

# I don't feel so well

## Integrating health checks in your .NET solutions

Alex Thissen  
Cloud architect

Follow along: <https://github.com/alexthissen/HealthMonitoring>

# Challenges for large-scale distributed systems

Keeping entire system running

Determine state of entire system and intervene

How to know health status of individual services?

## Collecting/correlating performance and health data

Events, metrics, telemetry, logs, traces

Usually centralized in a distributed landscape, e.g. micro-services



Raygun.io



NewRelic



AlertSite



AppMetrics



Azure Monitor



DataDog



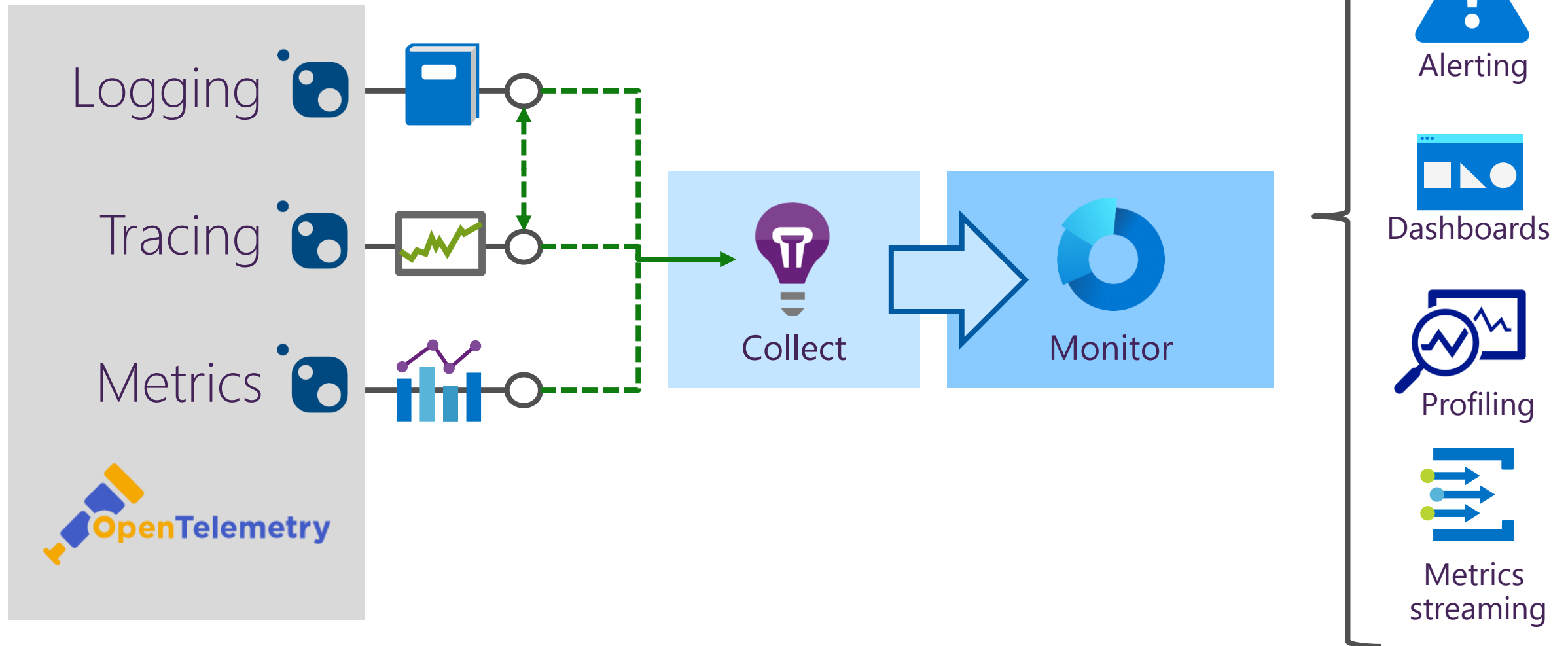
Sentry.io



Runscope

# Application instrumentation

Three signals for observability



# Traditional medicine and health

Doctor, am I sick?

12:34 ✓

Let's look at your vitals we measured:

- Pulse **180** per minute
- Blood pressure **150/110**

12:34

Does not look good.  
It seems you are unhealthy.

12:35

Thanks, doctor! 😊

12:36 ✓

## Centralized

Single point that knows how to assess health

## Challenging

Combining measurements to health information

Based on generic types of measured values

Absence of measurements

Differences in behavior from person to person

Unknown internals

Multiple places to access health

# Modern medicine and health

How are you doing today?

12:36

Let's see. My vitals say:  
- Pulse **180** per minute  
- Blood pressure **150/110**

12:34 ✓

It's okay, as I am working out now  
My back does not hurt.  
So, I'm healthy!

12:34 ✓

Good to know.  
Stay healthy!

12:36

## Self-assessment

Determining your own health status  
Know what defines healthy and unhealthy

## Context matters

Measurements might need to be interpreted differently

Depending on:

- Situation
- Circumstances
- Unmeasurable values

## You know best

# Difference between metrics and health info

## Metrics

Many individual measured values and counts of events

Watch performance and trends

Useful for diagnostics and troubleshooting

Logic external to origin



## Health

Intrinsic knowledge of implementation required

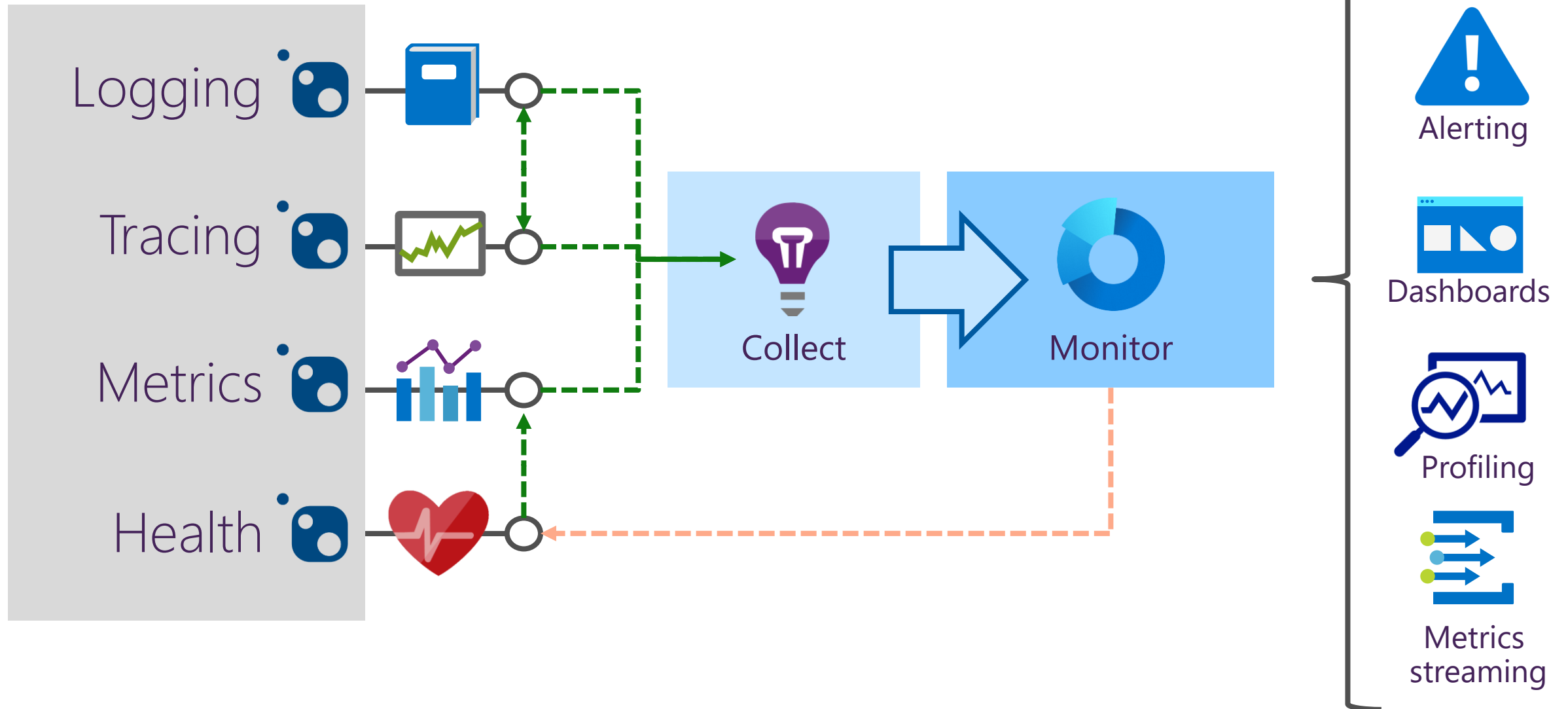
**DevOps mindset:**

Logic to determine health is part of origin

Deployed together, good for autonomy



# Application instrumentation



# Levels of health instrumentation



## Simple

### Availability

Any response  
Status code indication  
Formal endpoints

### Latency

Time to respond

### Internals

Memory  
Disk space



## Advanced

### External dependencies

- URL endpoints (e.g. Web API or CDN)
- Databases
- Service bus or queue
- Storage

### Readiness & liveness

Distinguish startup and normal operation  
Good for external lifetime management



## Preventive

### Predicting

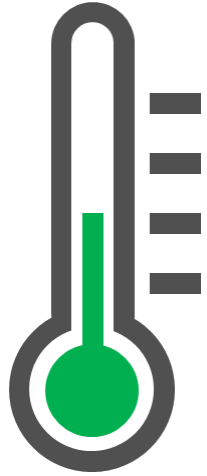
- Indication of impending failure
- Interesting with AI and ML

### Examples

- Expiring certificates
- Trends in memory pressure
- Failing resiliency countermeasures



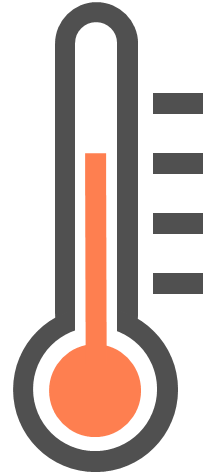
# Health status



Healthy

**200 OK**

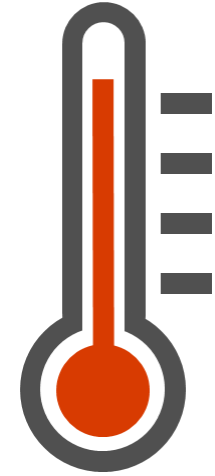
"Everything is fine"



Degraded

**200 OK**

"Could be doing better  
or about to become  
unhealthy"



Unhealthy

**503 Service**

**Unavailable**

"Not able to perform"

