SUPERNOVA

PROMETHEUS & GRAFANA WORKSHOP

WHY MONITOR?

When something goes wrong, how do you know? When something goes right, how do you know it's better?

Monitoring tools give us these insights to see problems early and to be able to prove the effectiveness of improvments.

GETTING STARTED

Create a simple boilerplate dotnet API with:

```
dotnet new webapi --name WeatherAPI
```

PROMETHEUS

WHAT IS IT?

Prometheus is a systems and service monitoring system. It collects metrics from configured targets at given intervals, evaluates rule expressions, displays the results, and can trigger alerts when specified conditions are observed.

LET'S CODE!

Add the prometheus nuget packages to your project

```
dotnet add package prometheus-net --version 7.0.0
```

```
dotnet add package prometheus-net.AspNetCore --version 7.0.0
```

Add the following to your Program.cs file just before `app.Run();`

```
app.UseRouting();

// Capture metrics about all received HTTP requests.
app.UseHttpMetrics();

app.UseEndpoints(endpoints =>
{
    // Enable the /metrics page to export Prometheus metrics.
    // Open http://localhost:5099/metrics to see the metrics.

    endpoints.MapMetrics();
});
```

and remove `app.UseHttpsRedirection();`

Run your app and navigate to `/metrics` full url should be similar to `http://localhost:5076/metrics` you should see (truncated):

```
# 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 process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1678424115.91
# HELP prometheus_net_metric_instances Number of metric instances currently registered across all metric families.
# TYPE prometheus_net_metric_instances gauge
prometheus_net_metric_instances{metric_type="gauge"} 21
```

Next add a AppMetrics.cs file and paste in the following:

```
using Prometheus;

namespace WeatherAPI
{
    public class AppMetrics
    {
        public static readonly Counter WeatherRequestCount = Metrics
            .CreateCounter("weather_request_total", "Number of weather api calls.");

        public static readonly Gauge LastRequestDuration = Metrics
            .CreateGauge("weather_last_request_duration", "Duration of last request.");

        public static readonly Histogram CallDuration = Metrics
            .CreateHistogram("weather_request_duration", "Histogram of weather api call duration.",
            new HistogramConfiguration
            {
                Buckets = new double[] { 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 }
            });
    }
}
```

Go to the `WeatherForecastController.cs` and replace the contents of the `Get()` method with:

```
using (AppMetrics.CallDuration.NewTimer())
{
    var stopWatch = new StopwatchC();
    stopWatch.Start();

    var data = Enumerable.Range(1, 5).Select(index => new WeatherForecast
    {
        Date = DateTime.Now.AddDays(index),
        TemperatureC = Random.Shared.Next(-20, 55),
        Summary = Summaries[Random.Shared.Next(Summaries.Length)]
    })
    .ToArray();

    var random = new Random();
    Thread.Sleep(random.Next(100, 1000));

    AppMetrics.LastRequestDuration.Set(stopWatch.ElapsedMilliseconds);

    return data;
}
```

Run your app, go to the Swagger page and execute GET /WeatherFrorecast 3 times. Then navigate to the `/metrics` pages, you should find:

```
weather_request_total 3

weather_last_request_duration 150

weather_request_duration_sum 0.45643240000000007
weather_request_duration_count 3
weather_request_duration_bucket{le="0.005"} 0
weather_request_duration_bucket{le="0.01"} 0
weather_request_duration_bucket{le="0.025"} 0
weather_request_duration_bucket{le="0.05"} 0
weather_request_duration_bucket{le="0.075"} 0
weather_request_duration_bucket{le="0.1"} 0
weather_request_duration_bucket{le="0.25"} 3
weather_request_duration_bucket{le="0.5"} 3
weather_request_duration_bucket{le="0.75"} 3
weather_request_duration_bucket{le="1"} 3
weather_request_duration_bucket{le="2.5"} 3
weather_request_duration_bucket{le="5"} 3
weather_request_duration_bucket{le="7.5"} 3
weather_request_duration_bucket{le="10"} 3
weather_request_duration_bucket{le="+Inf"} 3
```

You have setup your first prometheus counter, gauge and histrogram which can be used to graph the requests against your API.

GRAFANA

WHAT IS IT?

Grafana provides charts, graphs, and alerts when connected to supported data sources. In this case Prometheus will be our data source.

LET'S TRY IT OUT

Add the docker-compose.yml file to your project and paste in:

```
version: "3.9"
services:
  prometheus:
    image: bitnami/prometheus
    container_name: prometheus
    ports:
      - '9090:9090'
    volumes:
      - ./prometheus.yml:/opt/bitnami/prometheus/conf/prometheus.yml
    network_mode: host

  grafana:
    image: grafana/grafana
    container_name: grafana
    depends_on:
      - prometheus
    ports:
      - '3000:3000'
    network_mode: host
```

Then add another file prometheus.yml and paste in (NOTE: use the port numbers of your app, 5076 was my port locally when running WeatherAPI):

```
global:
  scrape_interval:     5s
  evaluation_interval: 5s
scrape_configs:
  - job_name: 'metrics_collection'
    scheme: 'http'
    static_configs:
      - targets: [
        'localhost:5076',
      ]
```

From the terminal in the same folder as the above file run `docker-compose up -d`

Then navigate to http://localhost:9090 in your browser and run the query `weather_request_total` you will then see the value scraped by Prometheus from your app.

Next navigate to Grafana at http://localhost:3000 and login with admin/admin, skip changing password. Hover over the cog icon for settings, click "Data sources", click "Add data source", select Prometheus and enter http://localhost:9090 into URL, the click "Save & test"

COUNTERS AND GRAPHS

Next hover over the Dashboards icon (four squares), click "New dashboard". Click "Add new panel" switch from "Builder" to "Code" mode and paste in `sum(rate(weather_request_total[1m]) * 60 )` set time range to last 5 minutes, click apply. You can set an auto refresh from the refresh icon, set to 5s. Now execute your API from swagger and see how Grafana shows you the request rate.

GAUGES

Add a New Panel, on the top right change the Visualization from "Time series" to "Gauge" and paste in `weather_last_request_duration` into the PromQL Query. Click "Apply".

HISTOGRAMS AS HEATMAPS

Add a New Panel, on the top right change the Visualization from "Time series" to "Heatmap" and paste in `sum by (le) (increase(weather_request_duration_bucket[1m]))` into the PromQL Query. Click "Apply".