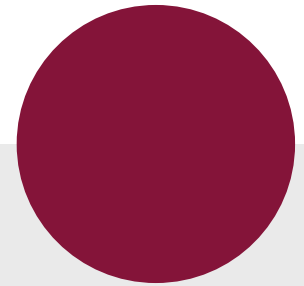




.consulting .solutions .partnership



# Anwendungsmetriken sammeln und auswerten mit Micrometer

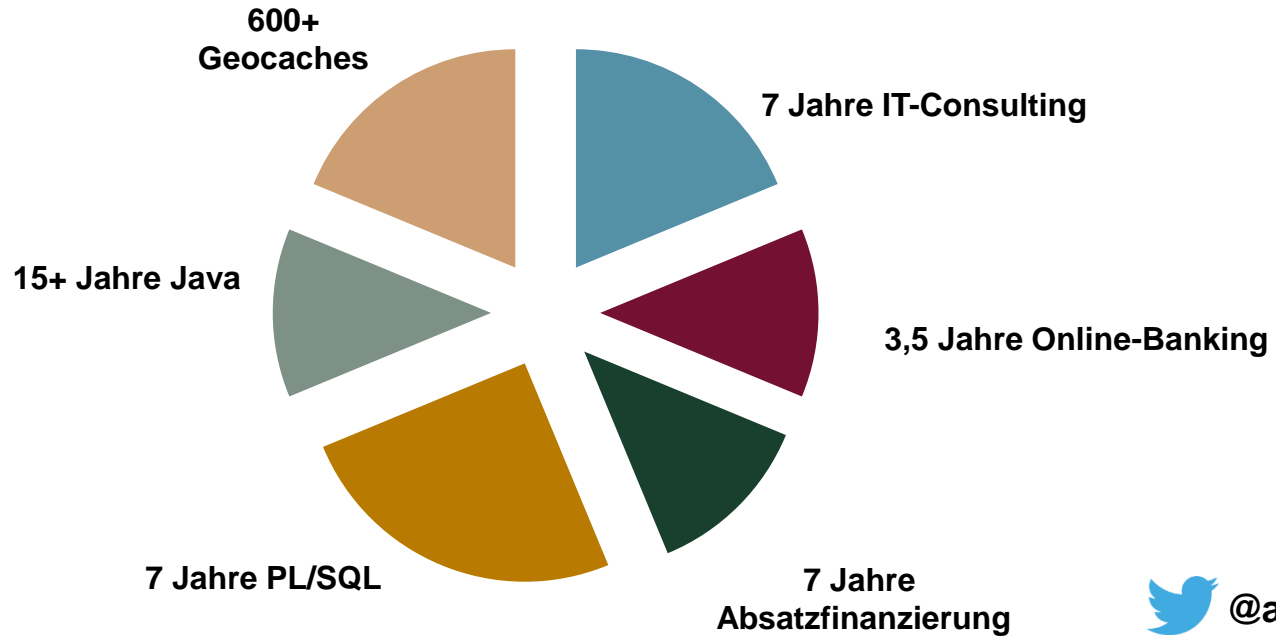
Alexander Schwartz, Principal IT Consultant

JavaLand Brühl, 2019-03-19

# Anwendungsmetriken sammeln und auswerten mit Micrometer

- 1 Metriken als Teil der Observability
- 2 Micrometer hilft Metriken zu sammeln
- 3 Setup der Beispiel-Infrastruktur
- 4 Metriken sammeln und anzeigen
- 5 Erfahrungen und Ausblick

