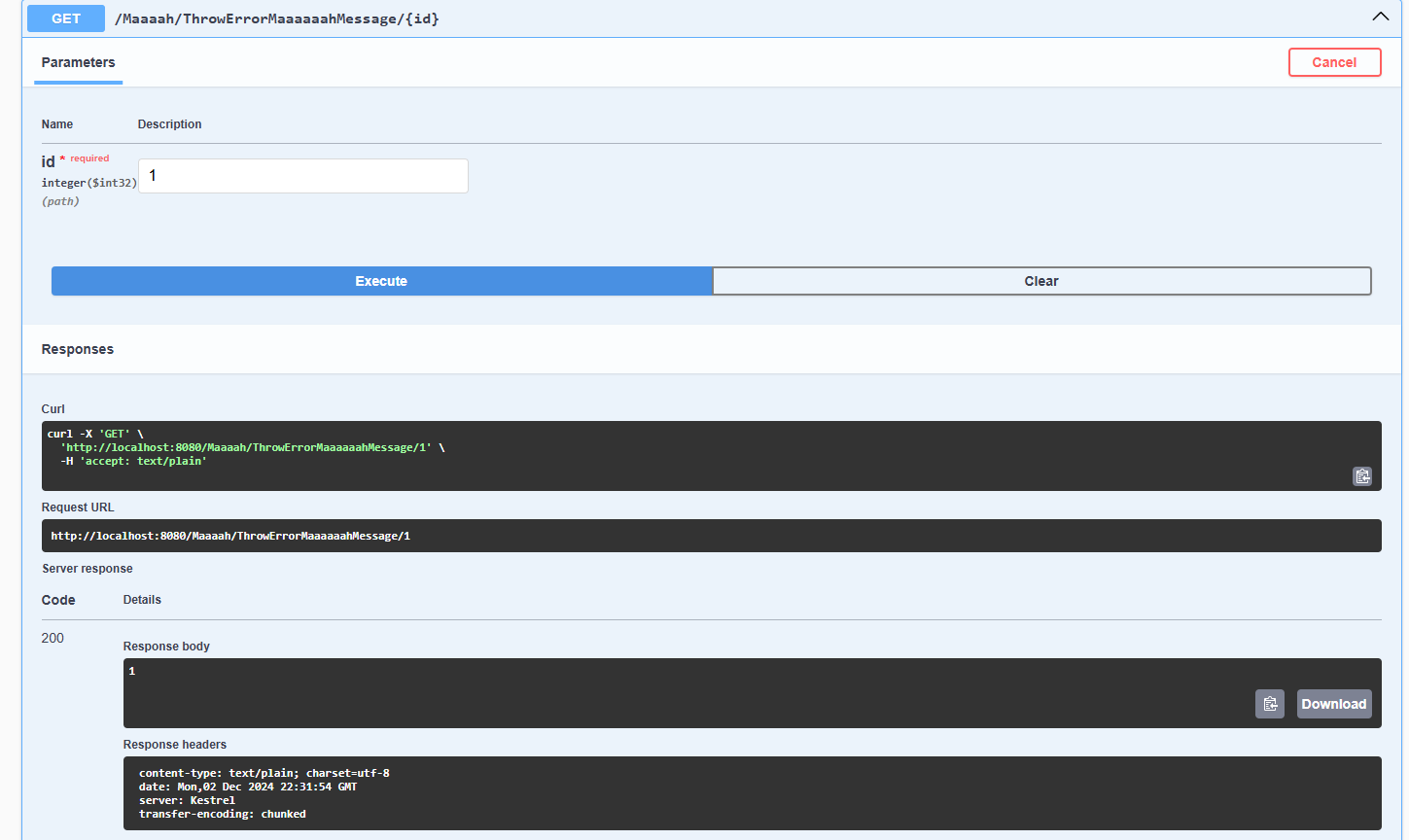






1. Виконаємо кілька запитів до нашого застосунку:
   1. Один успішний (код 200);
   2. Один із невалідними даними (403 код).

