DotNext 2023

Метрики в .NET на примере OpenTelemetry и Prometheus

Пугач Александр

Разработчик, Лаборатория Касперского

kaspersky

Intro

Метрики могут спасти жизнь вашего приложения

Agenda

- Метрики
- OpenTelemetry
- Prometheus
- Application instrumentation
- Альтернативы OpenTelemetry в .NET
- Alerting
- Grafana
- Заключение

Метрики

Метрики - это количественные данные, которые описывают производительность приложения или состояние системы.

Примеры

- RPS (requests per second)
- Latency (задержка)
- Throughput (пропускная способность)
- Error rate (частота ошибок)
- CPU Usage (использование ЦПУ)
- Memory usage (использование памяти)

Необходимость мониторинга приложений

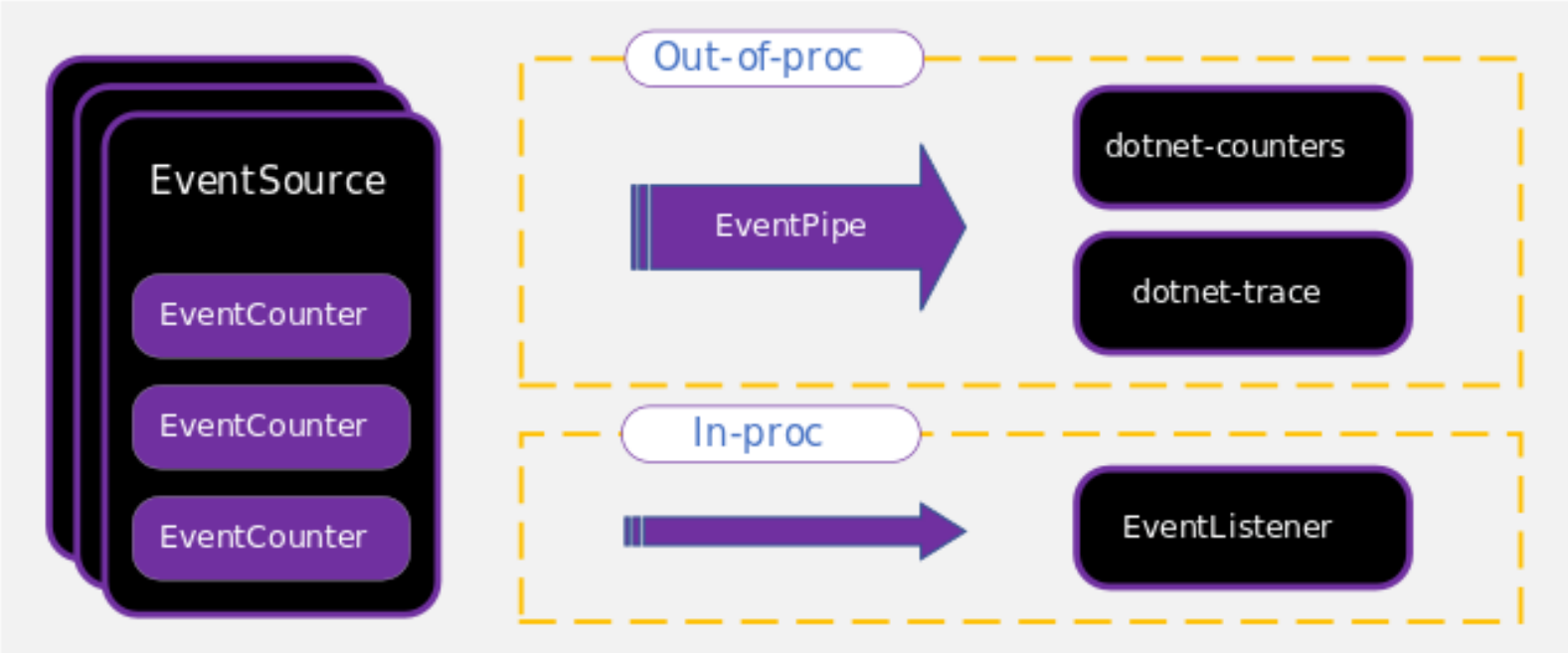
- Выявление и устранение деградаций
- Отслеживание и оптимизация производительности
- Выявление аномалий и повышение надежности и безопасности

.NET Metric APIs

- PerformanceCounter
- EventCounters
- System.Diagnostics.Metrics
- Third-party APIs (AppMetrics, Prometheus)

---

https://learn.microsoft.com/en-us/dotnet/core/diagnostics/compare-metric-apis

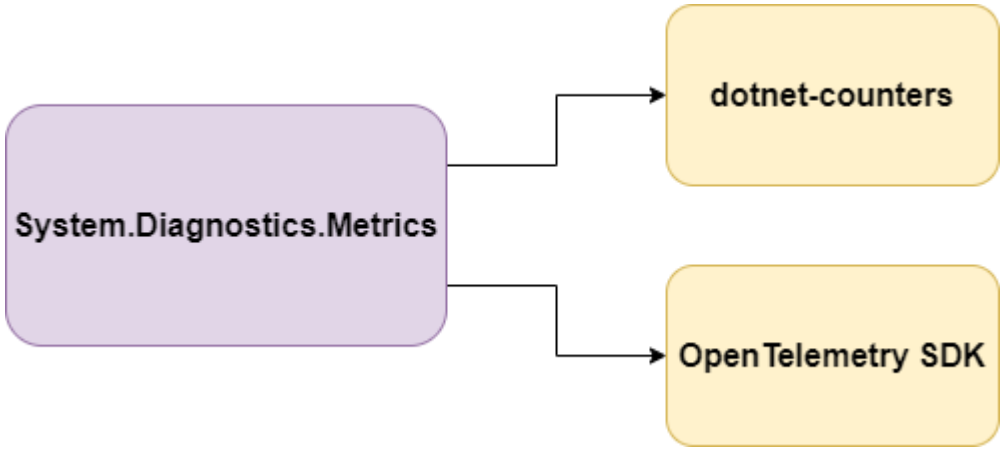


EventCounters

- Lightweight, cross-platform
- .NET 3.1+
- dotnet-counter
- dotnet-monitor
- Not support histograms and percentiles
- Not strongly typed
- No support multi-dimensional metrics

https://learn.microsoft.com/en-us/dotnet/core/diagnostics/compare-metric-apis

Метрики



System.Diagnostics.Metrics

- .NET 6.0
- OpenTelemetry
- dotnet-counter
- Histograms and percentiles
- Strongly typed
- Multi-dimensional metrics

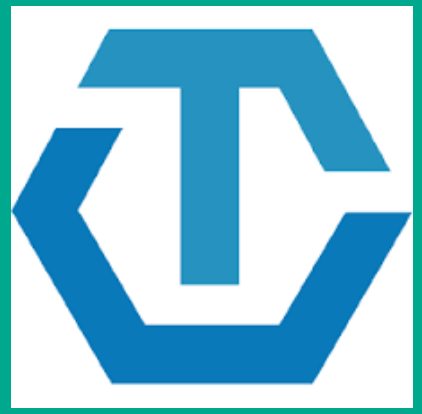
OpenTelemetry

OpenTelemetry is a collection of tools, APIs, and SDKs. Use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) to help you analyze your software's performance and behavior.