# Über mich – Principal IT Consultant @ msg Travel & Logistics



 @ahus1de

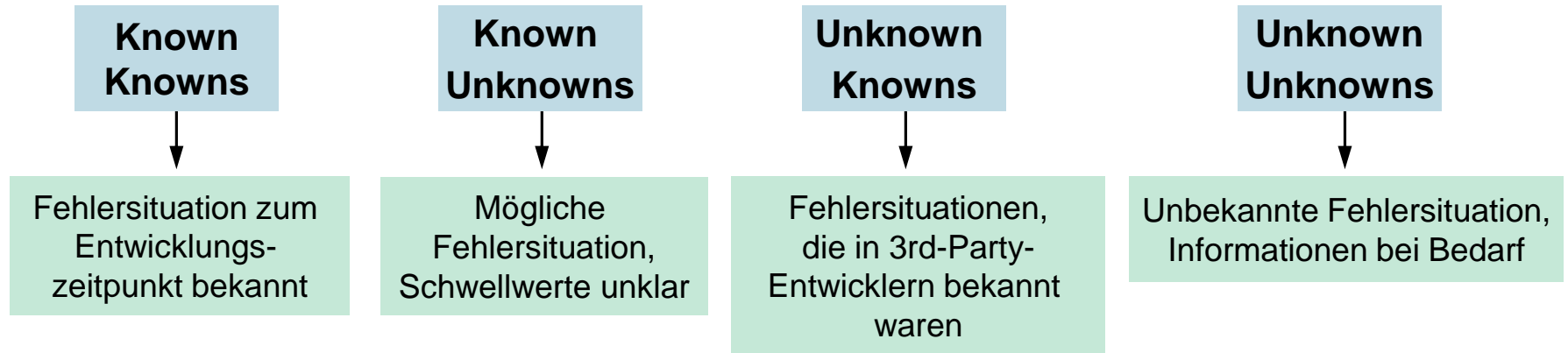
# Anwendungsmetriken sammeln und auswerten mit Micrometer

- 1 Metriken als Teil der Observability**
- 2 Micrometer hilft Metriken zu sammeln
- 3 Setup der Beispiel-Infrastruktur
- 4 Metriken sammeln und anzeigen
- 5 Erfahrungen und Ausblick

