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

Новиков Д. М.

Перевірив:

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:
   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 MaaaahController, 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:

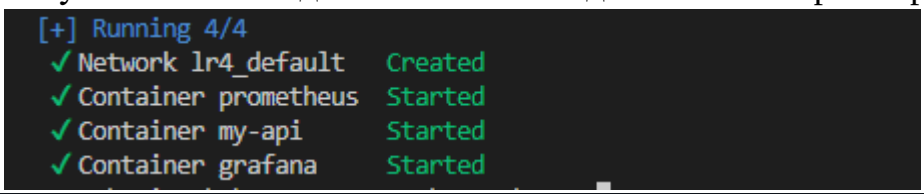
```yaml
services:
  my-api:
    image: ${DOCKER_REGISTRY-}my-api
    container_name: my-api
    ports:
      - 8080:8080
    build:
      context: .
      dockerfile: EFKLoggingApiMetrics/Compose.Dockerfile
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_HTTP_PORTS=8080

  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: {}
```

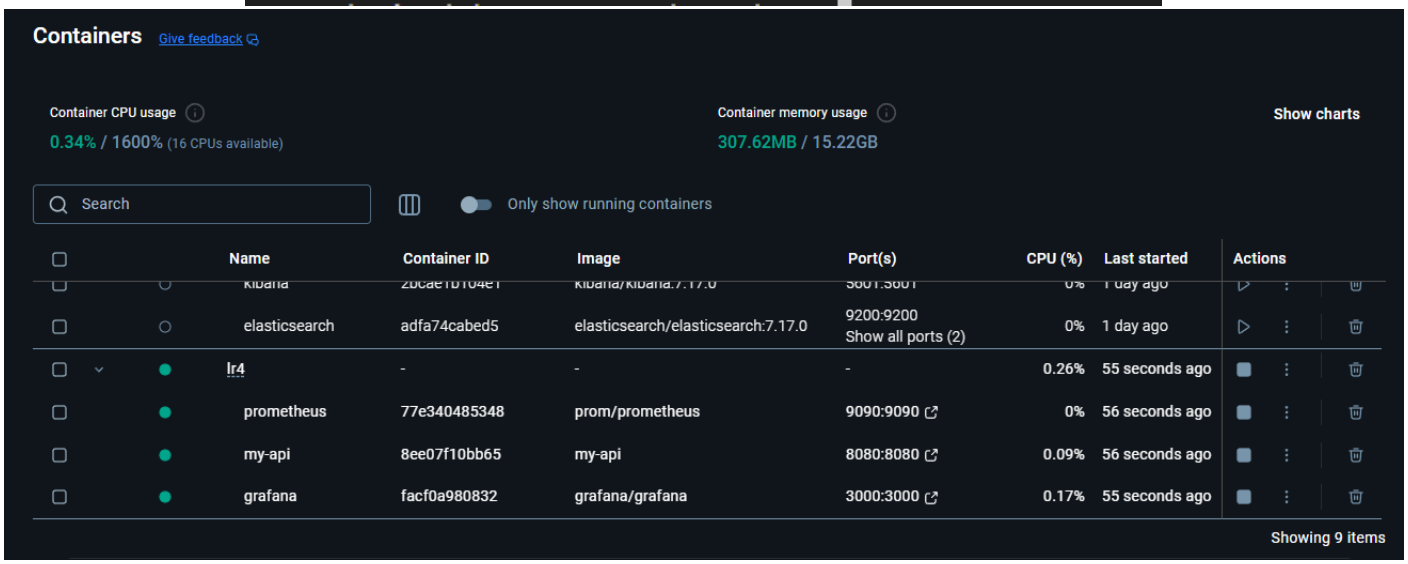
Prometheus.yml:

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

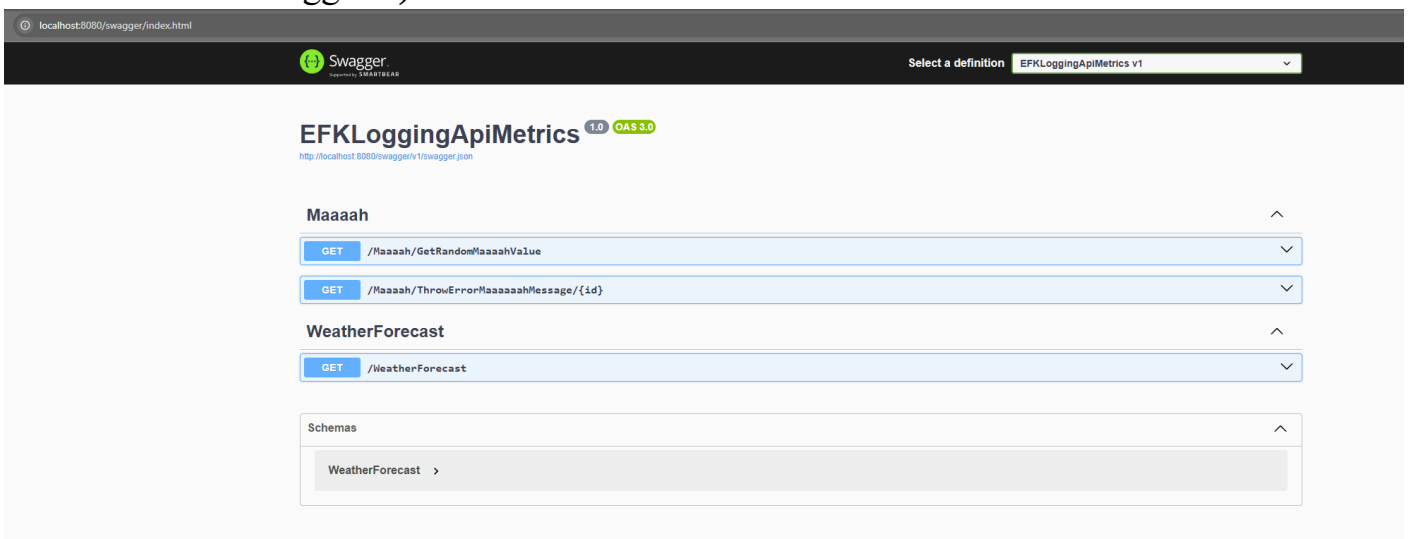
3. Перевіримо працездатність:

