MICROMETER:
NEW INSIGHTS INTO YOUR SPRING BOOT APPLICATION

Michael Simons, INNOQ
@rotnroll666
Michael Simons
Senior Consultant
at INNOQ Deutschland GmbH

- First Spring project 2009 (Spring 3)
- First Spring Boot project early 2014
- Writes about Java, Spring and architecture on info.michael-simons.eu
- Bestselling German book on Spring Boot 2 (springbootbuch.de)
- Rants about stuff as @rotnroll666 on Twitter
Ways to observe a system

- Logging
- Tracing
- Metrics
Micrometer

- New project by Pivotal
- "Like SLF4J but for metrics"
- Vendor-neutral application metrics facade
How to get Micrometer into Spring Boot?

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
How to get Micrometer into Spring Boot?

http://localhost:8080/metrics
How to get Micrometer into Spring Boot?

```json
{
    "mem": 664619,
    "mem.free": 327742,
    "instance.uptime": 602975,
    "uptime": 606611,
    "systemload.average": 3.19677734375,
    "heap.committed": 596480,
    "heap.init": 262144,
    "heap.used": 268737,
    "heap": 3728384,
    "nonheap.committed": 69472,
    "nonheap.init": 2496,
    "nonheap.used": 68142,
    "nonheap": 0,
    "threads.totalStarted": 101,
    "threads": 29,
    "datasource.primary.active": 0,
    "datasource.primary.usage": 0.0,
    "gauge.response.metrics": 2.0,
    "gauge.response.motd": 3.0,
    "gauge.response.star-star.favicon.ico": 8.0,
    "counter.status.200.star-star.favicon.ico": 1,
    "counter.status.200.metrics": 4,
    "counter.status.200.mine": 1008
}
```

„Drill down“ ie.

http://localhost:8080/metrics/counter.status.200.metrics
That changes a lot with Spring Boot 2

```xml
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.0.1.RELEASE</version>
  <relativePath/>
</parent>
```
That changes a lot with Spring Boot 2

curl -v localhost:8080/metrics
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /metrics HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 404
< Content-Type: application/json; charset=UTF-8
< Transfer-Encoding: chunked
< Date: Wed, 02 May 2018 20:26:59 GMT
<
* Connection #0 to host localhost left intact
{"timestamp":"2018-05-02T20:26:59.437+0000","status":404,"error":"Not Found","message":"No message available","path":"/metrics"}
That changes a lot with Spring Boot 2

management.endpoints.web.exposure.include = *
That changes a lot with Spring Boot 2

curl -v localhost:8080/actuator/metrics
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8080 (#0)
> GET /actuator/metrics HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200
< Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
< Transfer-Encoding: chunked
< Date: Wed, 02 May 2018 20:34:36 GMT
<
* Connection #0 to host localhost left intact
{"names":
["process.cpu.usage","jvm.memory.max","jvm.gc.pause","jvm.buffer.memory.used","tomcat.threads.current"]}%
What happened to the nice format? 😞

```json
{
  "names": [
    "jvm.buffer.memory.used",
    "jvm.memory.used",
    "jvm.memory.committed",
    "jdbc.connections.min",
    "hikaricp.connections.usage",
    "http.server.requests",
    "jdbc.connections.max",
    "hikaricp.connections.usage.percentile",
    "jdbc.connections.max",
    "tomcat.servlet.request.max",
    "system.cpu.count",
    "tomcat.sessions.active.current",
    "hikaricp.connections.usage.percentile",
    "jdbc.connections.max",
    "tomcat.servlet.request.max",
    "system.cpu.usage",
    "jvm.threads.live",
    "jvm.classes.loaded",
    "jvm.classes.unloaded",
    "jvm.threads.peak",
    "tomcat.threads.current",
    "tomcat.global.request",
    "hikaricp.connections.creation",
    "jvm.gc.memory.promoted",
    "tomcat.sessions.rejected",
    "tomcat.sessions.alive.max"
  ]
}
```
Now we have “real” drill down...

curl localhost:8080/actuator/metrics/http.server.requests | python -m json.tool

```json
{
    "name": "http.server.requests",
    "measurements": [
        {
            "statistic": "COUNT",
            "value": 509.0
        },
        {
            "statistic": "MAX",
            "value": 0.012536956
        }
    ],
    "availableTags": [
        {
            "tag": "method",
            "values": [
                "GET"
            ]
        },
        {
            "tag": "uri",
            "values": [
                "/mine",
                "/actuator/metrics/{requiredMetricName}"
            ]
        }
    ]
}
```
Dimensions all the way

curl localhost:8080/actuator/metrics/http.server.requests\?tag\=status:200\&tag\=uri:/mine | \
  python -m json.tool

```json
{
  "name": "http.server.requests",
  "measurements": [
    {
      "statistic": "COUNT",
      "value": 500.0
    },
    {
      "statistic": "MAX",
      "value": 0.0
    }
  ],
  "availableTags": [
    {
      "tag": "method",
      "values": ["GET"
    }
  ]
}
```
Core concepts
Core concepts