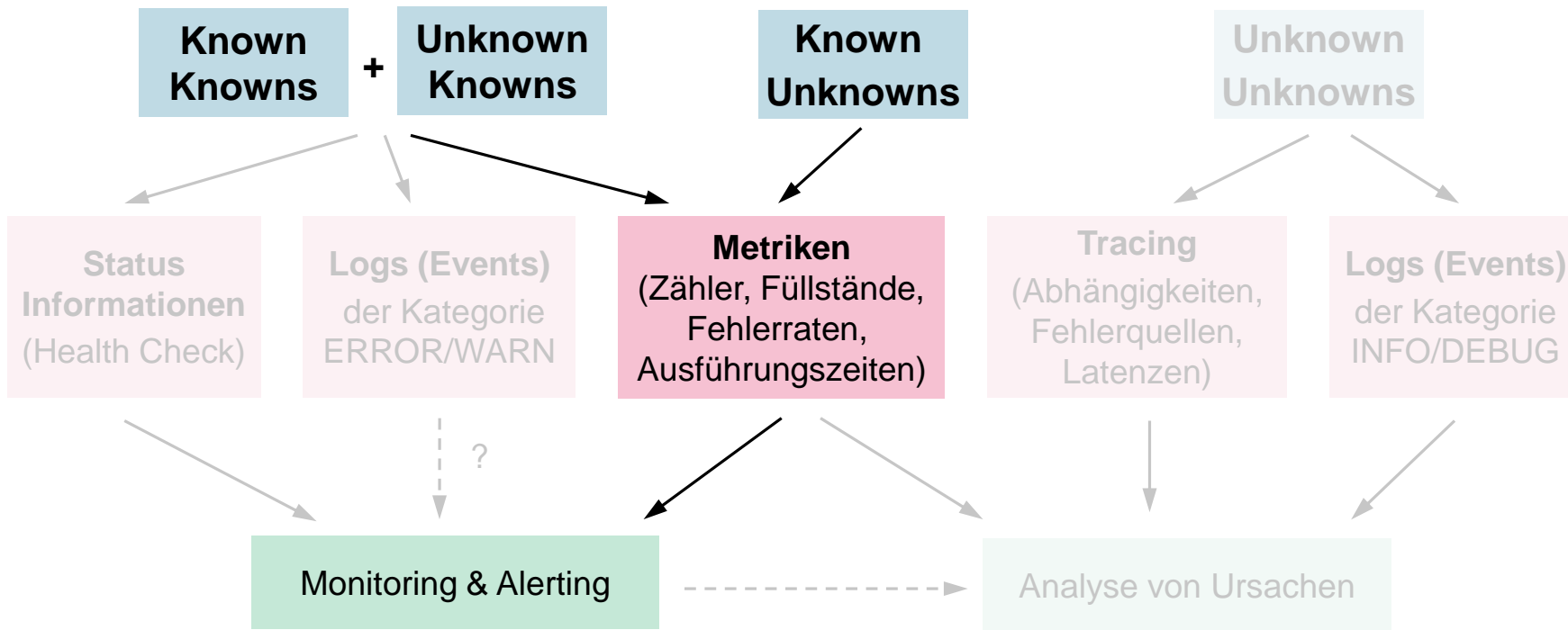
## Überblick Observability

**Monitoring** sagt mir, dass etwas kaputt ist (Symptom). Es ist Basis für eine Alarmierung, wenn es eine sofortige manuelle Reaktion erfordert.

**Observability** ist der ganze Rest den ich brauche um herauszufinden, warum etwas nicht funktioniert.



## Einbinden in das Monitoring



# Anwendungsmetriken sammeln und auswerten mit Micrometer

- 1 Metriken als Teil der Observability
- 2 Micrometer hilft Metriken zu sammeln**
- 3 Setup der Beispiel-Infrastruktur
- 4 Metriken sammeln und anzeigen
- 5 Erfahrungen und Ausblick

## Micrometer

Micrometer [maɪ'krɒm.i.təʊ] ist eine Fassade (API), mit der in JVM-Anwendungen herstellerunabhängig Metriken gesammelt werden können („SLF4J, but for metrics“).

- Multidimensionale Metriken
- Bestehende Integrationen für Bibliotheken und Backends (z. B. Prometheus, Datadog, Ganglia, Graphite, JMX, New Relic)
- Fertig integriert in Spring Boot 1.x und 2.x
- Kann auch Standalone verwendet werden

Homepage: <https://micrometer.io/>

Lizenz: Apache 2.0





## Micrometer API

**Basis für alle Metriken:** Name, optional: Tags und Beschreibung

### Ausgewählte Metrik-Typen:

- Counter: Zähler, z. B. Anzahl von erfolgreichen und nicht erfolgreichen Aufrufen
- Gauge: Messwert, z. B. Anzahl der aktiven Datenbank-Verbindungen
- Timer: Stoppuhr, z. B. Dauer von Aufrufen

```
Counter myOperationSuccess = Counter
    .builder("myOperation")
    .description("a description for humans")
    .tags("result", "success")
    .register(registry);

myOperationSuccess.increment();
```