https://opentelemetry.io/

OpenTelemetry

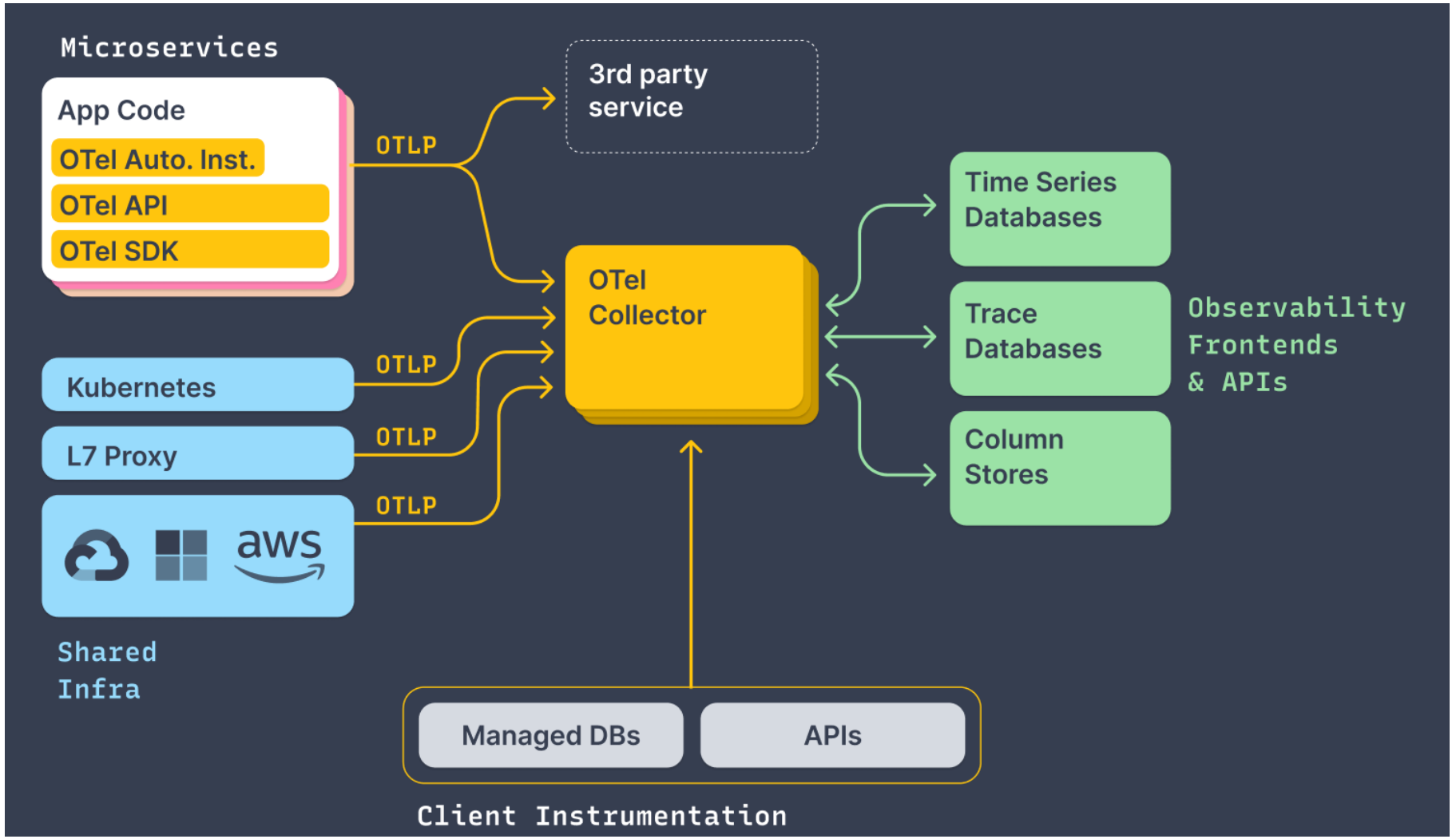
OpenCensus



OpenTracing

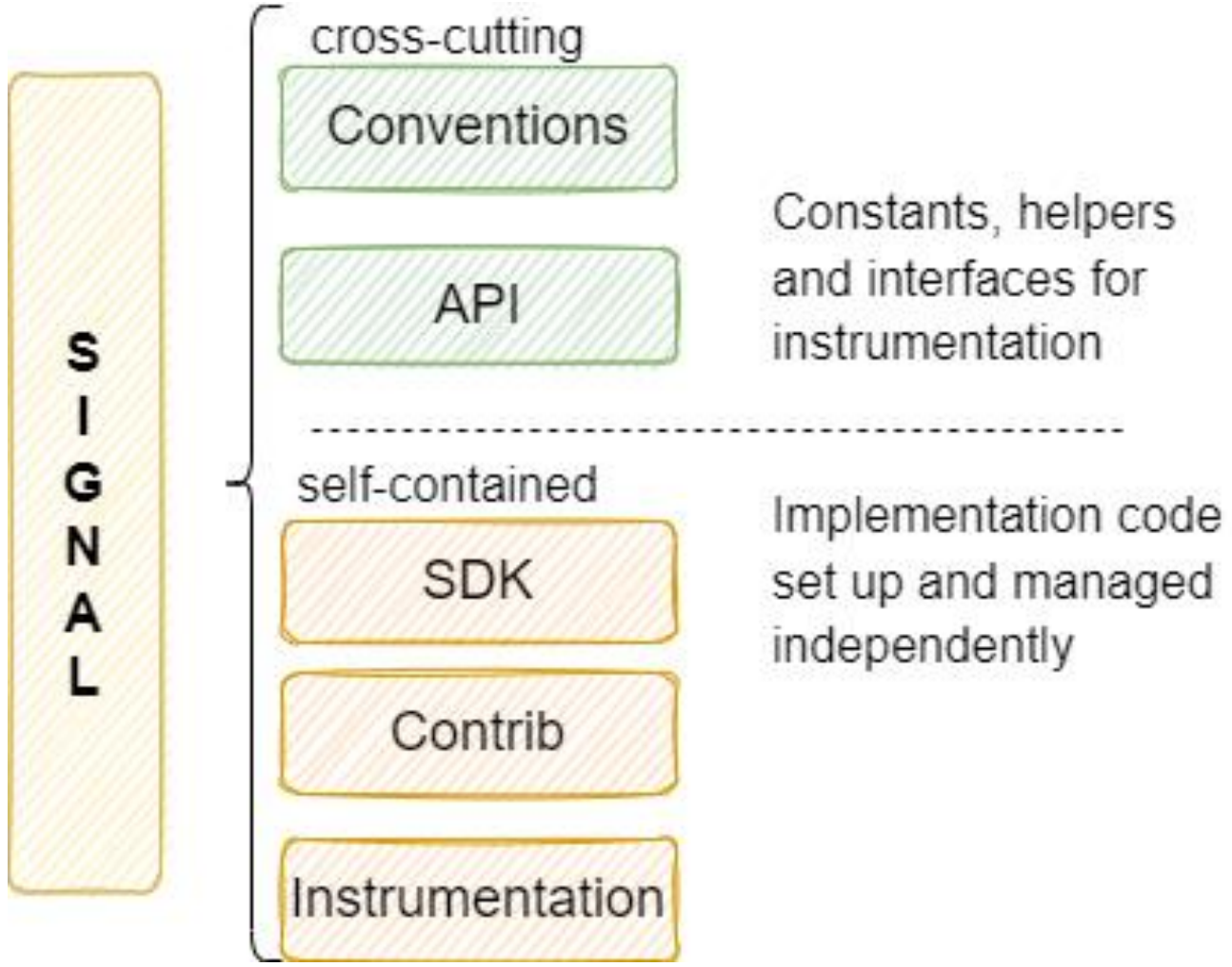
OpenTelemetry

OpenTelemetry



https://opentelemetry.io/docs/

Specification



https://opentelemetry.io/docs/reference/specification/overview/

Project Status

Signal	Status
Logs	Stable*
Metrics	Stable
Traces	Stable

opentelemetry-dotnet Public

The OpenTelemetry .NET Client

logging netcore asp-net-core asp-net distributed-tracing ilogger iloggerprovider

C# Apache-2.0 600 2,345 382 (30 issues need help) 17 Updated 11 hours ago

https://github.com/open-telemetry/opentelemetry-dotnet

System.Diagnostics.Metrics

```
+-- MeterProvider(default)
    |
    +-- Meter(name, version)
        |
        +-- Instrument<Counter, int>(name, unit, description)
            |
            +-- Measurement<int>(value, tags)
```

---

https://github.com/open-telemetry/opentelemetry-specification/blob/main/specification/metrics/api.md