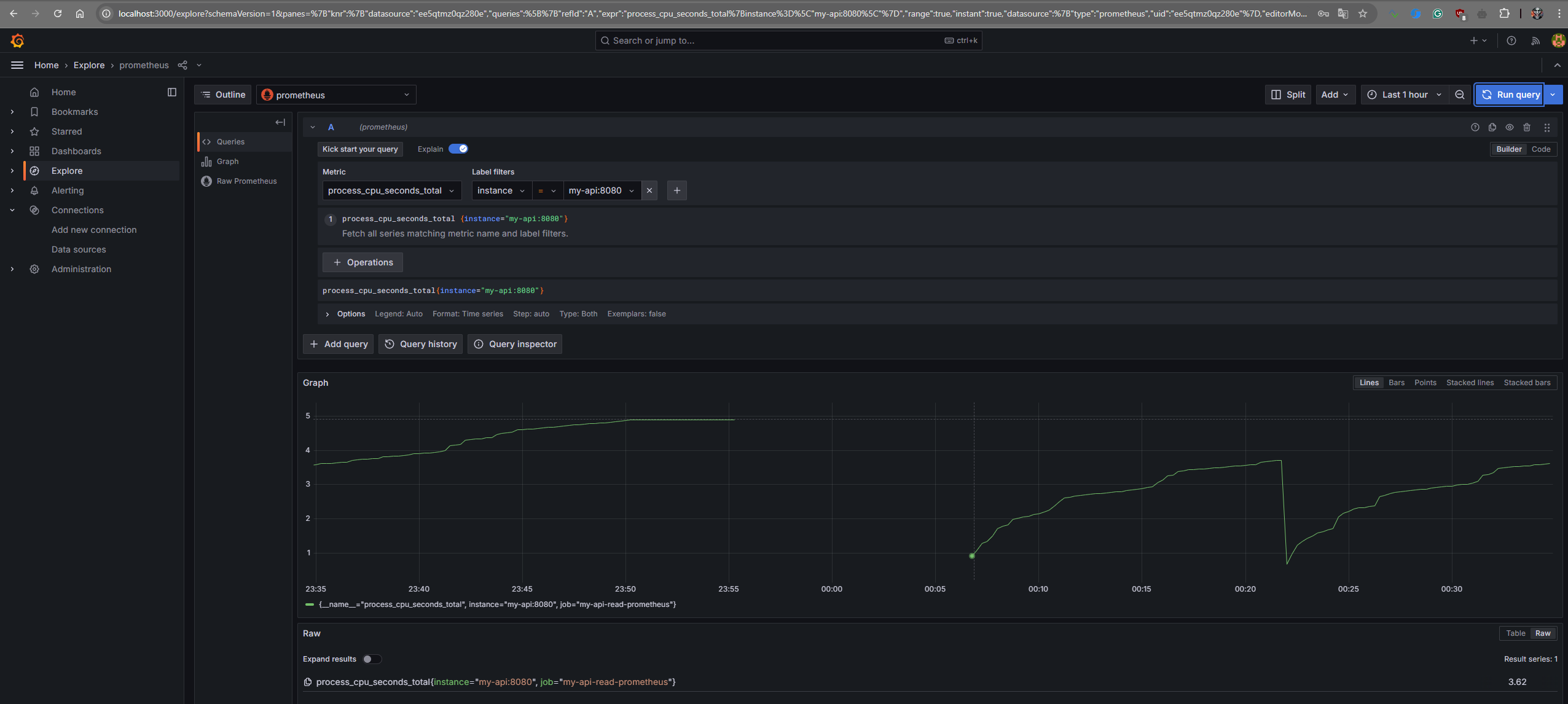




1. Перевіримо дашборд у Grafana. Він має відображати інформацію про час виконання щойно зроблених запитів, згруповану за кодами відповіді:



1. 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](https://github.com/JokerFunny/TRIST/tree/main/LR4).