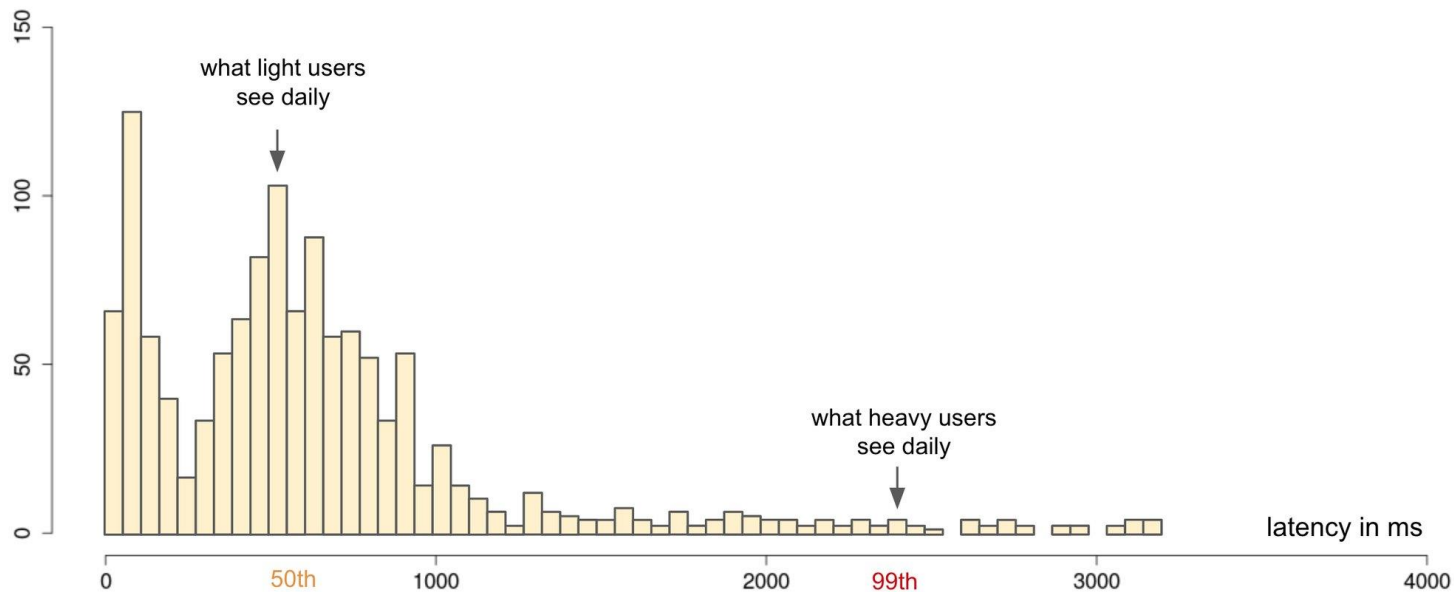
## Micrometer API

### **Abgeleitete Metriken:**

- Rate: z. B. Aufrufe pro Sekunde
- Percentile: z. B. 90% aller Aufrufe sind schneller als X ms
- Histogramm: z. B. X Aufrufe im Intervall von 50 bis 100 ms

Histogramme können über mehrere Instanzen aggregiert werden, Perzentile nicht!

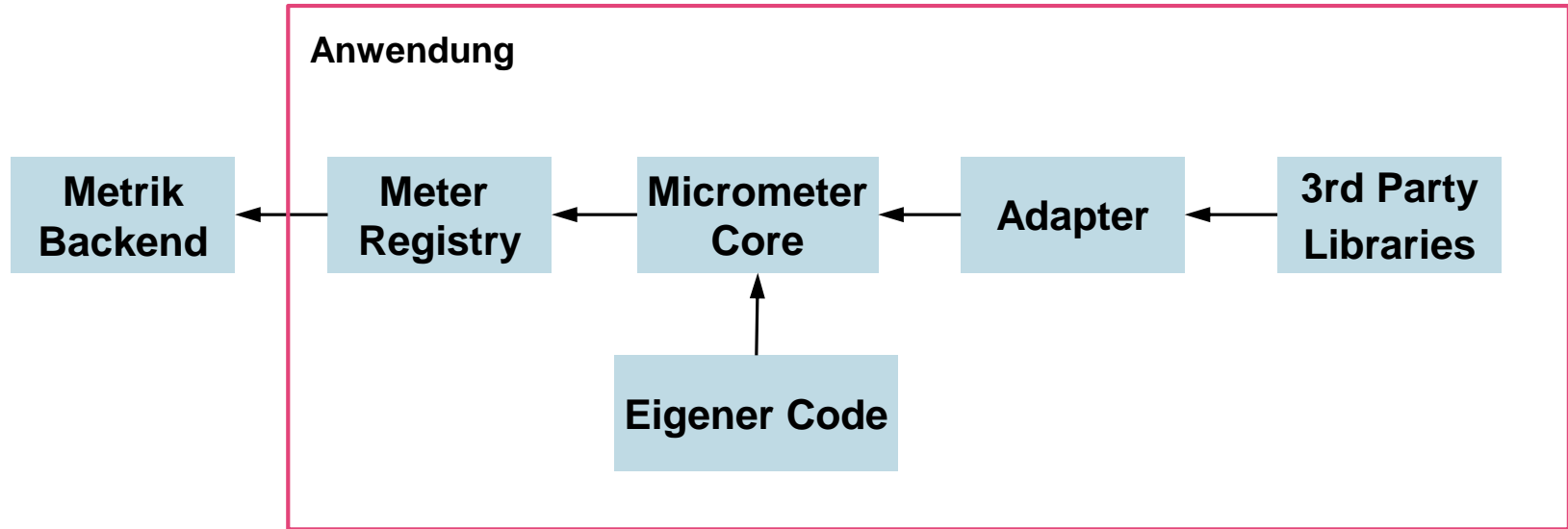
## A better way to explain why tail latency matters