Instruments

Синхронные

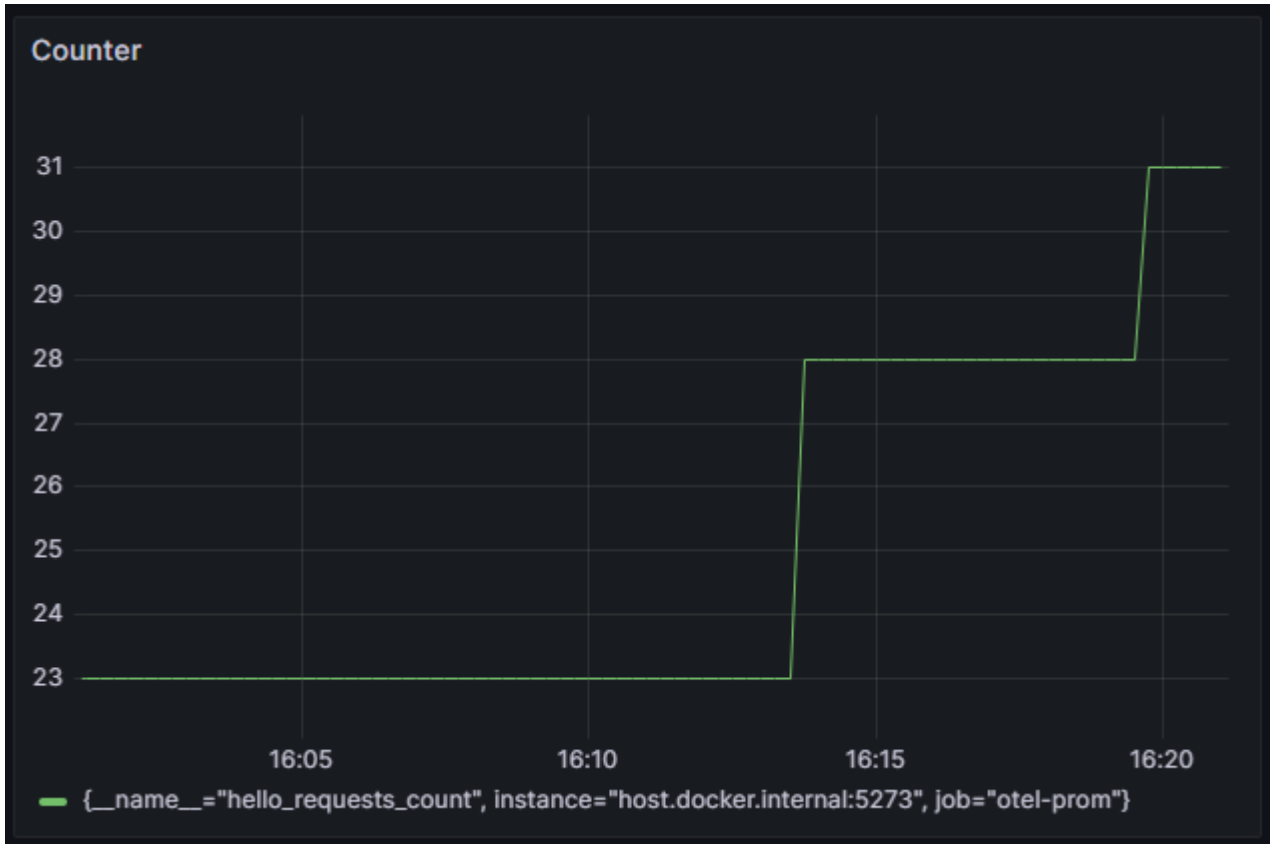
- Counter
- Histogram
- UpDownCounter

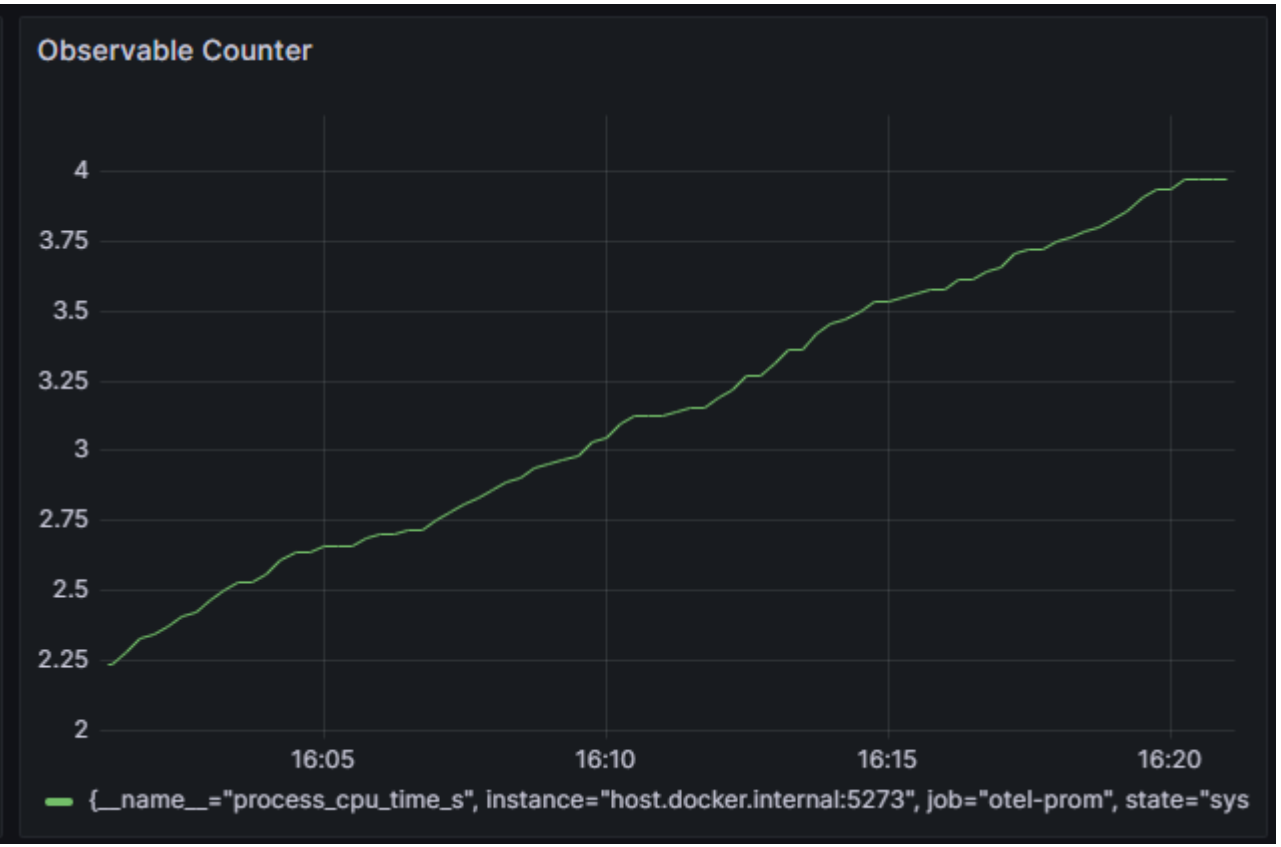
Асинхронные

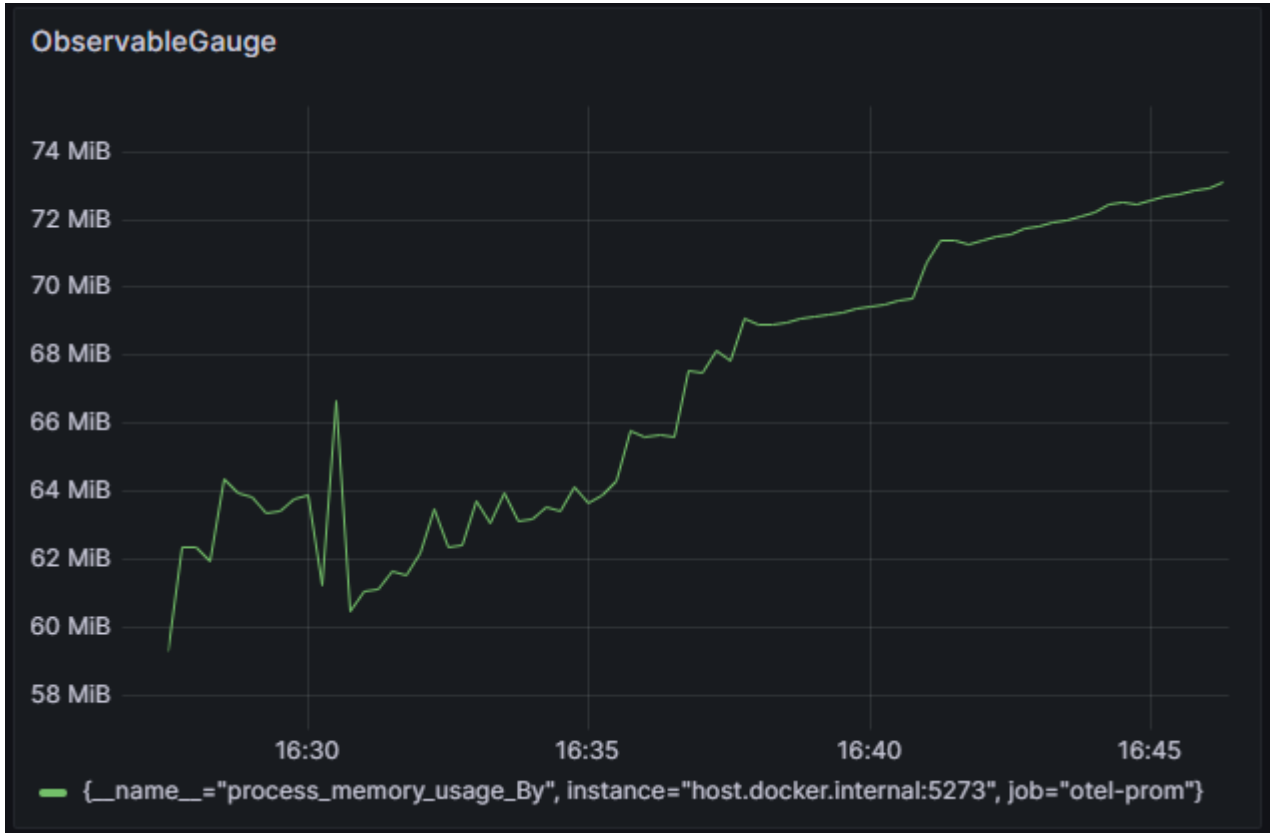
- ObservableCounter
- ObservableGauge
- ObservableUpDownCounter

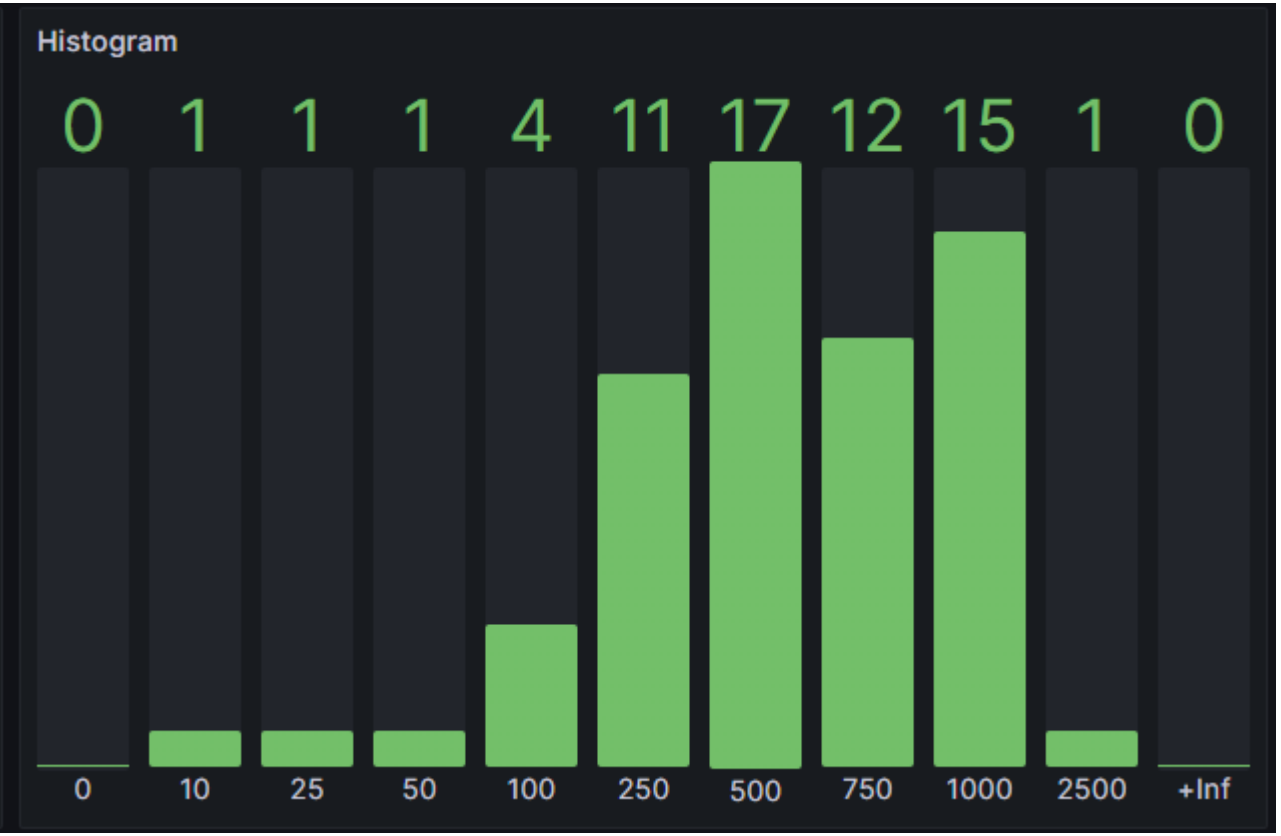
---

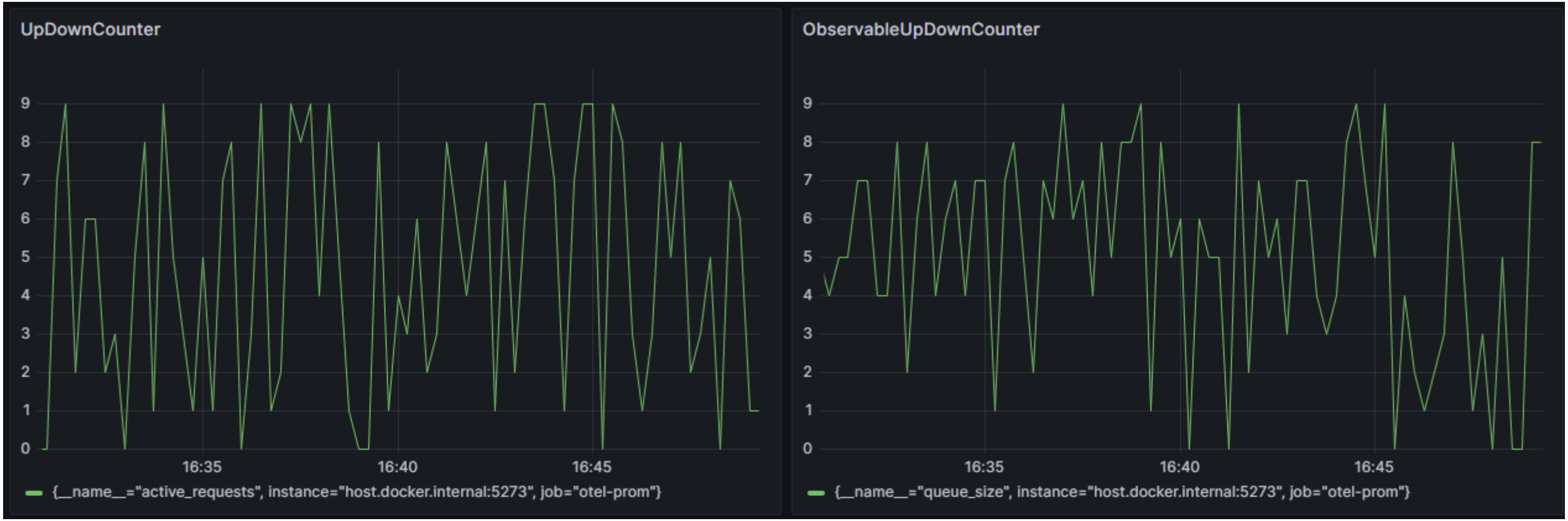
https://github.com/open-telemetry/opentelemetry-specification/blob/main/specification/metrics/api.md


Counter
31
30
29
28
27
26
25
24
23
16:05
16:10
16:15
16:20
{__name__="hello_requests_count", instance="host.docker.internal:5273", job="otel-prom"}


Observable Counter
4
3.75
3.5
3.25
3
2.75
2.5
2.25
2
16:05
16:10
16:15
16:20
{__name__="process_cpu_time_s", instance="host.docker.internal:5273", job="otel-prom", state="sys