# Integrating health checks

Available since .NET Core 2.2 

Available to all .NET applications  
Plugs deep into ASP.NET Core

**Microsoft.Extensions.Diagnostics.HealthChecks**  
**.Abstractions**  
**.EntityFramework**

**Microsoft.AspNetCore.Diagnostics.HealthChecks**

## Bootstrap health checks in ASP.NET Core app

### Dependency injection

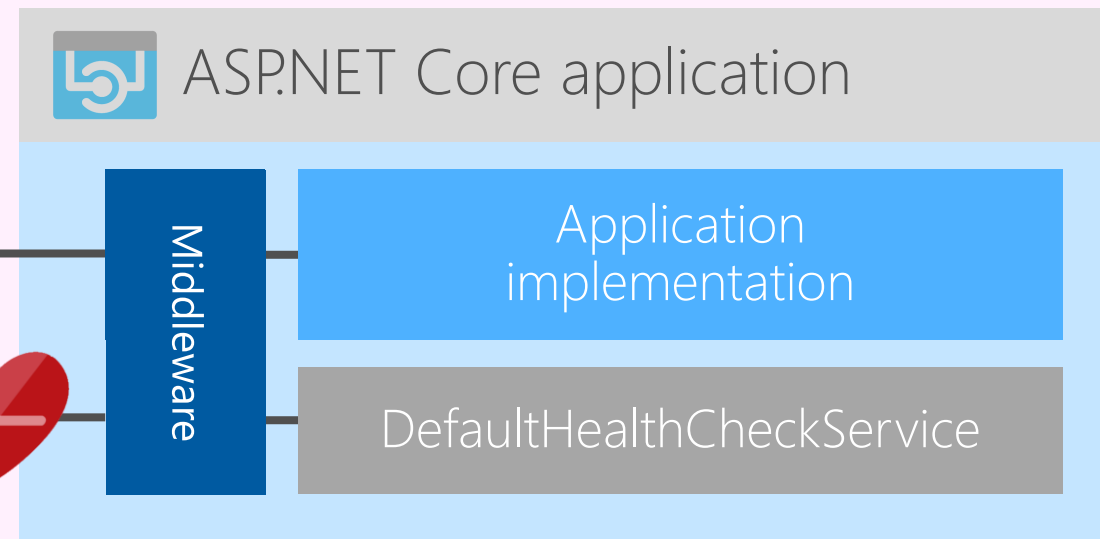
```
builder.Services.AddHealthChecks();
```

### ASP.NET Core middleware routing

```
app.MapHealthChecks("/health");
```

/api/v1/...

/health



# Using health checks

## What?

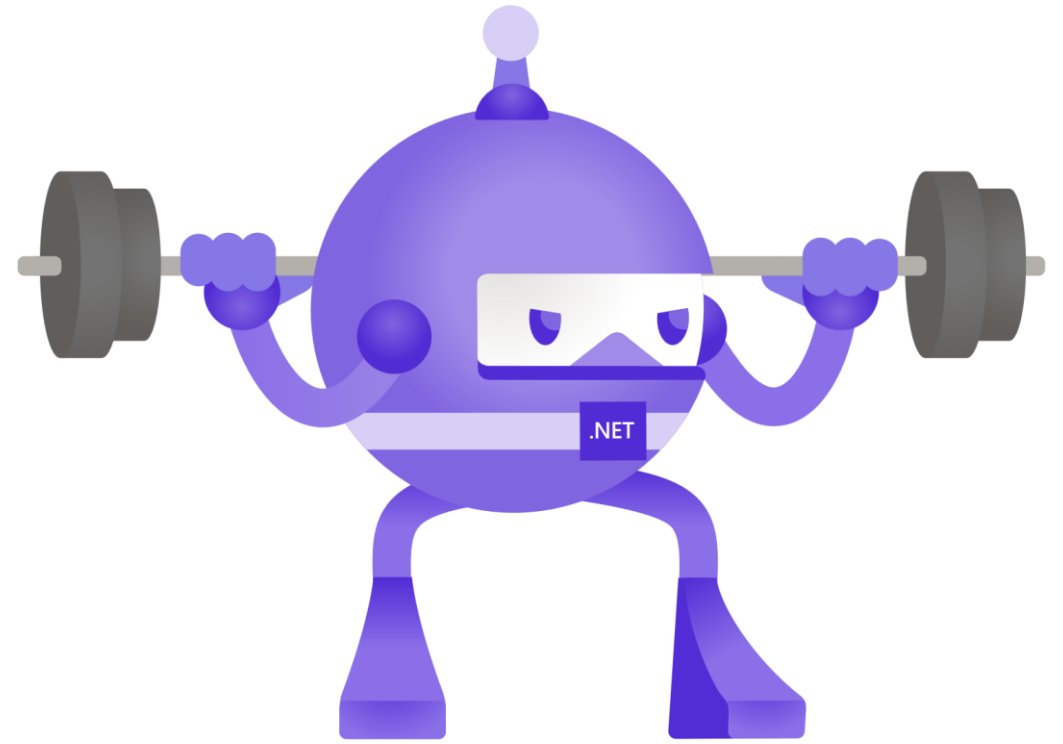
```
public interface IHealthCheck
{
    Task<HealthCheckResult> CheckHealthAsync(
        HealthCheckContext context,
        CancellationToken cancellationToken = default);
}
```

## When?

On demand from endpoints  
Periodically by publishers

## How?

Iterating over health check registrations

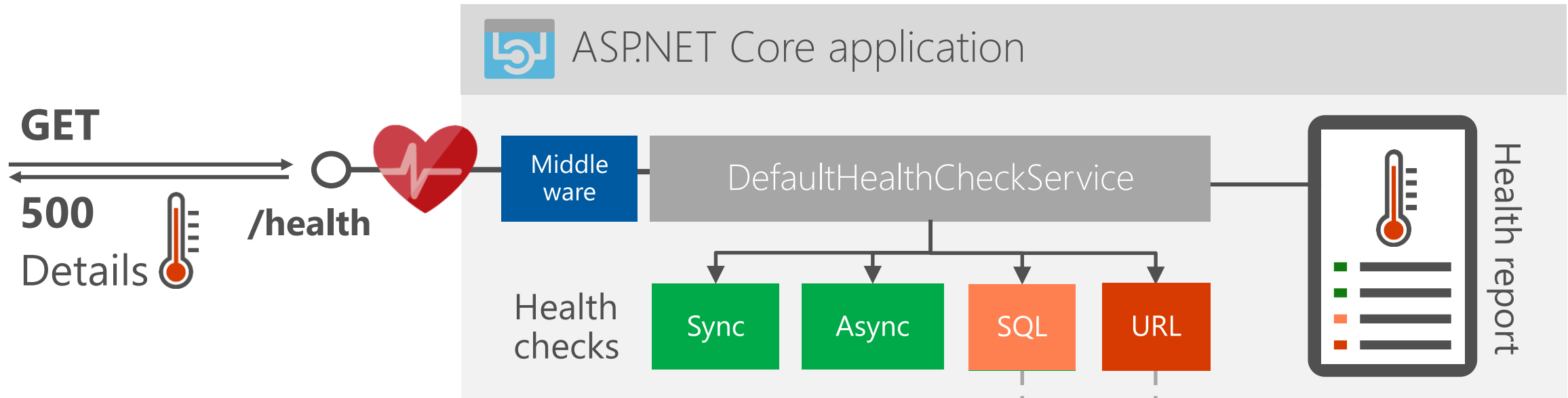


```
if (currentValue == HealthStatus.Failed)
{
    // Game over, man! Game over!
    // (We hit the worst possible status, so
    return currentValue;
}
```