Jaana B. Dogan (@rakyll)

1. <https://twitter.com/rakyll/status/1045075510538035200>

## Micrometer Architektur



Richtung des Pfeils: Datenfluss

## Micrometer Architektur

### **Micrometer Core:**

- Keine Abhängigkeiten zu anderen Bibliotheken, enthält nur die API

### **Meter Registry:**

- Stellt die Anbindung an ein Backend zum Speichern von Metriken her. Nur abhängig von Core und dem spezifischen Backend  
(Beispiel: micrometer-registry-prometheus)

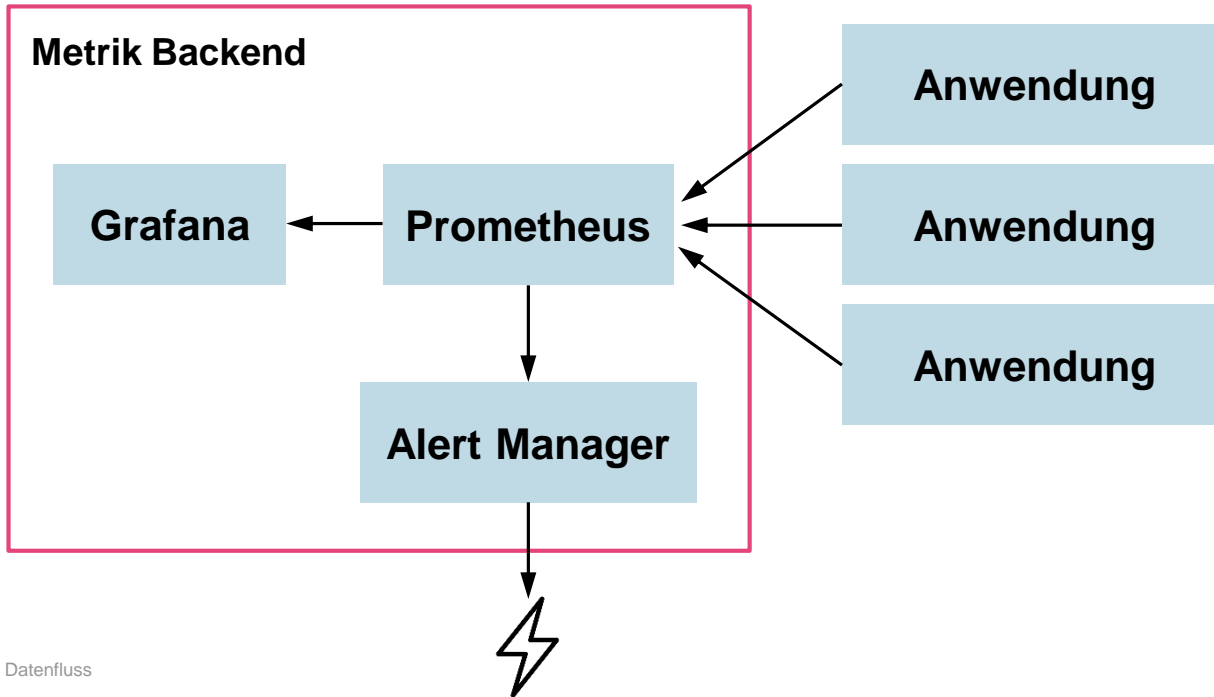
### **Adapter zum Abgreifen von Metriken:**