ObservableGauge
74 MiB
72 MiB
70 MiB
68 MiB
66 MiB
64 MiB
62 MiB
60 MiB
58 MiB
16:30
16:35
16:40
16:45
{__name__="process_memory_usage_By", instance="host.docker.internal:5273", job="otel-prom"}


Histogram
0 1 1 1 4 11 17 12 15 1 0
0
10
25
50
100
250
500
750
1000
2500
+Inf


UpDownCounter
9
8
7
6
5
4
3
2
1
0
16:35
16:40
16:45
{__name__="active_requests", instance="host.docker.internal:5273", job="otel-prom"}
ObservableUpDownCounter
9
8
7
6
5
4
3
2
1
0
16:35
16:40
16:45
{__name__="queue_size", instance="host.docker.internal:5273", job="otel-prom"}

OpenTelemetry	Prometheus
Counter	Counter
ObservableCounter	
ObservableGauge	Gauge
UpDownCounter	
ObservableUpDownCounter	
Histogram	Histogram

```
var meter = new Meter("MyLibrary", "1.0");
```

```
var meter = new Meter("MyLibrary", "1.0");

var counter = meter.CreateCounter<long>(
    "counter.name",
    unit: "things",
    description: "A count of things");
```

```
var meter = new Meter("MyLibrary", "1.0");

var counter = meter.CreateCounter<long>(
    "counter.name",
    unit: "things",
    description: "A count of things");

using var meterProvider = Sdk.CreateMeterProviderBuilder()
    .AddMeter(meter.Name)
    .AddConsoleExporter()
    .Build();
```

```
var meter = new Meter("MyLibrary", "1.0");

var counter = meter.CreateCounter<long>(
    "counter.name",
    unit: "things",
    description: "A count of things");

using var meterProvider = Sdk.CreateMeterProviderBuilder()
    .AddMeter(meter.Name)
    .AddConsoleExporter()
    .Build();

while (!Console.KeyAvailable)
{
    counter.Add(200, new KeyValuePair<string, object?>("tag2", "value2"));
}
```

```
Export counter.name, A count of things, Unit: things, Meter: MyLibrary/1.0
(2023-04-28T07:30:01.0401475Z, 2023-04-28T07:30:01.9741246Z] tag1: value1 LongSum
Value: 6404800

Export counter.name, A count of things, Unit: things, Meter: MyLibrary/1.0
(2023-04-28T07:30:01.0401475Z, 2023-04-28T07:30:02.9681482Z] tag1: value1 LongSum
Value: 14565200

Export counter.name, A count of things, Unit: things, Meter: MyLibrary/1.0
(2023-04-28T07:30:01.0401475Z, 2023-04-28T07:30:03.9684002Z] tag1: value1 LongSum
Value: 22780800
```

```
var observableGauge = meter.CreateObservableGauge(
    "observable.gauge.name",
    () => new Measurement<long>(
        Random.Shared.Next(),
        new KeyValuePair<string, object?>("tag3", "value3")),
    unit: "unit",
    description: "Random.Shared.Next()");
```

```
Export observable.gauge.name, Random.Shared.Next(), Unit: unit, Meter: MyLibrary/1.0
(2023-04-28T07:32:38.2508811Z, 2023-04-28T07:32:39.2140649Z] tag3: value3 LongGauge
Value: 1036084292

Export observable.gauge.name, Random.Shared.Next(), Unit: unit, Meter: MyLibrary/1.0
(2023-04-28T07:32:38.2508811Z, 2023-04-28T07:32:40.1875849Z] tag3: value3 LongGauge
Value: 381837055

Export observable.gauge.name, Random.Shared.Next(), Unit: unit, Meter: MyLibrary/1.0
(2023-04-28T07:32:38.2508811Z, 2023-04-28T07:32:40.2015744Z] tag3: value3 LongGauge
Value: 1684709289
```

```
var histogram = meter.CreateHistogram<long>(
    "histogram.name",
    unit: "ms",
    description: "Random.Shared.Next(0, 1000)");

while (!Console.KeyAvailable)
{
    histogram.Record(
        Random.Shared.Next(0, 1000),
        new KeyValuePair<string, object?>("tag4", "value4"));
}
```

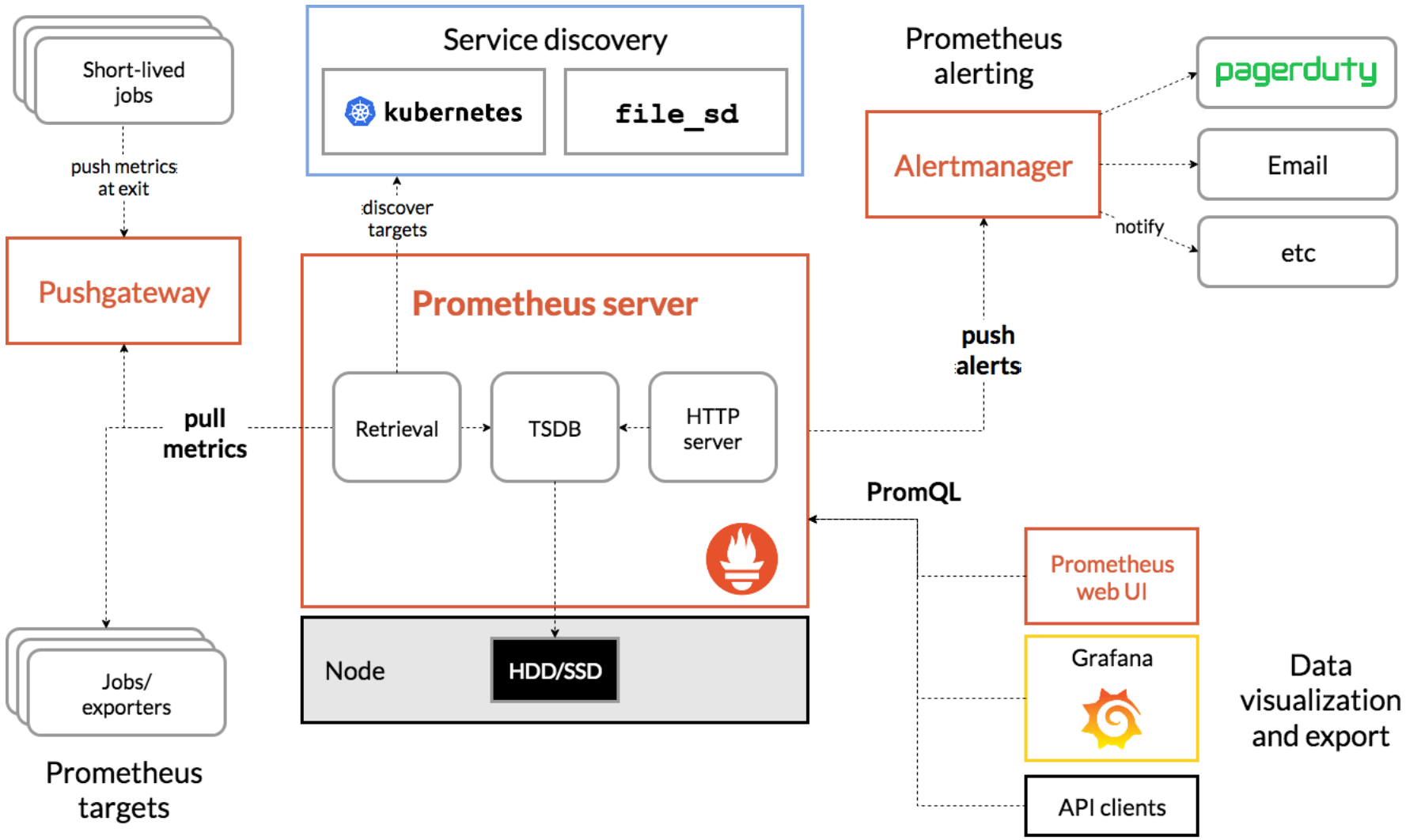
```
Export histogram.name, Random.Shared.Next(0, 1000), Unit: ms, Meter: MyLibrary/1.0
(2023-04-28T07:40:28.0501304Z, 2023-04-28T07:40:36.0014056Z] tag4: value4 Histogram
Value: Sum: 158902284 Count: 317671
(-Infinity,0]:341
(0,5]:1591
(0,10]:1599
(0,25]:4661
(0,50]:8045
(0,75]:7913
(0,100]:7700
(0,250]:47665
(0,500]:79485
(0,1000]:158671
(0,+Infinity]:0
```

Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit collects and stores its metrics as time series data

https://prometheus.io/

Prometheus



https://prometheus.io/docs/introduction/overview/

Почему Prometheus?

- Простая установка и настройка
- Гибкий язык запросов PromQL
- Масштабируемость (Thanos, Victoria Metrics, Cortex)
- Открытый исходный код
- Широкая поддержка
- Интеграция с Kubernetes

```
using var meterProvider = Sdk.CreateMeterProviderBuilder()
    .AddMeter(meter.Name)
    .AddPrometheusExporter(opt =>
    {
        opt.StartHttpListener = true;
        opt.HttpListenerPrefixes = new[] { $"http://localhost:5000/" };
        opt.ScrapeEndpointPath = "/metrics";
    })
    .Build();
```

dotnet add package OpenTelemetry.Exporter.Prometheus.HttpListener -v 1.5.0-rc.1

```
# HELP counter_name_things A count of things
# TYPE counter_name_things counter
counter_name_things{tag1="value1"} 223334800 1682677498910

# HELP observable_gauge_name_unit Random.Shared.Next()
# TYPE observable_gauge_name_unit gauge
observable_gauge_name_unit{tag3="value3"} 790047966 1682677498911
```

http://localhost:5000/metrics/

```
# HELP histogram_name_ms Random.Shared.Next(0, 1000)
# TYPE histogram_name_ms histogram
histogram_name_ms_bucket{tag4="value4",le="0"} 210 1682677498911
histogram_name_ms_bucket{tag4="value4",le="5"} 1303 1682677498911
histogram_name_ms_bucket{tag4="value4",le="10"} 2415 1682677498911
histogram_name_ms_bucket{tag4="value4",le="25"} 5699 1682677498911
histogram_name_ms_bucket{tag4="value4",le="50"} 11185 1682677498911
histogram_name_ms_bucket{tag4="value4",le="75"} 16650 1682677498911
histogram_name_ms_bucket{tag4="value4",le="100"} 22222 1682677498911
histogram_name_ms_bucket{tag4="value4",le="250"} 55272 1682677498911
histogram_name_ms_bucket{tag4="value4",le="500"} 110993 1682677498911
histogram_name_ms_bucket{tag4="value4",le="1000"} 222488 1682677498911
histogram_name_ms_bucket{tag4="value4",le="+Inf"} 222488 1682677498911
histogram_name_ms_sum{tag4="value4"} 111484579 1682677498911
histogram_name_ms_count{tag4="value4"} 222488 1682677498911
```

```
var builder = WebApplication.CreateBuilder(args);

var meter = new Meter("MyLibrary", "1.0");
builder.Services.AddOpenTelemetry()
    .WithMetrics(builder =>
    {
        builder
            .AddMeter(meter.Name)
            .AddPrometheusExporter();
    });

var app = builder.Build();
```

dotnet add package OpenTelemetry.Extensions.Hosting -v 1.5.1
dotnet add package OpenTelemetry.Exporter.Prometheus.AspNetCore 1.5.0-rc.1

```
var counter = meter.CreateCounter<long>(
    "hello.requests.count",
    description: "The number of hello requests");
app.MapGet("/hello", async () =>
{
    counter.Add(1);
    await Task.Delay(Random.Shared.Next(1000));
    return "Hello, World!";
});

app.UseOpenTelemetryPrometheusScrapingEndpoint("metrics");

app.Run();
```

```
# TYPE hello_requests_count counter
# HELP hello_requests_count The number of hello requests
hello_requests_count 4 1682680813723

# EOF
```