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

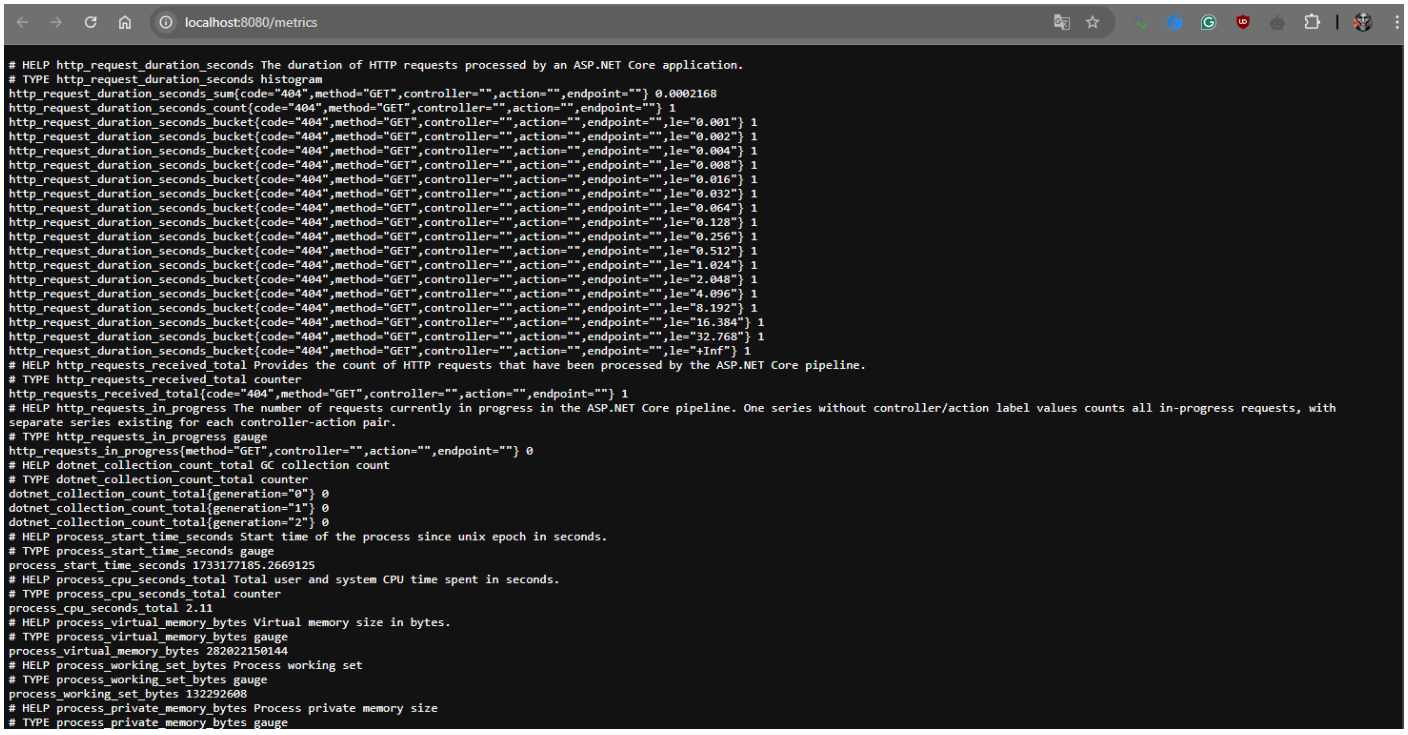




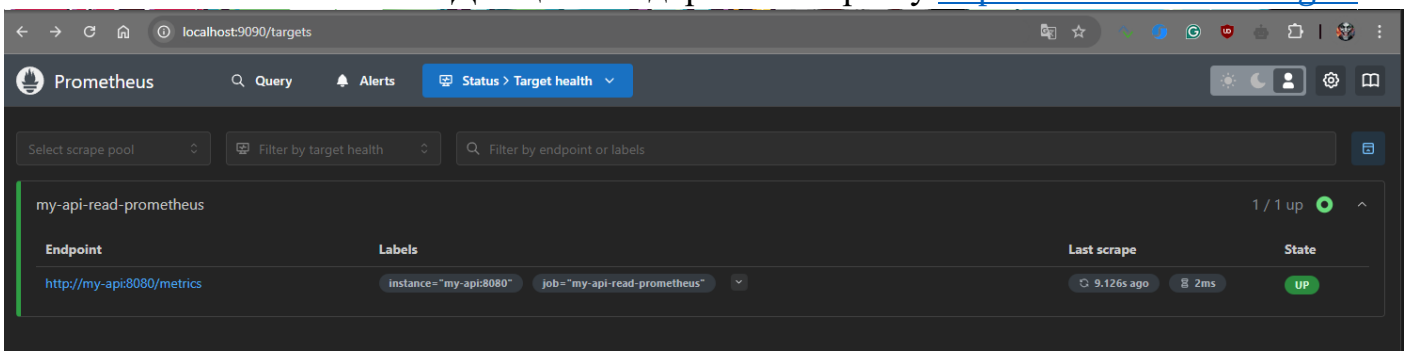
б) Перевіримо роботу застосунку, перейшовши за відповідним посиланням http://localhost:8080/swagger/index.html (для доступу до swagger'а):