**From:**

<https://github.com/dotnet/aspnetcore/blob/main/src/HealthChecks/Abstractions/src/HealthReport.cs>

# Integrating health checks



## builder.Services

### .AddHealthChecks()

```
.AddCheck("sync", () => ... )  
.AddAsyncCheck("async", async () => ... )  
.AddCheck<SqlConnectionHealthCheck>("SQL")  
.AddCheck<UrlHealthCheck>("URL");
```

# Demo

ASP.NET Core 8.0  
Health object model  
Health checks  
Endpoints

# Custom health checks

## Only 1 out-of-box check

Entity Framework [DbContext](#)

-  Microsoft.Extensions.Diagnostics.  
HealthChecks.EntityFrameworkCore

```
services.AddHealthChecks()  
    .AddDbContextCheck<GamingDbContext>("EF")
```

## Build your own

1. Delegate for sync or async factory
2. Implementation of [IHealthCheck](#)

## Community packages

-  [AspNetCore.Diagnostics.HealthChecks.\\*](#)

## Xabaril/BeatPulse



System (Disk Storage, Memory)

Network (Tcp, Ftp, Sftp, Imap, Smtpt, Dns resolve)

Azure Storage (Blobs, Tables and Queues)

Azure Service Bus (Event Hub, Service Bus queues and topics), SignalR

RabbitMQ

Kafka

Redis

Elasticsearch

EventStore

Identity Server

AWS DynamoDB

SqlServer

MongoDb

Oracle

DocumentDb

MySQL

Sqlite

Postgress Sql

## Yours?

# Beyond the basics

## Register multiple health endpoints

Order of registrations matters

## Middleware options

Change HTTP status codes per health result

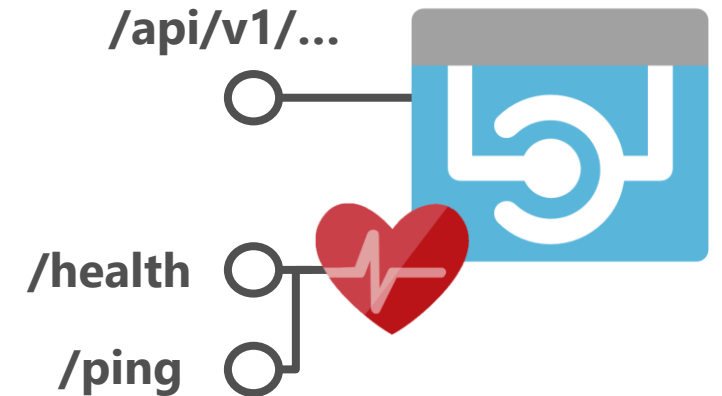
Allow client-side caching

Change response writing

Predicate for filtering health checks to evaluate

## Register custom health check as singleton

```
builder.Services.AddSingleton<KafkaHealthCheck>();  
builder.Services.AddSingleton(new SqlConnectionHealthCheck(  
    new SqlConnection(Configuration.GetConnectionString("MyDB"))));
```



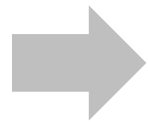
# Visualizing health checks

1. Customize health endpoint output for more details  
Specify delegate from `HealthCheckOptions.ResponseWriter`
2. Query endpoint(s)
3. Build user interface






## **Xabaril BeatPulse** `AspNetCore.HealthChecks.UI`

 Host in ASP.NET Core application

 Run from Docker container



Health Checks status Refresh every  seconds [Change](#)

	NAME	HEALTH	ON STATE FROM	LAST EXECUTION
	Readiness checks		Healthy 14 hours ago	9/29/2019, 11:55:38 PM
	Liveliness checks		Healthy 14 hours ago	9/29/2019, 11:55:38 PM