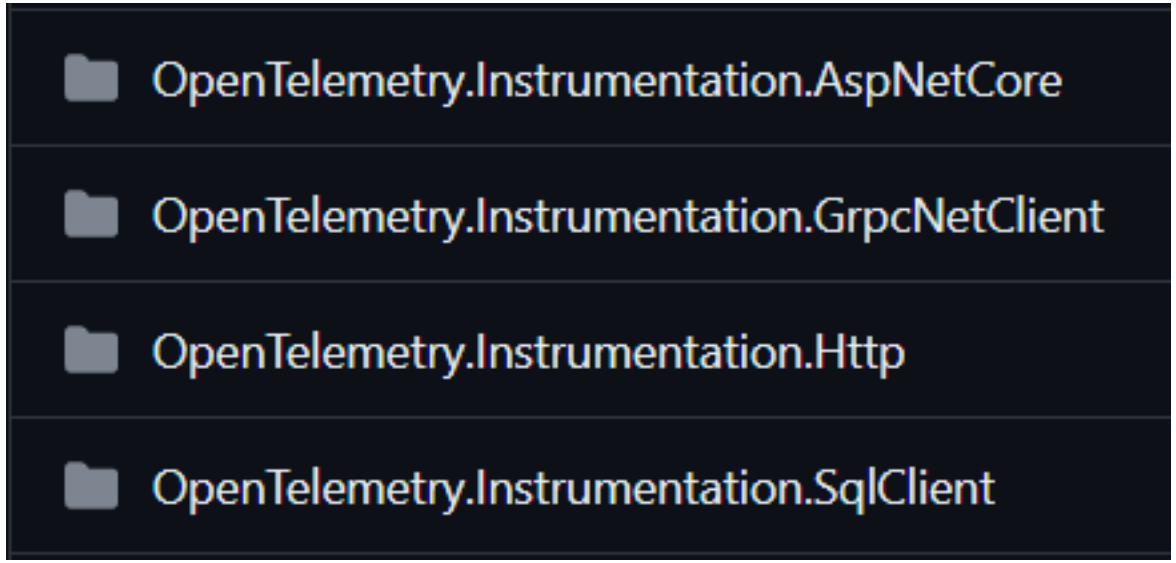
http://localhost:5000/metrics/

Custom Instruments

- opentelemetry-dotnet
- opentelemetry-dotnet-contrib
- Third-party libraries
- Custom instruments

Custom instruments

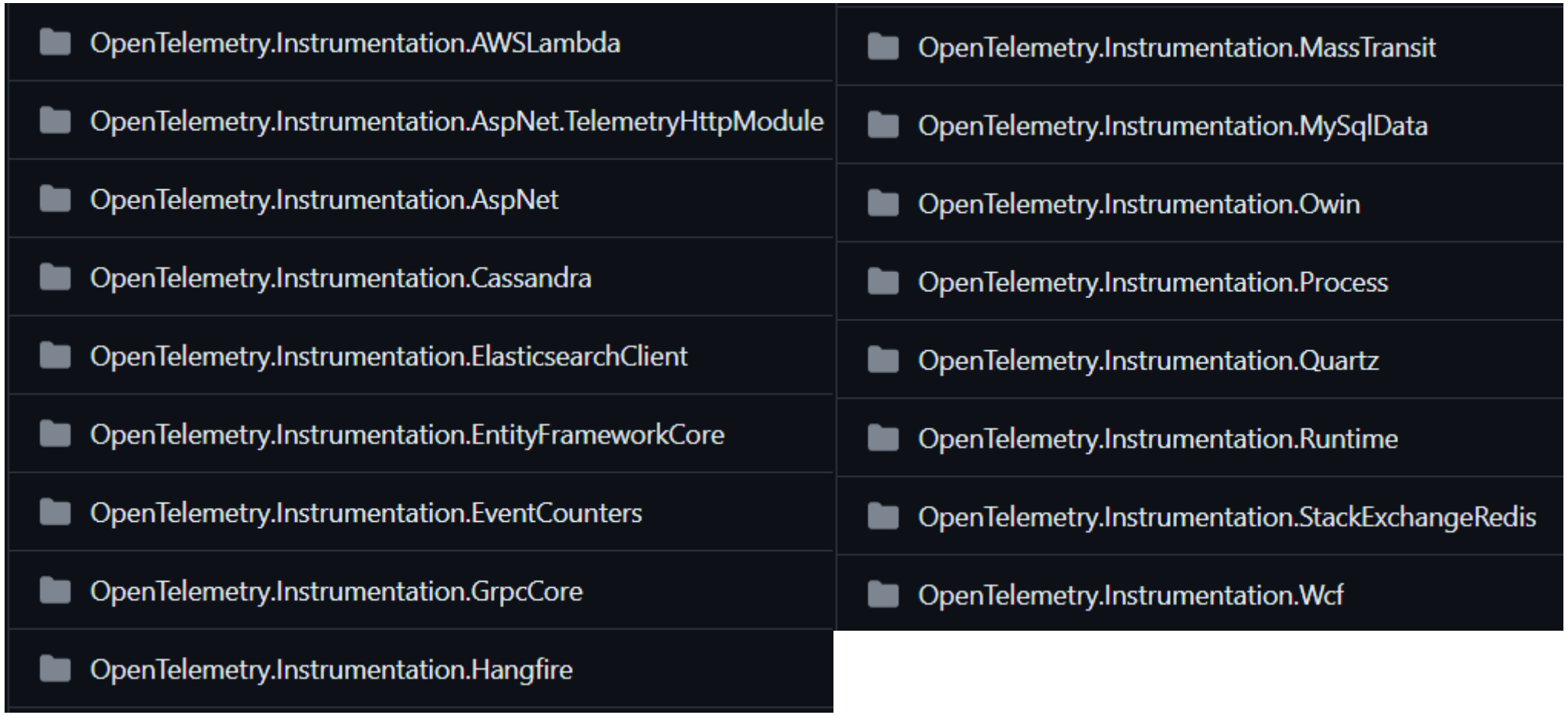
opentelemetry-dotnet



https://github.com/open-telemetry/opentelemetry-dotnet/tree/main/src

Custom instruments

opentelemetry-dotnet-contrib



https://github.com/open-telemetry/opentelemetry-dotnet-contrib/tree/main/src

DiagnosticSource and DiagnosticListener

System.Diagnostics.DiagnosticSource

```
internal AspNetCoreMetrics(AspNetCoreMetricsInstrumentationOptions options)
{
    Guard.ThrowIfNull(options);
    this.meter = new Meter(InstrumentationName, InstrumentationVersion);
    var metricsListener = new HttpInMetricsListener("Microsoft.AspNetCore", this.meter, options);
    this.diagnosticSourceSubscriber = new DiagnosticSourceSubscriber(metricsListener, this.isEnabled);
    this.diagnosticSourceSubscriber.Subscribe();
}
```

https://github.com/open-telemetry/opentelemetry-dotnet/tree/main/src/OpenTelemetry.Instrumentation.AspNetCore

DEMO

Custom instruments


```
public class UpTimeService
{
    private static readonly DateTime StartTime;

    0 references
    static UpTimeService()
    {
        using var process = Process.GetCurrentProcess();
        StartTime = process.StartTime;
    }

    1 reference
    public static TimeSpan UpTime => DateTime.Now - StartTime;
}
```

Custom instruments


```
internal sealed class UpTimeMetrics : IDisposable
{
    internal static readonly string InstrumentationName = typeof(UpTimeMetrics).FullName!;
    internal static readonly string? InstrumentationVersion
        = typeof(UpTimeMetrics).Assembly.GetName().Version?.ToString();

    private readonly Meter _meter;

    1 reference
    public UpTimeMetrics()
    {
        _meter = new Meter(InstrumentationName, InstrumentationVersion);
        var _ = _meter.CreateObservableGauge(
            "application.uptime",
            () => (long)UpTimeService.UpTime.TotalMilliseconds,
            unit: "ms",
            description: "Milliseconds elapsed since application startup");
    }

    0 references
    public void Dispose()
    {
        _meter.Dispose();
    }
}
```

Custom instruments


```
public static class MeterProviderBuilderExtensions
{
    1 reference
    public static MeterProviderBuilder AddUpTimeInstrumentation(
        this MeterProviderBuilder builder)
    {
        builder.AddMeter(UpTimeMetrics.InstrumentationName);
        builder.AddInstrumentation(new UpTimeMetrics());

        return builder;
    }
}
```

Custom instruments


```
# TYPE application_uptime_ms gauge
# UNIT application_uptime_ms ms
# HELP application_uptime_ms Milliseconds elapsed since application startup
application_uptime_ms 12896 1683759414333
```

http://localhost:5000/metrics/

Application instrumentation

- Прямые вызовы
- Декораторы

Application instrumentation


```
public class OtelMetrics : IDisposable
{
    public static readonly string MeterName = typeof(OtelMetrics).FullName!;
    private readonly Meter _meter;

    0 references
    public OtelMetrics()
    {
        var version = typeof(OtelMetrics).Assembly
            .GetName().Version?.ToString();
        _meter = new Meter(MeterName, version);

        HandlerSuccessCounter = _meter.CreateCounter<long>(
            "handler.success", "count", "The number of handler success execution");

        HandlerFailCounter = _meter.CreateCounter<long>(
            "handler.fail", "count", "The number of handler fail execution");

        HandlerLatencyHistogram = _meter.CreateHistogram<long>(
            "handler.latency", "ms", "Message latency");

        HandlerDurationGauge = new Gauge<long>(
            _meter, "handler.duration", "ms", "Handle duration milliseconds");
```