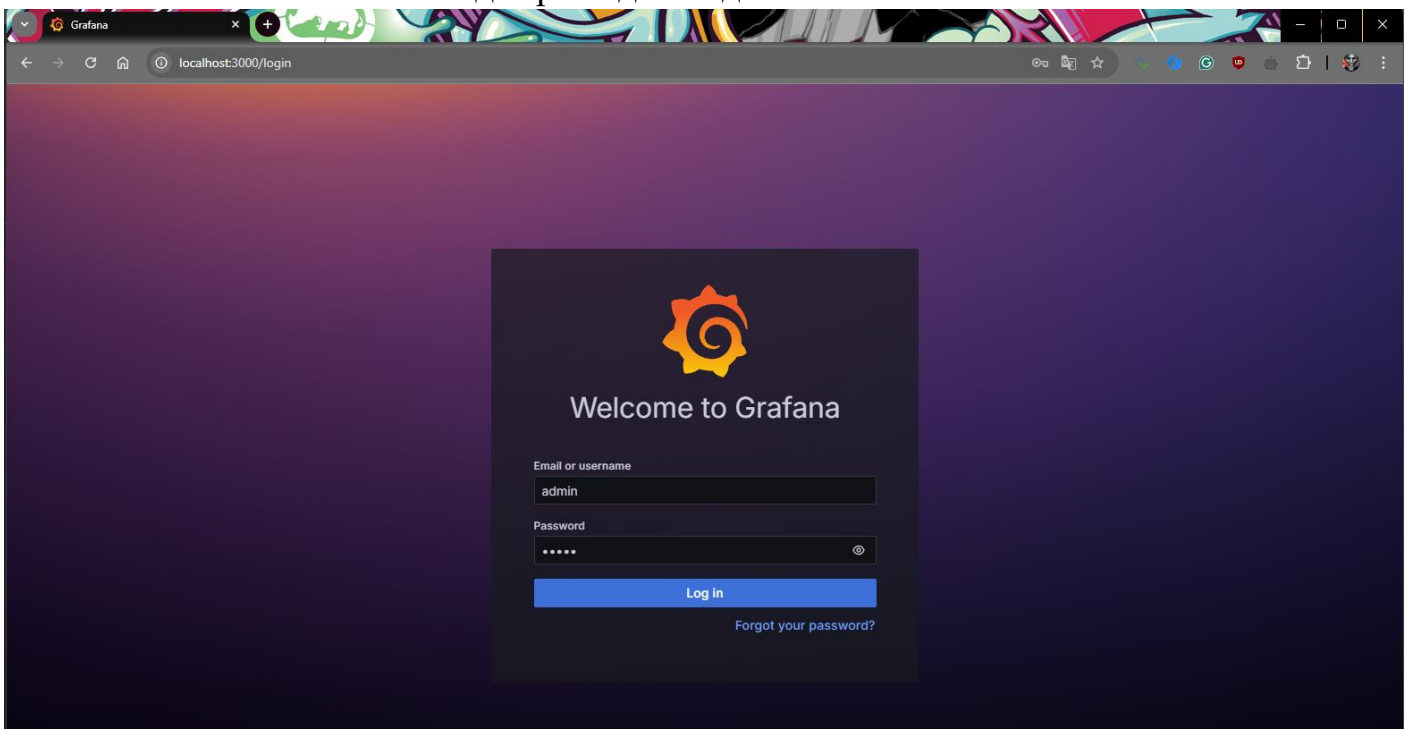
в) Перевіримо, чи застосунок збирає метрики, перейшовши за посиланням http://localhost:8080/metrics (адрес API + /metrics endpoint):