<DevSum>  
20th edition

Demo

A bit more advanced healthchecks

# Monitoring health



Endpoints



Frequency



Locations



Alerts

## AVAILABILITY TEST

↑↓ 20 MIN

↑↓ AVAILABILITY ↑↓

### Overall

0.00%

0.00%

▼ ⚠ Retro Gaming Web API Health check

0.00%

0.00%



⚠ Central US

0.00%

0.00%

⚠ East US

0.00%

0.00%

⚠ North Central US

0.00%

0.00%

⚠ South Central US

0.00%

0.00%

⚠ West US

0.00%

0.00%

Results

COUNT

FILTERING

1.09k

1

Success

1 Failed

# Health check publishers

Pushes out health info periodically

## Options

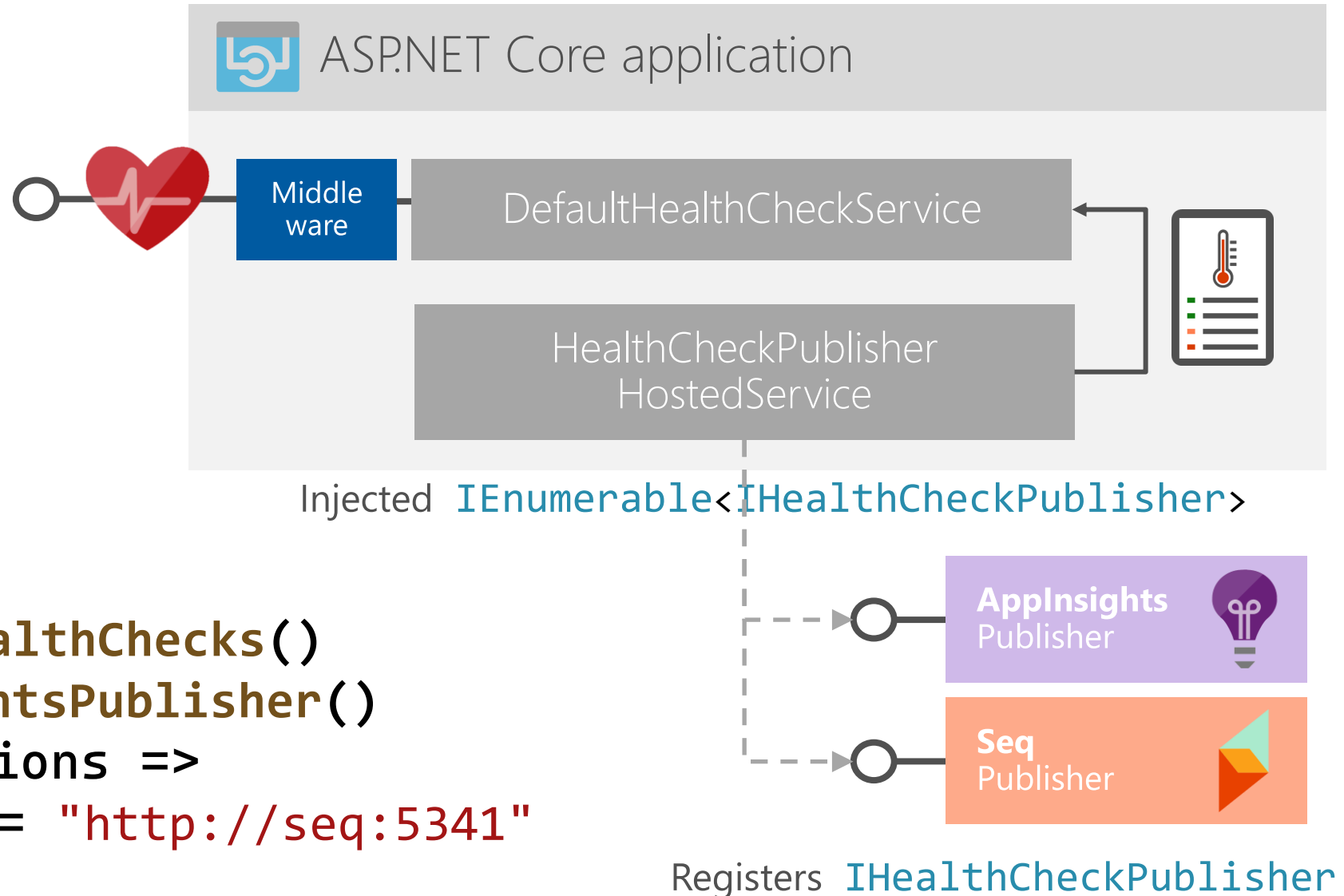
Timeout: max time to execute check

Delay: time to wait after startup

Period: period of execution

Predicate: Filter for checks to execute

```
builder.Services.AddHealthChecks()  
    .AddApplicationInsightsPublisher()  
    .AddSeqPublisher(options =>  
        options.Endpoint = "http://seq:5341"  
    );
```



<DevSum>  
20th edition

Demo

Health publishers  
Prometheus

# Resilient and self-healing applications

## Resiliency

Use cloud patterns:

- Circuit Breaker
- Timeout
- Retry



## Performance

Metrics

Instrumentation



## Availability

Zero-downtime upgrades

Readiness

Liveliness



## Monitoring

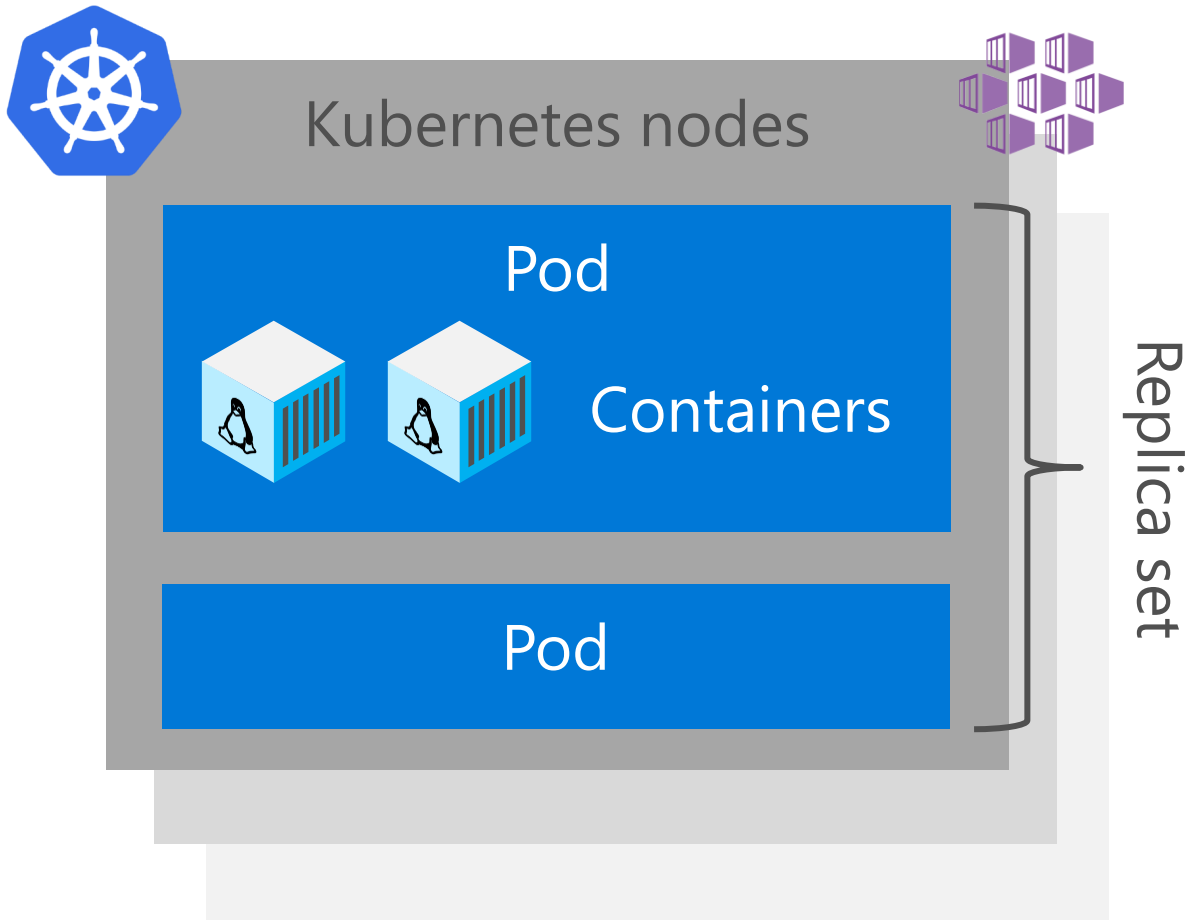
Health endpoint monitoring

Alerts



# Readiness and liveness

Probing containers to check for availability and health



## k8s-deployment.yaml

```
readinessProbe:  
  httpGet:  
    path: /health/ready  
    port: 8080  
  initialDelaySeconds: 20  
  periodSeconds: 10  
  timeoutSeconds: 10  
  failureThreshold: 3
```

```
livenessProbe:  
  httpGet:  
    path: /health/lively  
    port: 8080
```

## Readiness

Ready to receive incoming traffic

Not ready:  
remove container from load balancer

## Liveness

Indicates when to restart a container

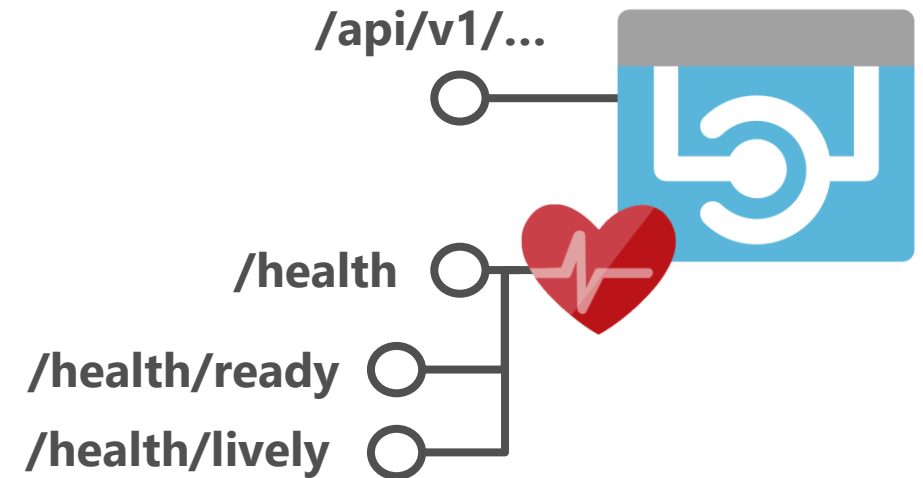
# Implementing readiness and liveness

1. Add health checks with tags

```
services.AddHealthChecks()  
    .AddCheck<CircuitBreakerHealthCheck>( "circuitbreakers",  
    tags: new string[] { "ready" }));
```

2. Register multiple endpoints with filter using Options predicate

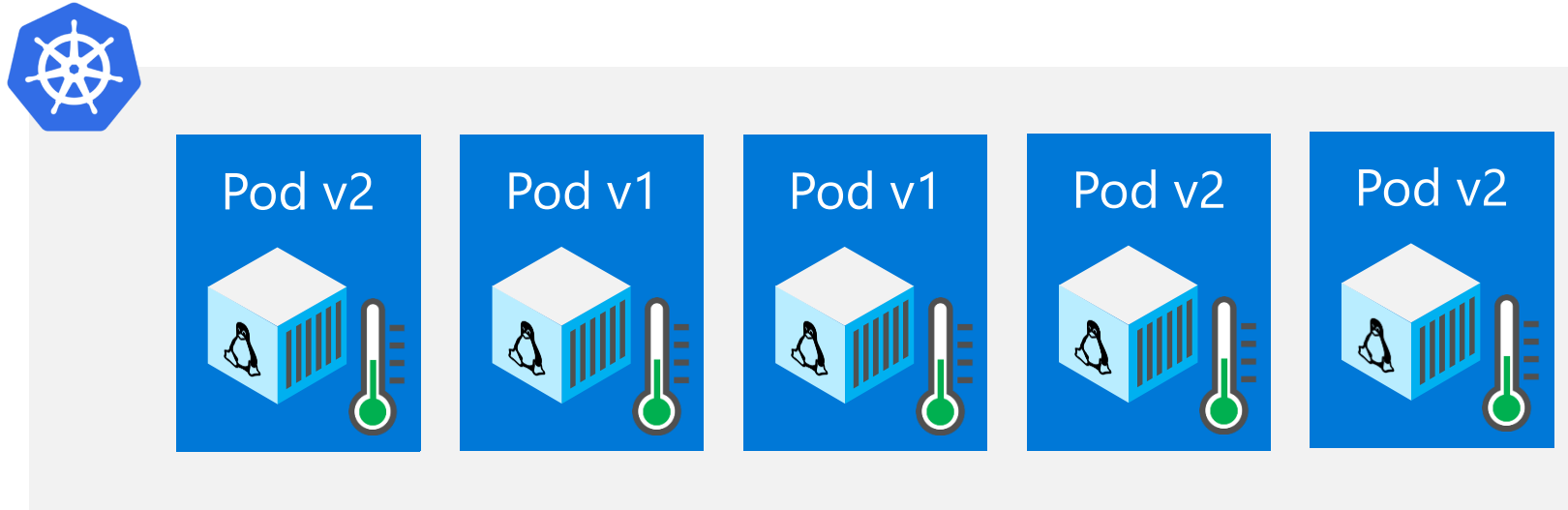
```
app.UseHealthChecks("/health/ready",  
    new HealthCheckOptions() {  
        Predicate = _reg => true && _reg.Tags.Contains("ready")  
    });
```