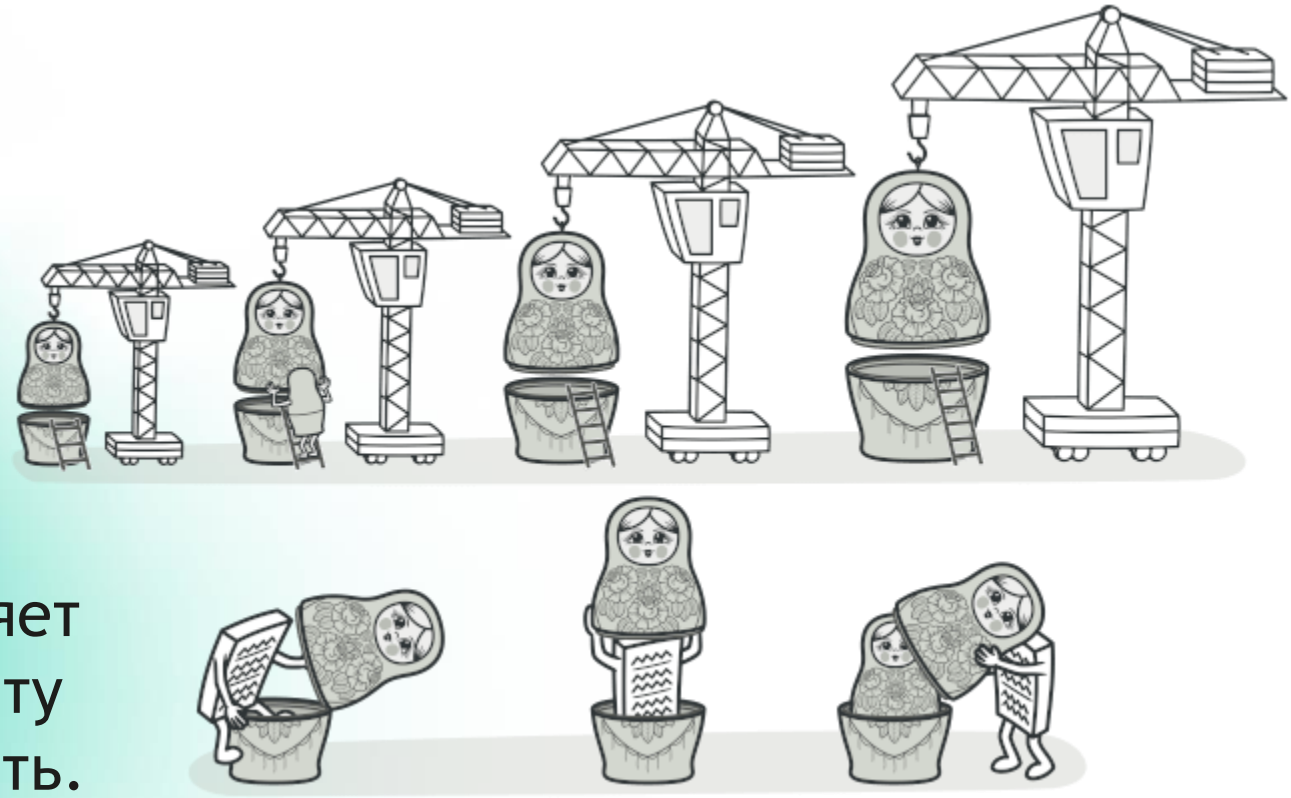
Application instrumentation


```
public MessageHandlerWithMetrics(IRepository<TEntity> repository, IMapper mapper,
    ILogger<MessageHandlerWithMetrics<TMessage, TEntity>> logger, OtelMetrics otelMetrics)
{
    _repository = repository;
    _mapper = mapper;
    _logger = logger;
    _otelMetrics = otelMetrics;
    _tag = new KeyValuePair<string, object?>("message", typeof(TMessage).Name);

    _otelMetrics.HandlerSuccessCounter.Add(0, _tag);
    _otelMetrics.HandlerFailCounter.Add(0, _tag);
}
```

Application instrumentation


```
public async Task Handle(TMessage message, CancellationToken ct)
{
    var stopwatch = Stopwatch.StartNew();
    try
    {
        _logger.LogInformation($"Handle message {typeof(TMessage).Name}");
        var entity = _mapper.Map<TEntity>(message);
        await _repository.MergeEntity(entity, ct);

        _otelMetrics.HandlerSuccessCounter.Add(1, _tag);
        _otelMetrics.HandlerLatencyHistogram.Record(
            (long)(DateTime.UtcNow - message.TimestampUtc).TotalMilliseconds, _tag);
    }
    catch (Exception)
    {
        _otelMetrics.HandlerFailCounter.Add(1, _tag);
        throw;
    }
    finally
    {
        _otelMetrics.HandlerDurationGauge.SetValue(stopwatch.ElapsedMilliseconds, _tag);
    }
}
```

Decorator



Декоратор - структурный шаблон проектирования, который позволяет динамически подключать к объекту дополнительную функциональность.

Application instrumentation


```
public MonitoredMessageHandler(IMessageHandler<TMessage> messageHandler,
    OtelMetrics otelMetrics)
{
    _messageHandler = messageHandler;
    _otelMetrics = otelMetrics;
    _tag = new KeyValuePair<string, object?>("message", typeof(TMessage).Name);

    _otelMetrics.HandlerSuccessCounter.Add(0, _tag);
    _otelMetrics.HandlerFailCounter.Add(0, _tag);
}
```

Application instrumentation


```
public async Task Handle(TMessage message, CancellationToken ct)
{
    var stopwatch = Stopwatch.StartNew();

    try
    {
        await _messageHandler.Handle(message, ct);
        _otelMetrics.HandlerSuccessCounter.Add(1, _tag);

        _otelMetrics.HandlerLatencyHistogram.Record(
            (long)(DateTime.UtcNow - message.TimestampUtc).TotalMilliseconds, _tag);
    }
    catch (Exception)
    {
        _otelMetrics.HandlerFailCounter.Add(1, _tag);
        throw;
    }
    finally
    {
        _otelMetrics.HandlerDurationGauge.SetValue(stopwatch.ElapsedMilliseconds, _tag);
    }
}
```

dotnet add package Scrutor

```
services.Decorate(
    typeof(IMessageHandler<>),
    typeof(MonitoredMessageHandler<>));
```

HealthChecks

https://learn.microsoft.com/en-US/aspnet/core/host-and-deploy/health-checks?view=aspnetcore-7.0

HealthChecks


```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddHealthChecks();

var app = builder.Build();

app.MapHealthChecks("/health");

app.Run();
```

```
public class SampleHealthCheck : IHealthCheck
{
    0 references
    public Task<HealthCheckResult> CheckHealthAsync(
        HealthCheckContext context,
        CancellationToken cancellationToken = default)
    {
        var isHealthy = Random.Shared.Next(2) == 1;

        if (isHealthy)
            return Task.FromResult(
                HealthCheckResult.Healthy("A healthy result."));

        return Task.FromResult(
            new HealthCheckResult(context.Registration.FailureStatus,
            "An unhealthy result."));
    }
}
```

HealthChecks

Program.cs

```
services
    .AddHealthChecks()
    .AddCheck<SampleHealthCheck>("sample");
```

OtelMetrics.cs

```
HealthCheckGauge = new Gauge<long>(
    _meter, "health.check", description: "Health check status");

HealthGauge = new Gauge<long>(
    _meter, "health", description: "General application health status");
```

HealthChecks


```
public class MetricsHealthCheckPublisher : IHealthCheckPublisher
{
    private readonly OtelMetrics _otelMetrics;
    0 references
    public MetricsHealthCheckPublisher(OtelMetrics otelMetrics) => _otelMetrics = otelMetrics;

    0 references
    public Task PublishAsync(HealthReport report, CancellationToken cancellationToken)
    {
        foreach (var entry in report.Entries)
        {
            var tag = new KeyValuePair<string, object?>("name", entry.Key);
            _otelMetrics.HealthCheckGauge.SetValue((long)entry.Value.Status, tag);
        }

        _otelMetrics.HealthGauge.SetValue((long)report.Status);

        return Task.CompletedTask;
    }
}
```

```
services
    .AddHealthChecks()
    .AddCheck<SampleHealthCheck>("sample");

services.AddSingleton<
    IHealthCheckPublisher,
    MetricsHealthCheckPublisher>();
```

HealthChecks


```
# TYPE health_check gauge
# HELP health_check Health check status
health_check{name="sample"} 2 1683923545936

# TYPE health gauge
# HELP health General application health status
health 2 1683923545936
```

http://localhost:5000/metrics/

OpenTelemetry Collector

OpenTelemetry collector

https://opentelemetry.io/docs/collector/

OpenTelemetry collector

Файл: otel-collector-config.yaml

```yaml
receivers:
  otlp:
    protocols:
      grpc:

exporters:
  prometheus:
    endpoint: "0.0.0.0:8889"
    metric_expiration: 5m
    const_labels:
      env: production

extensions:
  health_check:

service:
  extensions: [health_check]
  pipelines:
    metrics:
      receivers: [otlp]
      exporters: [prometheus]
```

Файл: docker-compose.yaml

```yaml
version: "3.7"
services:
  otel-collector:
    image: otel/opentelemetry-collector:0.77.0
    command: ["--config=/etc/otel-collector-config.yaml"]
    volumes:
      - ./otel-collector-config.yaml:/etc/otel-collector-config.yaml
    ports:
      - "8889:8889"   # Prometheus exporter metrics
      - "4317:4317"   # OTLP gRPC receiver
```

OpenTelemetry collector


```
builder.Services.AddOpenTelemetry()
    .WithMetrics(builder =>
    {
        builder
            .SetResourceBuilder(ResourceBuilderHelper.CreateResourceBuilder(typeof(Program)))
            .AddMeter(OtelMetrics.MeterName)
            .AddInstrumentation<OtelMetrics>()
            .AddUpTimeInstrumentation()
            .AddOtlpExporter(op =>
            {
                op.Endpoint = new Uri("http://localhost:4317");
            });
    });
```

dotnet add package OpenTelemetry.Exporter.OpenTelemetryProtocol -v 1.5.1

OpenTelemetry collector


```
public static ResourceBuilder CreateResourceBuilder(Type assemblyTag)
{
    var assembly = Assembly.GetAssembly(assemblyTag)!.GetName();
    var resourceBuilder = ResourceBuilder
            .CreateDefault()
            .AddService(
                serviceName: assembly.Name!,
                serviceVersion: assembly.Version?.ToString(),
                autoGenerateServiceInstanceId: false,
                serviceInstanceId: Environment.MachineName);

    resourceBuilder.AddAttributes(new Dictionary<string, object>
    {
        ["machine.name"] = Environment.MachineName
    });

    var environment = Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT");
    if (environment is not null)
        resourceBuilder.AddAttributes(new Dictionary<string, object>
        {
            ["environment.name"] = environment
        });

    return resourceBuilder;
}
```