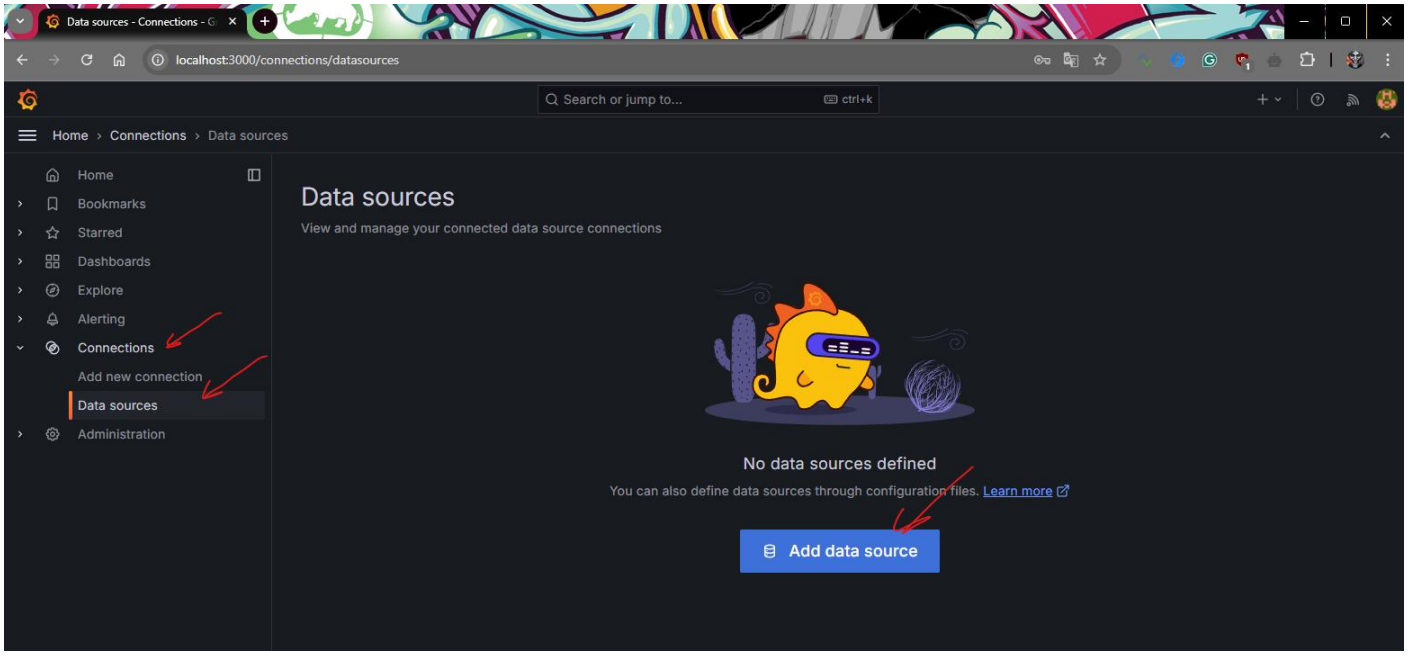


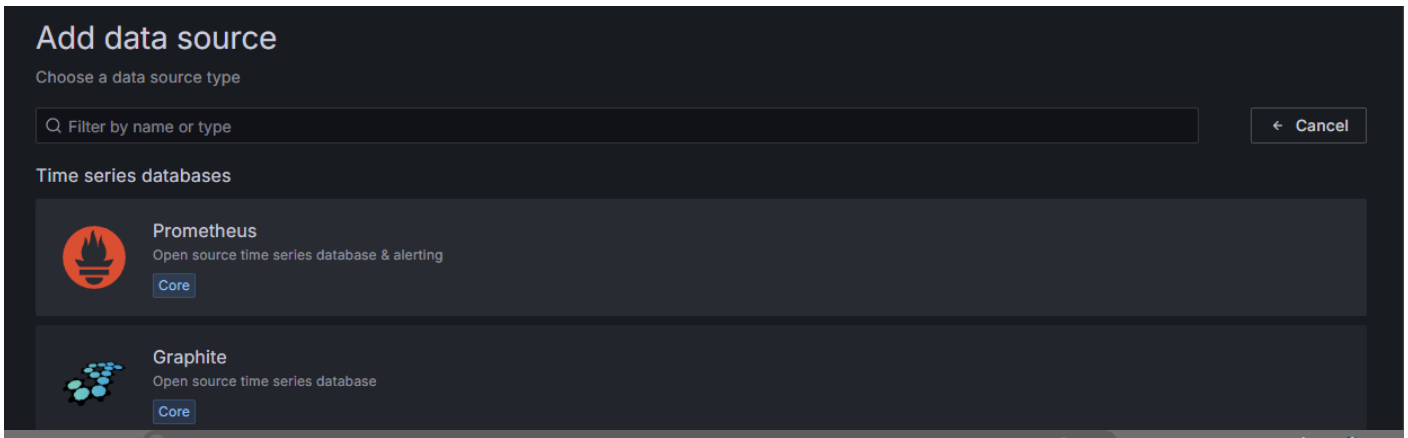
г) Перевіримо, чи Prometheus працює коректно і чи має підключення до нашого API. Для цього відкриємо сторінку http://localhost:9090/targets:

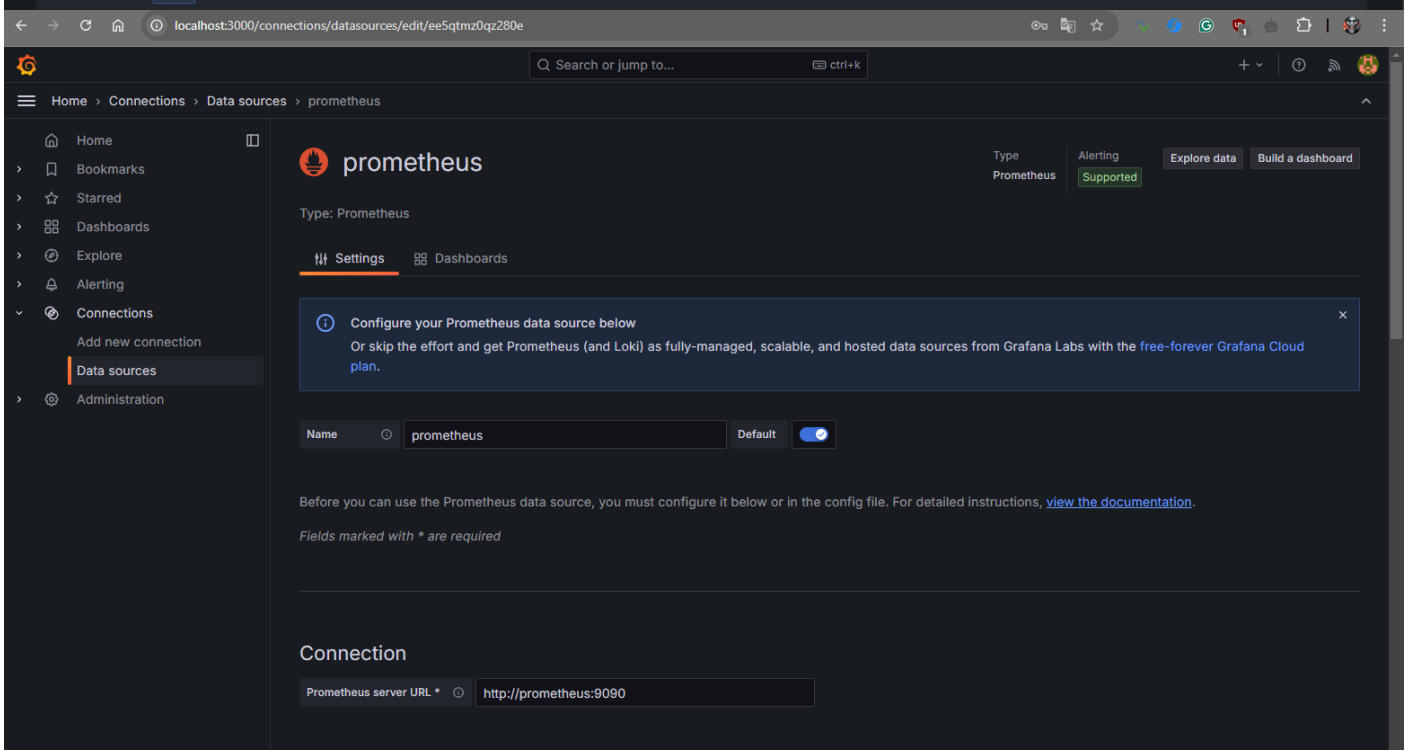


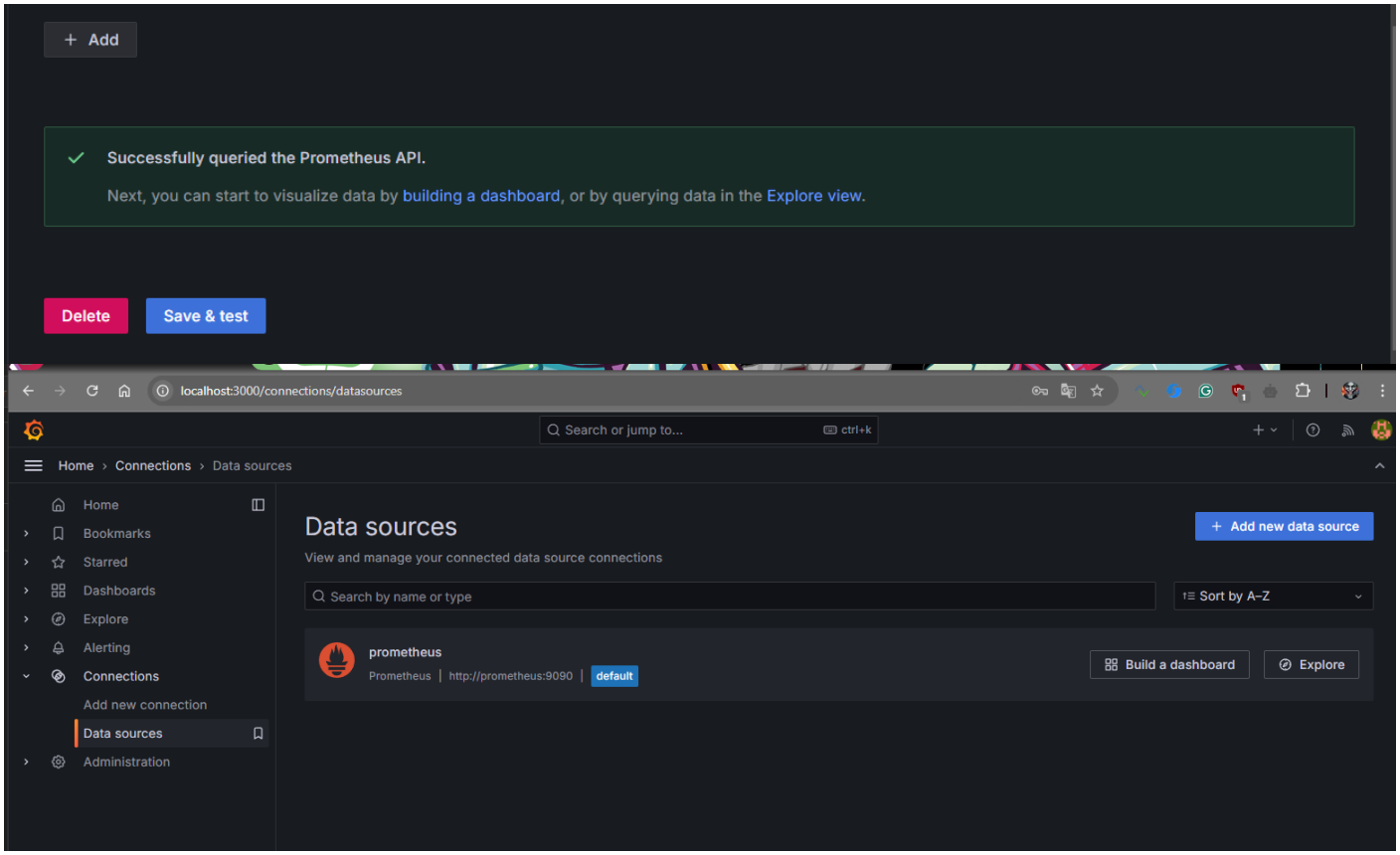
д) Перейдемо до Grafana, переконаємося, що вона працює, та налаштуємо Prometheus як джерело даних для Grafana:



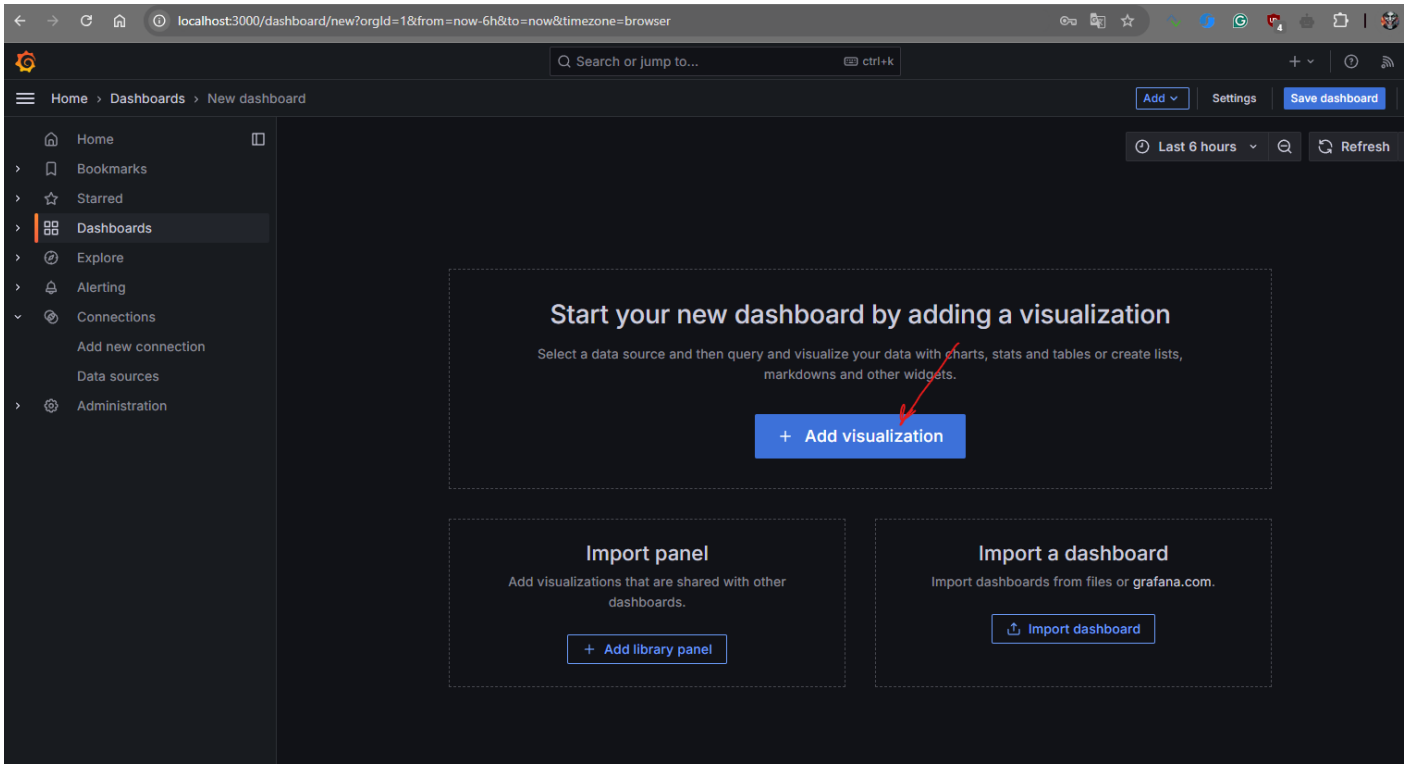

Data sources - Connections -
localhost:3000/connections/datasources
Search or jump to... ctrl+k
Home › Connections › Data sources
Home
Bookmarks
Starred
Dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
Data sources
View and manage your connected data source connections
No data sources defined
You can also define data sources through configuration files. Learn more
Add data source

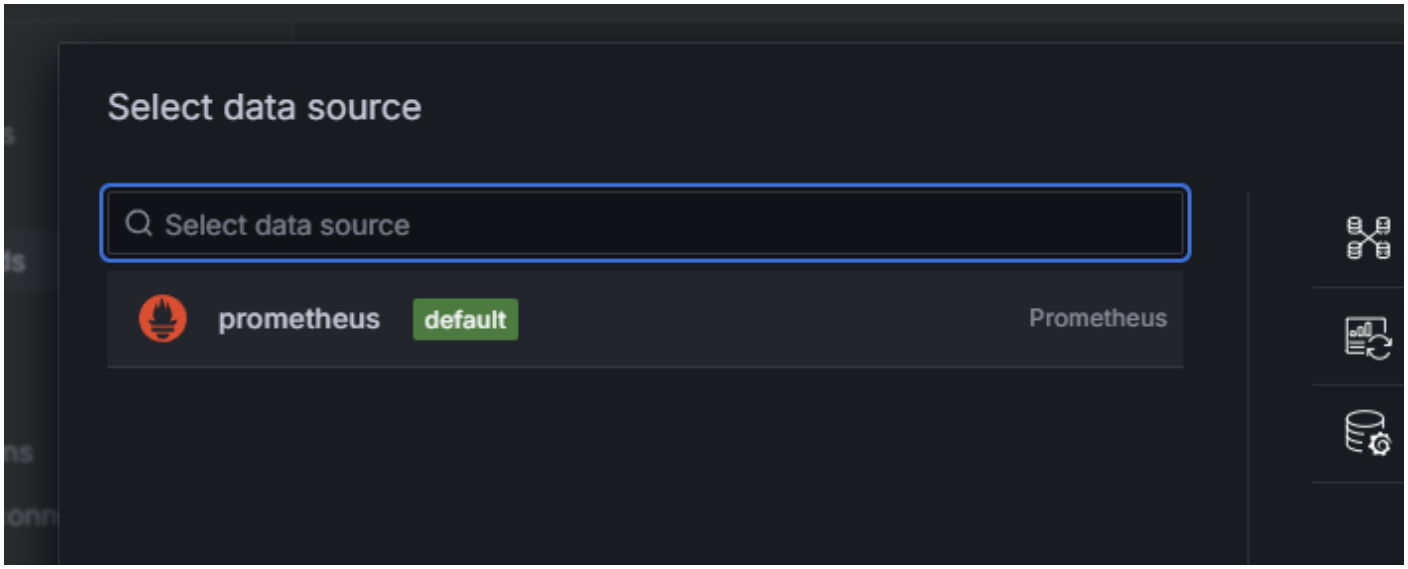

Add data source
Choose a data source type
Filter by name or type
Cancel
Time series databases
Prometheus
Open source time series database & alerting
Core
Graphite
Open source time series database
Core