- Dimensionality
- Registries
- Meters
- An SPI for registries
**Dimensionality**

- A taxonomy: Hierarchical metrics
- Works fine for a limited amount of information
- `host-xyz.counter.status.200.mine` is still fine, but...
- Patterns for dashboards etc. get out of hands, blind to new taxonomies
From taxonomy to folksonomy

- Classification system based on tags
- Term from around 2004
- Advantages according to Wikipedia
  - Reflection of the users vocabulary
  - Flexible
  - Multidimensional
- Tags are key/value pairs as dimension of a metric
Creating tags

- Either as global common tags or
- on the metrics themselves

```
Timer.builder("presentation.slide.timer")
  .description("This is a timer.")
  .tags("conference", "Spring I/O", "place", "Barcelona")
  .register(meterRegistry);
```
Creating tags

• Either as global common tags or
• on the metrics themselves

Gauge.builder("jvm.memory.used", Runtime.getRuntime(), r -> r.totalMemory() - r.freeMemory())
  .tag("host", "chronos")
  .tag("region", "my-desk")
  .register(meterRegistry);
Registries
Registries

Collect, store and export metrics

• Simple registry

MeterRegistry registry = new SimpleMeterRegistry();
Registries

Collect, store and export metrics

• Simple registry
• Composite registries

CompositeMeterRegistry composite = new CompositeMeterRegistry();

Counter counter = composite.counter("counter");
counter.increment(); // noop

SimpleMeterRegistry simple = new SimpleMeterRegistry();
composite.add(simple);
counter.increment(); // now stuff happens
Registries

Collect, store and export metrics

• Simple registry
• Composite registries
• The global registry

MeterRegistry registry = Metrics.globalRegistry;
Registries

Collect, store and export metrics
• Simple registry
• Composite registries
• The global registry

The global registry is special
• An empty composite by default
• Can a be a source of meters everywhere
• One has to add a simple registry
• Or let Spring Boot wire up its registry to it

MeterRegistry registry = Metrics.globalRegistry;
Metrics: Measurement of meters over time
Gauges
What's next?
Three different kinds of metrics

- System metrics
- Application metrics
- Domain metrics (or KPIs)
System metrics

- Classloader
- JVM Memory usage
- JVM Garbage Collector
- JVM Thread usage
- Processor usage

```java
new ClassLoaderMetrics().bindTo(registry);
new JvmMemoryMetrics().bindTo(registry);
new JvmGcMetrics().bindTo(registry);
new ProcessorMetrics().bindTo(registry);
new JvmThreadMetrics().bindTo(registry);
```
Application metrics with Spring Boot

- Spring MVC
- Spring WebFlux
- RestTemplate (HTTP Client)
- Spring Integration
- Spring MQ
Domain metrics (or KPIs)

- What you get: A nice API to add YOUR metrics
- Bring your own metrics
  - „How many products did I sell the last hour?“
  - „How long did the booking process take?“
  - „How many pending customers are there?“
Monitoring
Where to store this?
Supported dimensional monitoring systems

- Atlas
- Datadog
- Datadog StatsD
- Influx
- Prometheus
- SignalFx
- Telegraf StatsD
- Wavefront
Supported hierarchical monitoring systems

- Graphite
- Ganglia
- JMX
- Etsy StatsD
How to store metrics?
Push model

- Atlas
- Datadog
- Graphite
- Ganglia
- Influx
- JMX
- all StatsD flavors
- SignalFx
- Wavefront
Poll model

• Prometheus

```
scrape_configs:
  - job_name: 'reactive-java-chain'
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ['localhost:8080']
  - job_name: 'reactive-kotlin-chain'
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ['localhost:8090']
```
Closing notes

- Micrometer has more features
  - Histograms
  - SLAs
  - Filter and transformer
- Micrometer is not specific to Spring Boot 2
- Can be used stand-alone
- Can also be used with Spring Boot 1

```xml
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-spring-legacy</artifactId>
  <version>${micrometer.version}</version>
</dependency>
```
Ressources

- **Demo:**
  github.com/michael-simons/blockchain-playground  
  (Contains talk as welk)

- **Slides:** speakerdeck.com/michaelsimons

- **Excellent Webinar by Jon Schneider:**
  https://www.brighttalk.com/webcast/14893/316569

- **Spring Boot Buch (German)**
  @SpringBootBuch // springbootbuch.de
Picture sources

• Meter: https://unsplash.com/photos/zPZ9vqqDNBA
• Library: https://unsplash.com/photos/xLhKkk2tKkA
• Tally counter: https://www.flickr.com/photos/jauladeardilla/5768409603/
• Gauge: https://unsplash.com/photos/HM731qUoUas
• Timer: https://unsplash.com/photos/rBPOfVqROzY
• Demo:
  https://unsplash.com/photos/Uduc5hJX2Ew
  https://unsplash.com/photos/FIPc9_VocJ4
  https://unsplash.com/photos/gp8BLyaTaA0
• Monitoring
  https://unsplash.com/photos/mcSDtbWXUZU