OpenTelemetry collector


```
# HELP application_uptime_milliseconds Milliseconds elapsed since application
startup
# TYPE application_uptime_milliseconds gauge
application_uptime_milliseconds{env="production",instance="PUGACH-
WORKSTAT",job="Dotnext.Demo.Service"} 61543

# HELP target_info Target metadata
# TYPE target_info gauge
target_info{environment_name="Development",instance="PUGACH-
WORKSTAT",job="Dotnext.Demo.Service",machine_name="PUGACH-
WORKSTAT",service_version="1.0.0.0"} 1
```

http://localhost:8889/metrics

Альтернативы OpenTelemetry в .NET

App.Metrics

https://www.app-metrics.io/getting-started/

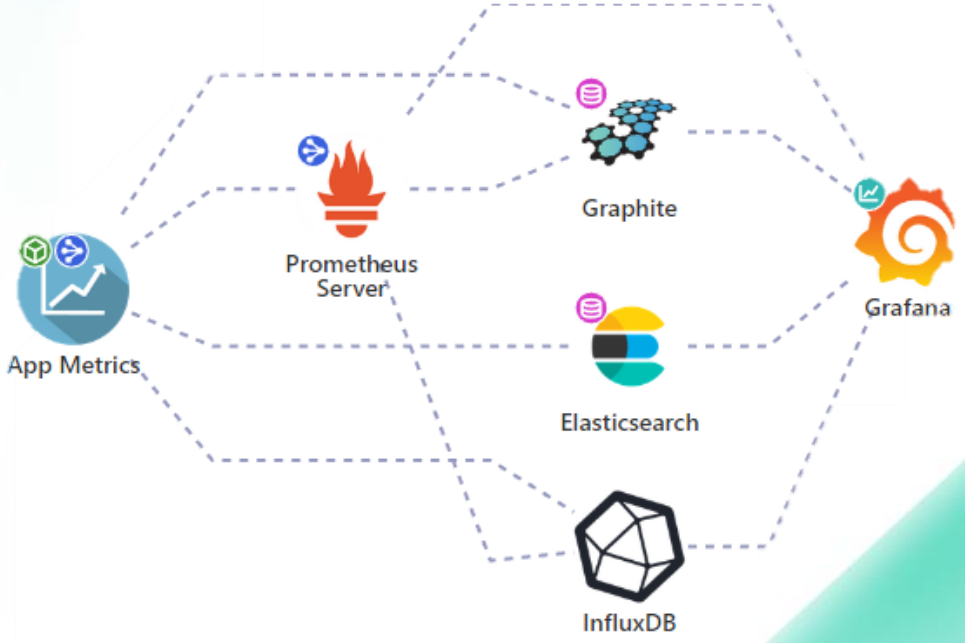
prometheus-net

https://github.com/prometheus-net/prometheus-net

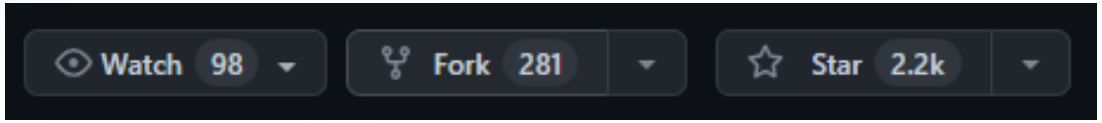
Prometheus.Client

https://github.com/prom-client-net/prom-client

App.Metrics

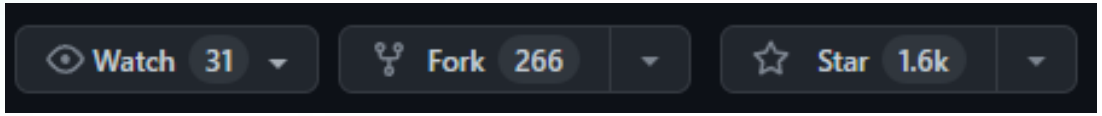


- Единый API для метрик с богатым функционалом
- Поддержка большого количество бекендов(InfluxDB, Prometheus, Graphite, Datadog, …)
- Большое количество расширений (HealthCheck, AspNetCore, Sql …)
- Хорошая документация

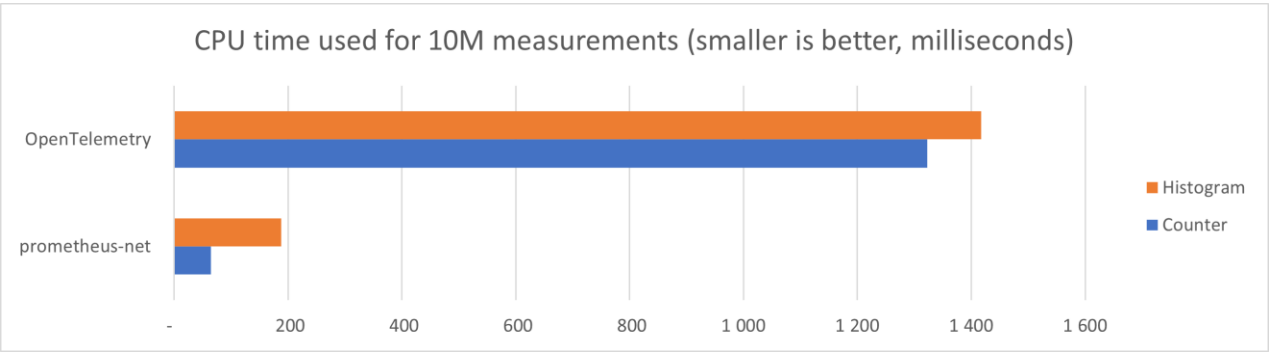


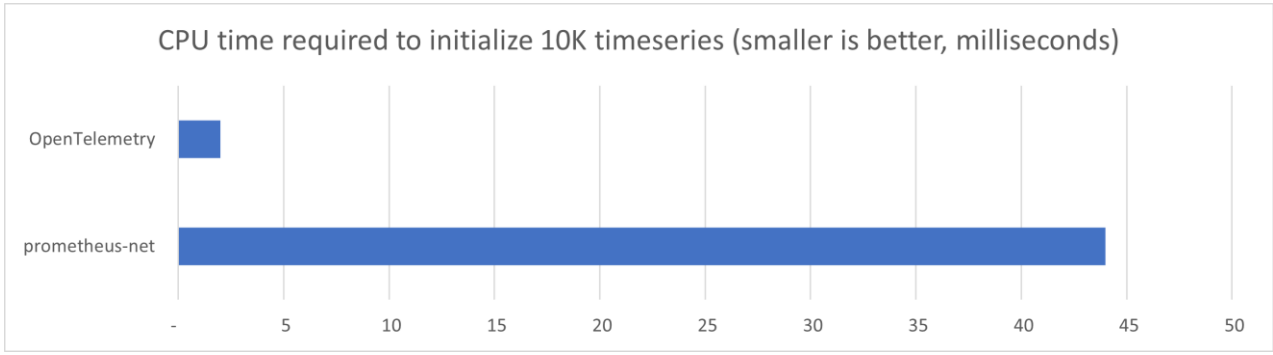
prometheus-net

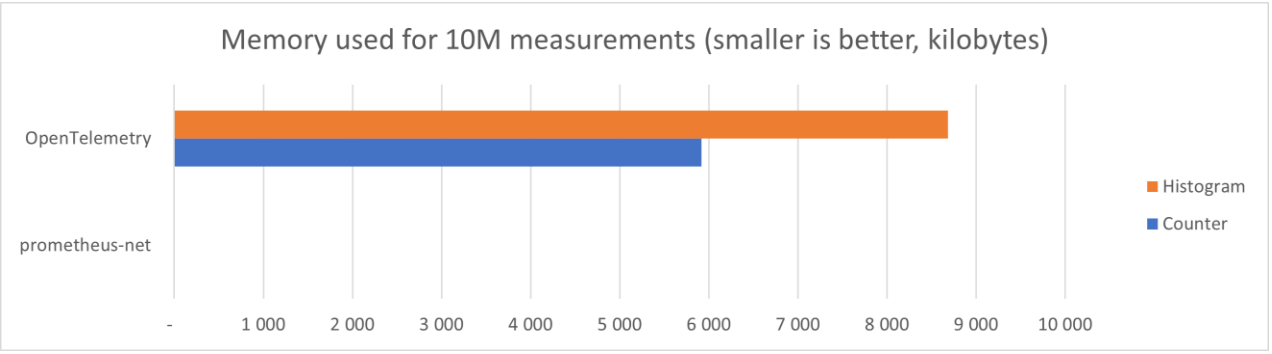
- Производительность
- Большое количество расширений (HealthCheck, AspNetCore, Sql …)
- Интеграция с Pushgateway
- Интеграция с EventCounters, DiagnosticSource, .NET Meters
- Хорошая документация