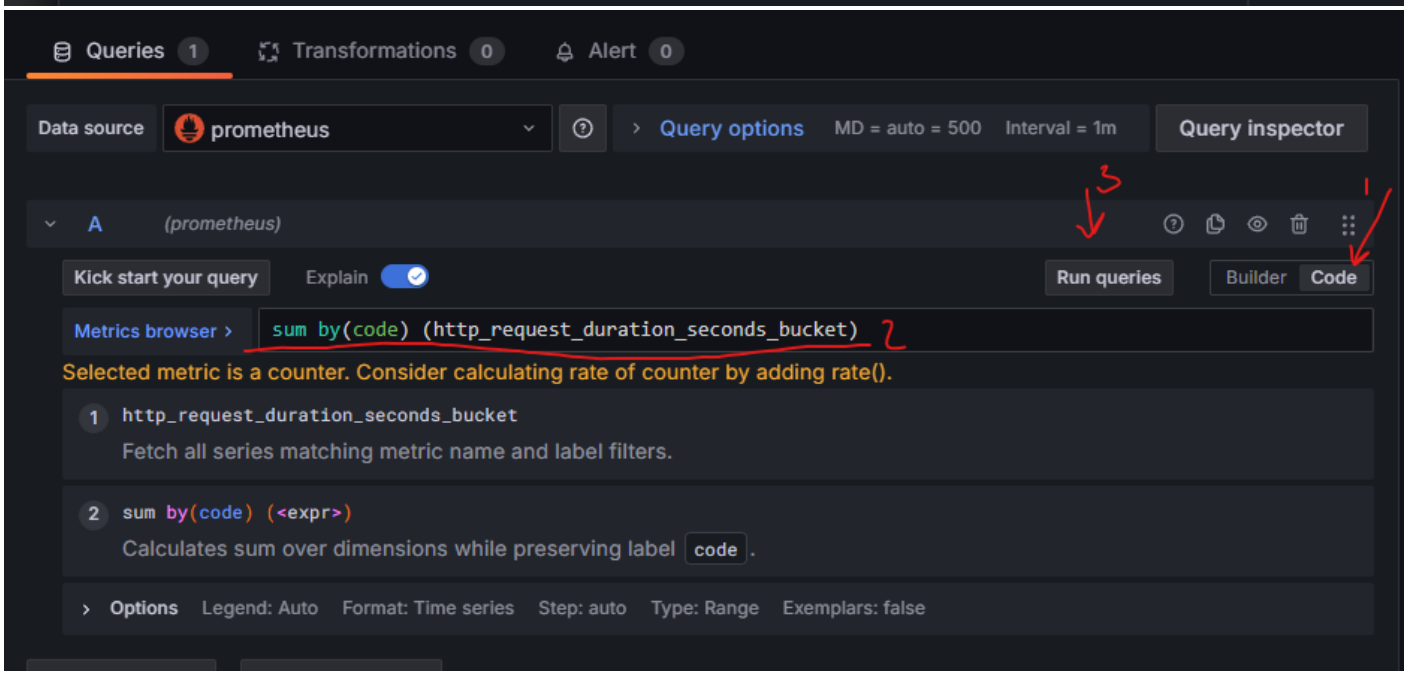

localhost:3000/connections/datasources/edit/ee5qtmz0qz280e
Search or jump to... ctrl+k
Home › Connections › Data sources › prometheus
Home
Bookmarks
Starred
Dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
prometheus
Type
Prometheus
Alerting
Supported
Explore data
Build a dashboard
Type: Prometheus
Settings
Dashboards
Configure your Prometheus data source below
Or skip the effort and get Prometheus (and Loki) as fully-managed, scalable, and hosted data sources from Grafana Labs with the free-forever Grafana Cloud plan.
Name
prometheus
Default
Before you can use the Prometheus data source, you must configure it below or in the config file. For detailed instructions, view the documentation.
Fields marked with * are required
Connection
Prometheus server URL *
http://prometheus:9090