## Remember:

Order of registration matters

# Zero downtime deployments



```
spec:  
  replicas: 3  
  revisionHistoryLimit: 0  
  strategy:  
    type: RollingUpdate  
    rollingUpdate:  
      maxSurge: 2  
      maxUnavailable: 0
```

Original pods only taken offline after new healthy one is up  
Allows roll forward upgrades: Never roll back to previous version



# Demo

Readiness and liveness probes

Docker containers

Kubernetes

# .NET Aspire



*"An opinionated, cloud ready stack for building observable, production ready, distributed applications"*

## Orchestration

Composition  
Service discovery  
Connection string management



## Components



## Observability



# Health checks in .NET Aspire

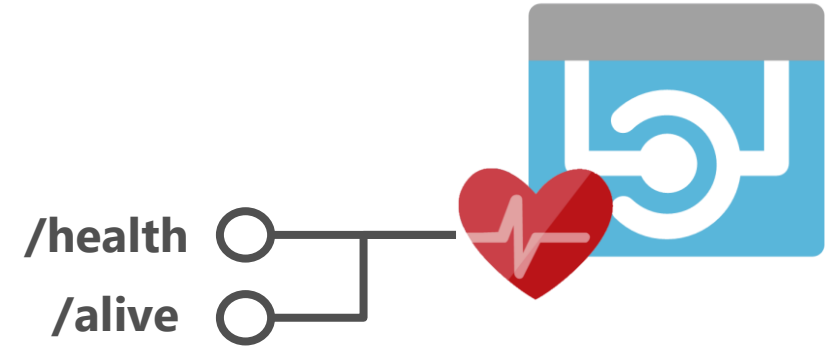
## Default health check "self"

Simple check for liveness with tag "live"

## Maps two (extra) health endpoints

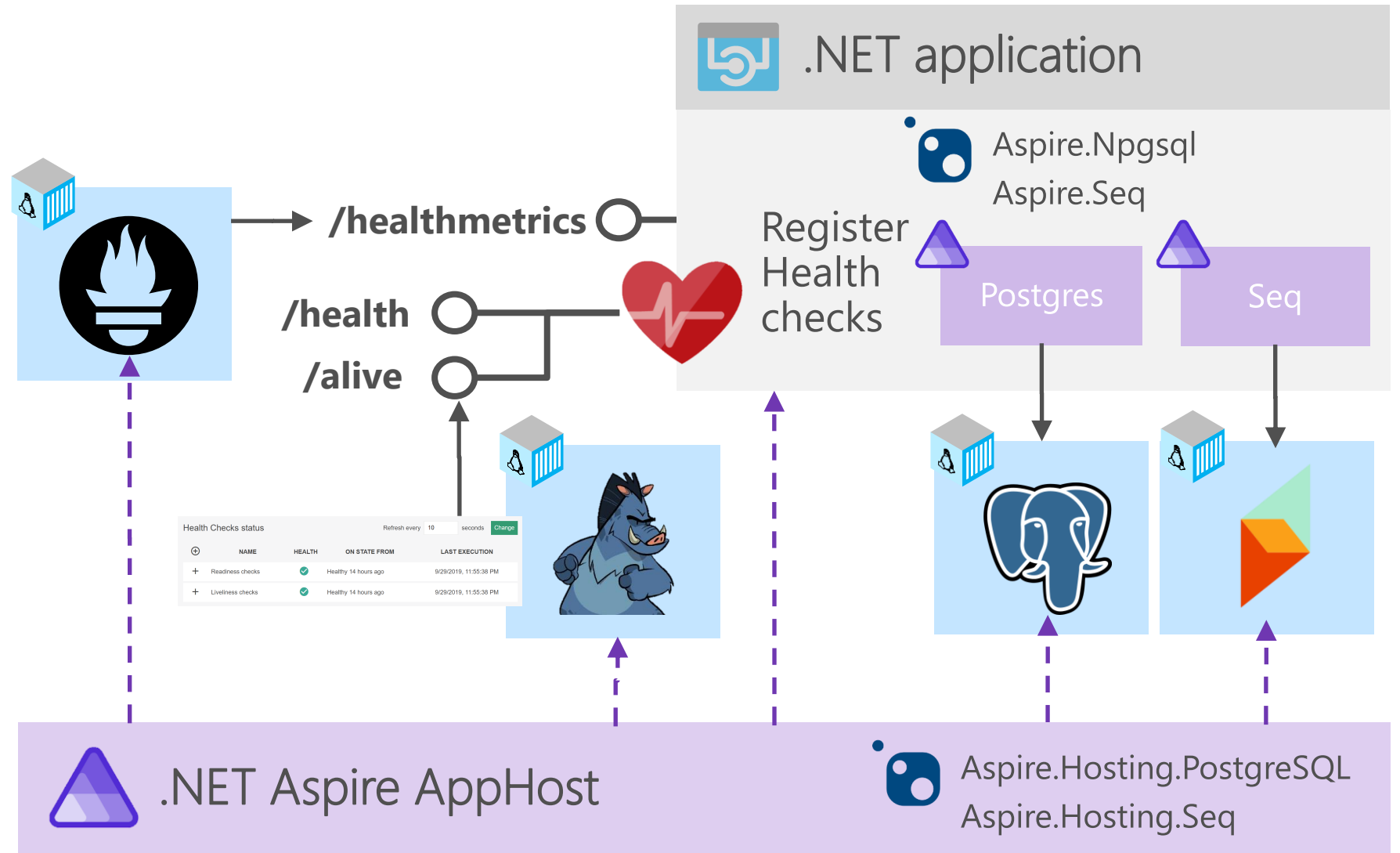
For development environment only

## Components can add health checks



```
builder.AddNpgsqlDbContext<MyDbContext>(
    "postgresdb",
    static settings => settings.DisableHealthChecks = true);
```