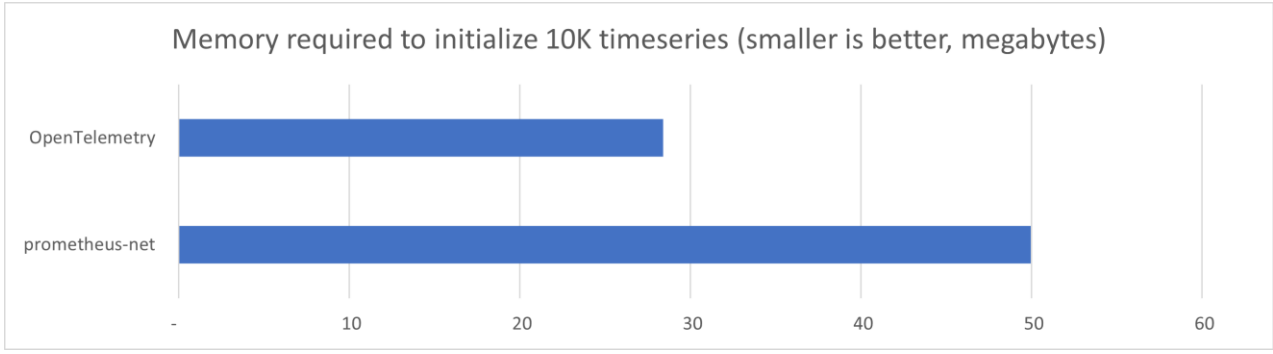


OpenTelemetry vs prometheus-net benchmarks







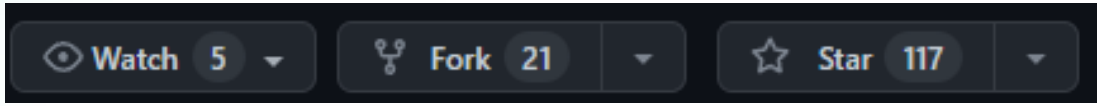


prometheus-net.AspNetCore 7.0.0

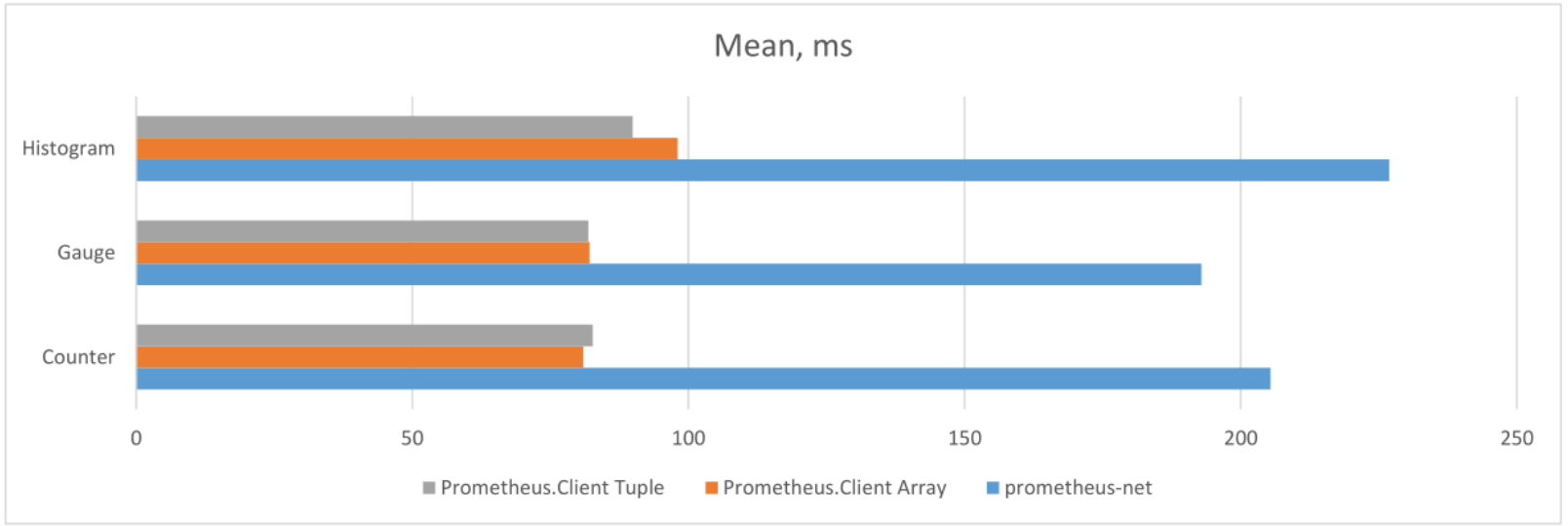
OpenTelemetry.Exporter.Prometheus.AspNetCore 1.4.0-rc.2

Prometheus.Client

- Производительность
- Потребление памяти
- Простота использования



Prometheus.Client vs prometheus-net benchmarks



Initialize 10K timeseries

prometheus-net 8.0.0

Prometheus.Client 5.2.0

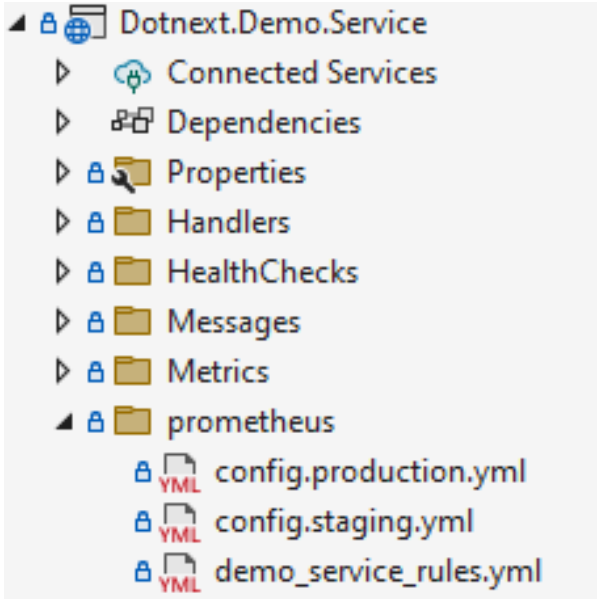
Альтернативы OpenTelemetry в .NET

App.Metrics	prometheus-net	Prometheus.Client
• Единый API для метрик с богатым функционалом • Поддержка большого количество бекендов(InfluxDB, Prometheus, Graphite, Datadog, …) • Большое количество расширений (HealthCheck, AspNetCore, Sql …) • Хорошая документация Когда использовать: Требуется независимость от бекенд сервисов и поддержка дополнительных расширений	• Производительность • Большое количество расширений (HealthCheck, AspNetCore, Sql …) • Интеграция с Pushgateway • Интеграция с EventCounters, DiagnosticSource, .NET Meters • Хорошая документация Когда использовать: Требуется использовать только Prometheus и поддержка дополнительных расширений	• Производительность • Потребление памяти • Простота использования Когда использовать: Требуется минимальный overhead на метрики

Alerting

Alerting


```yaml
groups:
- name: demo_service
  rules:
  - alert: demo_service_unhealthy.production
    expr: health{job="Dotnext.Demo.Service", env="production"} == 0
    for: 5m
    labels:
      priority: low
      severity: warning
    annotations:
      summary: "{{ $labels.app }} is unhealthy on '{{ $labels.instance }}'"
```



Alerting

Файл demo_service_rules.yml

```
groups:
- name: demo_service
  rules:
  - alert: demo_service_unhealthy.${ .env }$
    expr: health{job="${ .app }$", env="${ .env }$"} == 0
    for: ${ .default_duration }$
    labels:
      priority: low
      severity: warning
    annotations:
      summary: "{{ $labels.app }} is unhealthy on '{{ $labels.instance }}'"
```

Файл: config.production.yml

```
app: Dotnext.Demo.Service
env: production
default_duration: 5m
```

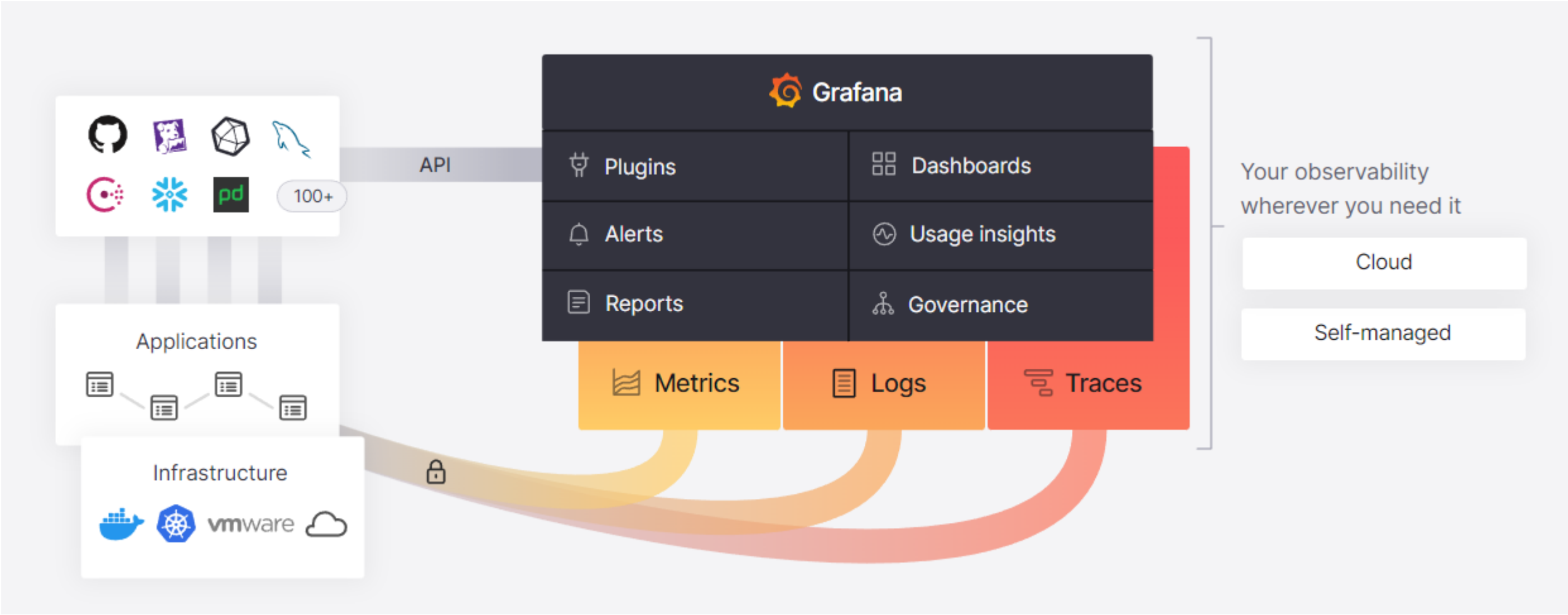
```
./gomplate_windows-amd64-slim.exe \
    -f demo_service_rules.yml \
    -c .=config.production.yml \
    -o demo_service_rules.production.yml \
    --left-delim '${' \
    --right-delim '}$'
```

https://github.com/hairyhenderson/gomplate

Grafana

https://grafana.com/

Grafana


Grafana
API
100+
Plugins
Dashboards
Alerts
Usage insights
Reports
Governance
Metrics
Logs
Traces
Applications
Infrastructure
vmware
Your observability
wherever you need it
Cloud
Self-managed

DEMO

Grafana


System
CPU Usage
100%
80%
60%
40%
20%
0%
05:00
05:30
06:00
06:30
07:00
07:30
08:00
Max Mean Last *
hqdwhssis4.avp.ru:5008 system 12.5% 1.65% 0%
hqdwhssis4.avp.ru:5008 user 31.3% 2.52% 0%
Process Memory Usage
1.84 GB
1.83 GB
1.83 GB
1.82 GB
1.82 GB
1.81 GB
process_memory_usage_bytes hqdwhssis4.avp.ru:5008
Process Threads
hqdwhssis4.avp.ru:5008
66
CPU count
hqdwhssis4.avp.ru:5008
24
ThreadPool Threads
hqdwhssis4.avp.ru:5008
6
ThreadPool Threads Queue Length
hqdwhssis4.avp.ru:5008
0
Runtime Memory Usage hqdwhssis4.avp.ru:5008
1.50 GB
1 GB
500 MB
0 B
Max Mean Last *
Committed Memory Size hqdwhssis4.avp.ru:5008 1.53 GB 1.52 GB 1.53 GB
Objects Size hqdwhssis4.avp.ru:5008 1.31 GB 1.13 GB 1.13 GB
Allocations Size hqdwhssis4.avp.ru:5008 1.78 MB 211 kB 8.17 kB
Heap Fragmentation hqdwhssis4.avp.ru:5008
3 MB
2.50 MB
2 MB
1.50 MB
1 MB
500 kB
0 B
gen0 gen1 gen2 loh poh
Heap Generations hqdwhssis4.avp.ru:5008
1 GB
800 MB
600 MB
400 MB
200 MB
0 B
gen0 gen1 gen2 loh poh

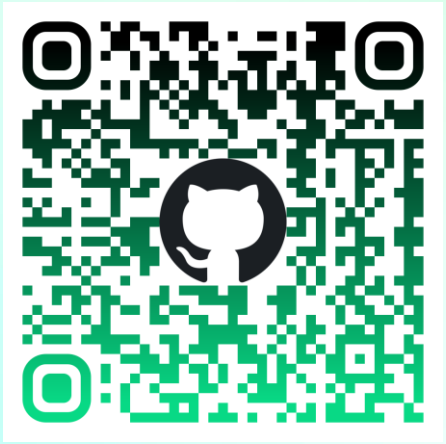
Выводы

- Метрики – это важно
- Измеряйте как можно раньше
- OpenTelemetry – это хороший выбор
- Учитывайте накладные расходы, которые вызывают метрики
- Выбирайте инструменты, которые вам подходят

Спасибо за внимание!

Буду рад ответить на вопросы!

Александр Пугач



kaspersky

@pugacha