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

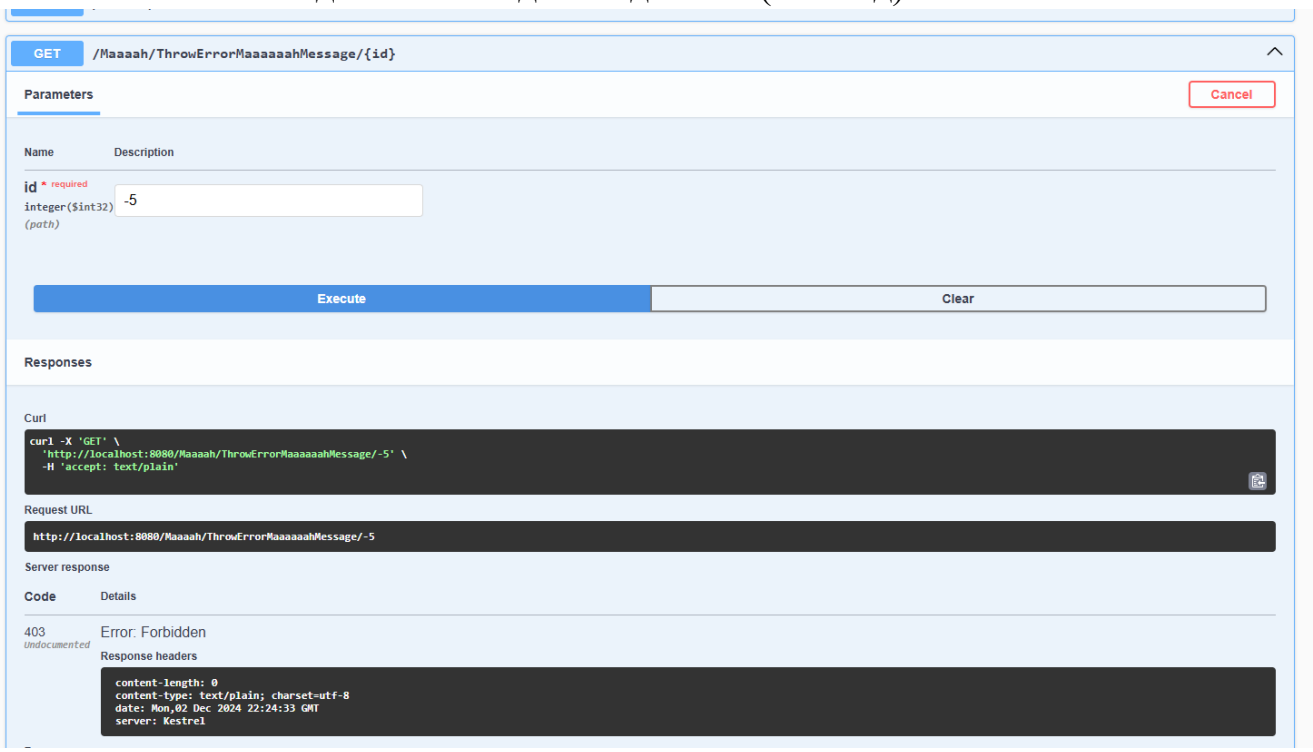


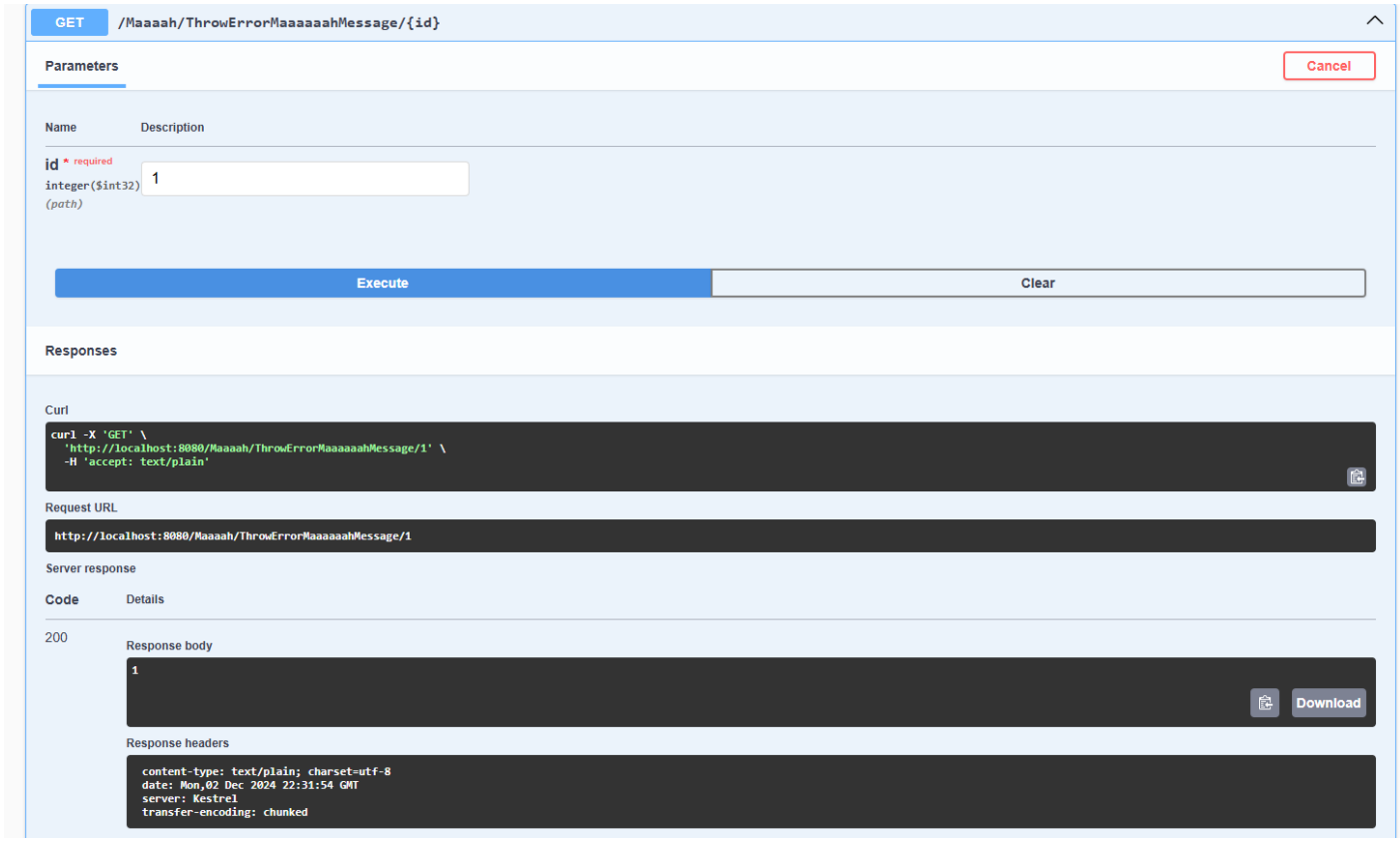




ж) Виконаємо кілька запитів до нашого застосунку:

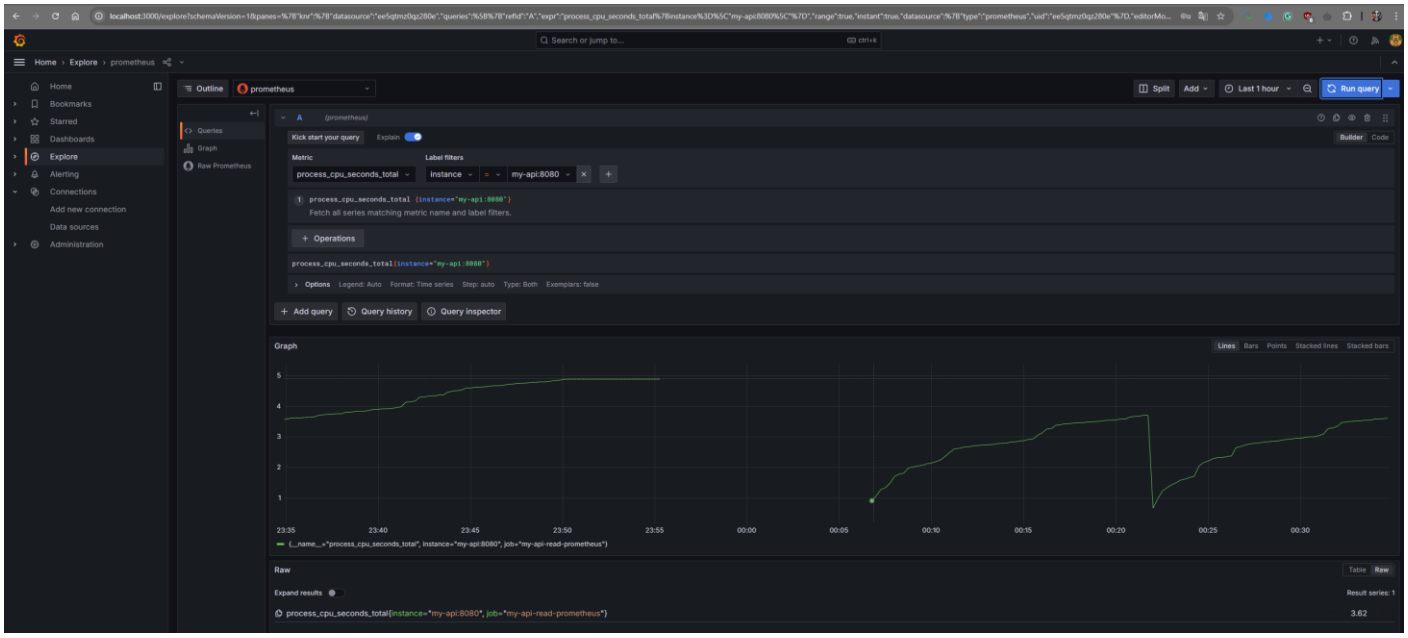
a. Один успішний (код 200);
b. Один із невалідними даними (403 код).





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.