# Demo



# Securing

Expose as little detail as possible

Use different internal port

Inside a cluster ports are not exposed by default  
Leverage notion of a management port

Add authentication using  
middleware

```
app.MapHealthChecks("/securehealth",  
    new HealthCheckOptions() {  
        Predicate = _ => false  
    }).RequireAuthorization();
```

Publish instead of endpoint



# Best practices

1. Assume degraded state

2. Set short timeouts on checks

Inside health checks and for publishers

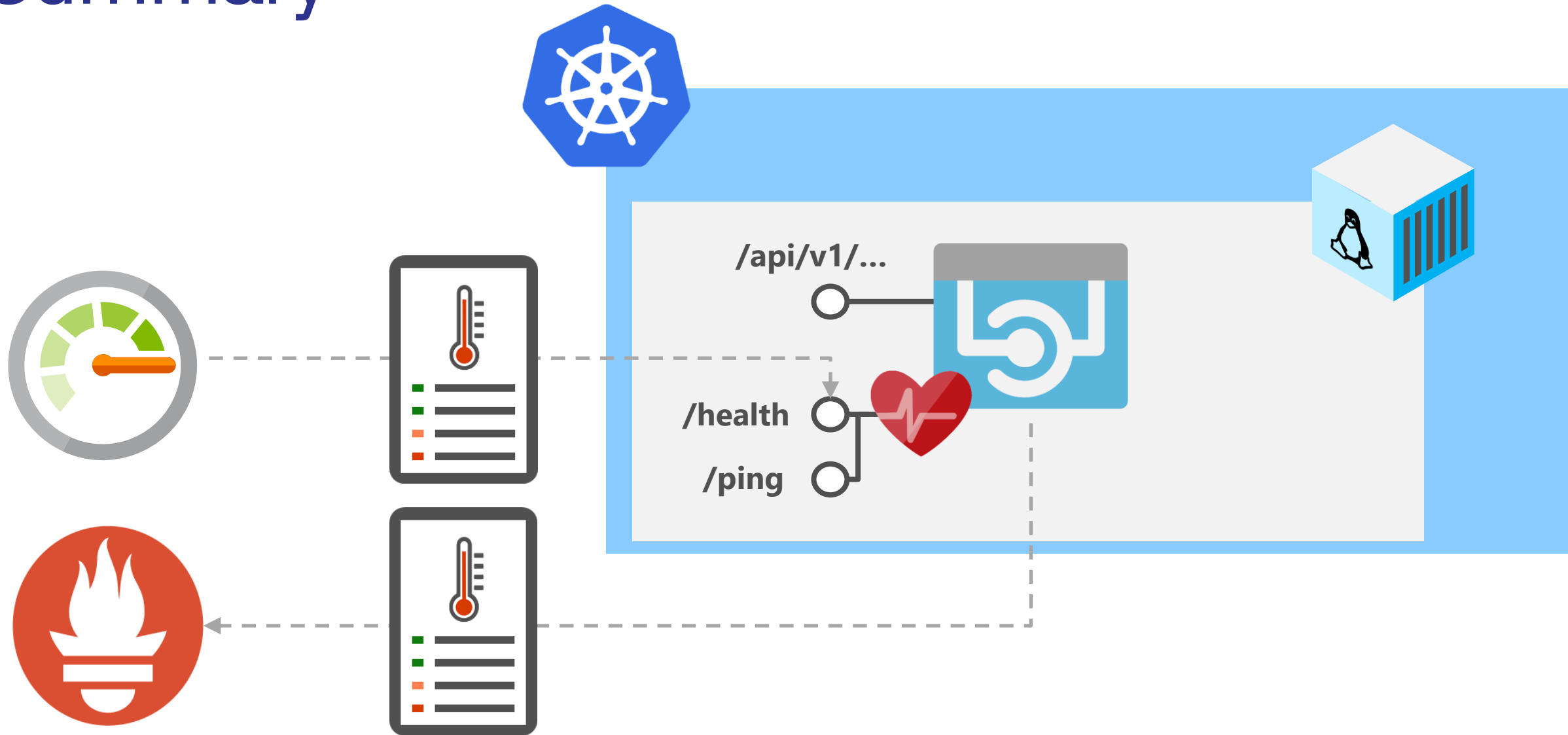
For example, when connecting to external dependencies

3. Avoid complicated health checks

4. Register health checks as singletons in DI

5. Reason about which health checks to use

# Summary



# Questions and Answers

Alex Thissen

@alexthissen

alex.thissen@xebia.com

<https://github.com/alexthissen/HealthMonitoring>



# Resources

## ASP.NET Core Health monitoring

<https://docs.microsoft.com/en-us/azure/architecture/patterns/health-endpoint-monitoring>

<https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/health-checks>

<https://github.com/aspnet/Diagnostics/tree/master/src>

## Kubernetes

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/>

## BeatPulse Xabaril

<https://github.com/Xabaril/AspNetCore.Diagnostics.HealthChecks>

## Demo source code

<https://github.com/alexthissen/healthmonitoring>