- spring-boot-actuator-autoconfigure: Stellt Prometheus-Endpoint bereit, holt Metiken aus Caches, AMQP, Web
- spring-cloud-netflix-core: Holt Metriken aus Circuit-Breaker-Library Hystrix

# Anwendungsmetriken sammeln und auswerten mit Micrometer

- 1 Metriken als Teil der Observability
- 2 Micrometer hilft Metriken zu sammeln
- 3 Setup der Beispiel-Infrastruktur**
- 4 Metriken sammeln und anzeigen
- 5 Erfahrungen und Ausblick

## Beispiel Infrastruktur



Richtung des Pfeils: Datenfluss

## Prometheus ist ein Monitoring System und eine Time Series Datenbank

Aufgaben: Sammeln und Speichern von Metriken,  
Abfragen z. B. für Dashboards, Alarmierung, Trendberechnung

Homepage: <https://prometheus.io>

Lizenz: Apache 2.0





## Grafana stellt interaktive Dashboards bereit

Aufgaben: Abfrage von Daten aus verschiedenen Quellen,  
Anzeigen als interaktive Dashboards

Homepage: <https://grafana.com>

Linenz: Apache 2.0



# Grafana

# Anwendungsmetriken sammeln und auswerten mit Micrometer

- 1 Metriken als Teil der Observability
- 2 Micrometer hilft Metriken zu sammeln
- 3 Setup der Beispiel-Infrastruktur
- 4 Metriken sammeln und anzeigen**
- 5 Erfahrungen und Ausblick

## Metriken mit Prometheus einsammeln

Prometheus fragt bei der Anwendung die aktuellen Metriken ab (Pull-Prinzip)

### 1. Spring Boot konfigurieren

```
# for testing purposes make it available publicly
management.endpoints.enabled-by-default=true
management.endpoints.web.exposure.include=health,info,metrics,prometheus
```

### 2. Anwendung stellt eine URL für Metriken bereit (/actuator/metrics bzw. /actuator/metrics/<name>)

```
{
  "name": "hikaricp.connections.active",
  "description": "Active connections",
  ...
}
```

## Metriken mit Prometheus einsammeln

### 3. Anwendung stellt eine URL für Prometheus bereit (/actuator/prometheus)

```
# HELP hikaricp_connections_active Active connections
# TYPE hikaricp_connections_active gauge
hikaricp_connections_active{pool="HikariPool-1",} 0.0
...
```

### 4. In Prometheus ist ein Target definiert (alternative via Service-Discovery)

```
- job_name: 'springboot'
  scrape_interval: 5s
  metrics_path: '/actuator/prometheus'
  static_configs:
    - targets: ['127.0.0.1:8080']
```

## Daten in Grafana anzeigen

- Metriken direkt anzeigen  
`http_server_requests_seconds_count`
- Filtern nach einem Label  
`http_server_requests_seconds_count{status!='200'}`
- Aufruftrate ermitteln im 5-Minuten-Intervall  
`rate (http_server_requests_seconds_count [5m])`
- Verhältnis der Fehler ermitteln als gleitender Durchschnitt  
`sum by (uri) (rate (http_server_requests_seconds_count {status!='200'} [5m]))`  
`/`  
`sum by (uri) (rate (http_server_requests_seconds_count [5m]))`

## Metriken in der Applikation sammeln

REST-Endpunkte zusätzlich zum Zähler mit Histogramm ausstatten

```
@GetMapping("/countedCall")
@Timed(histogram = true)
public String countedCall() {
    /* ... */
}
```

**Abfrage über:**

```
histogram_quantile(0.95,
    sum (rate(http_server_requests_seconds_bucket[5m])) by (le,uri))
```

## Metriken in der Applikation sammeln

### Aufrufe von Methoden messen

```
@Configuration
public class MetricsApplicationConfig {
    // as of now, this aspect needs to be created manually, see
    // https://github.com/micrometer-metrics/micrometer/issues/361
    @Bean
    public TimedAspect timedAspect(MeterRegistry registry) {
        return new TimedAspect(registry);
    }
}
```

```
@Component
public class ServiceClass {
    @Timed(value = "doSomething", description = "desc")
    public int doSomething() {
        /* ... */
    }
}
```

## Metriken in der Applikation sammeln

### Eigene Metriken erzeugen

```
@Component
public class Service {
    private final Counter myOperationCounterSuccess;
    public Service(MeterRegistry registry) {
        myOperationCounterSuccess = Counter
            .builder("myOperation")
            .description("a description for humans")
            .tags("result", "success")
            .register(registry);
    }
    public void myOperation() {
        myOperationCounterSuccess.increment();
        /* ... */
    }
}
```



# Anwendungsmetriken sammeln und auswerten mit Micrometer

- 1 Metriken als Teil der Observability
- 2 Micrometer hilft Metriken zu sammeln
- 3 Setup der Beispiel-Infrastruktur
- 4 Metriken sammeln und anzeigen
- 5 Erfahrungen und Ausblick**

## Metriken so einfach wie Logs

- Integriert in Spring Boot und viele 3<sup>rd</sup>-Party-Bibliotheken
- Auch Stand-Alone (ohne Spring Boot) verwendbar
- Dimensionalität und Beschreibungen sind eine deutliche Verbesserung gegenüber z. B. Dropwizard Metrics
- Zusätzliche Metriken werden automatisch gespeichert und können später ausgewertet werden
- Metriken sind effizienter als Log-Ausgaben – ggf. ergänzen oder ersetzen sie Logs
- Metriken können in Unit-Tests getestet werden
- Micrometer liefert JVM, Framework und Domain Metrics – kombiniert mit Infrastruktur-Metriken im gleichen Backend können Metriken in Relation gesetzt werden

## Links

### **Micrometer.io**

<https://micrometer.io>

### **Google SRE Book (Kapitel “Monitoring Distributed Systems”)**

<https://landing.google.com/sre/>

### **Prometheus:**

<https://prometheus.io>

### **Beispielprojekt**

<https://github.com/ahus1/prometheusspringbootminimal>

### **Grafana**

<https://grafana.com>

### **Folien**

<https://www.ahus1.de/post/micrometer>

### **Prometheus Chrome Plugin**

<https://github.com/fhemberger/chrome-prometheus-formatter>




@ahus1de



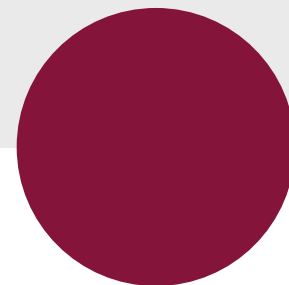
**Alexander Schwartz**  
Principal IT Consultant

+49 171 5625767  
alexander.schwartz@msg.group

 @ahus1de

**msg systems ag** (Headquarters)  
Robert-Buerkle-Str. 1, 85737 Ismaning  
Germany

**[www.msg.group](http://www.msg.group)**



.consulting .solutions .partnership