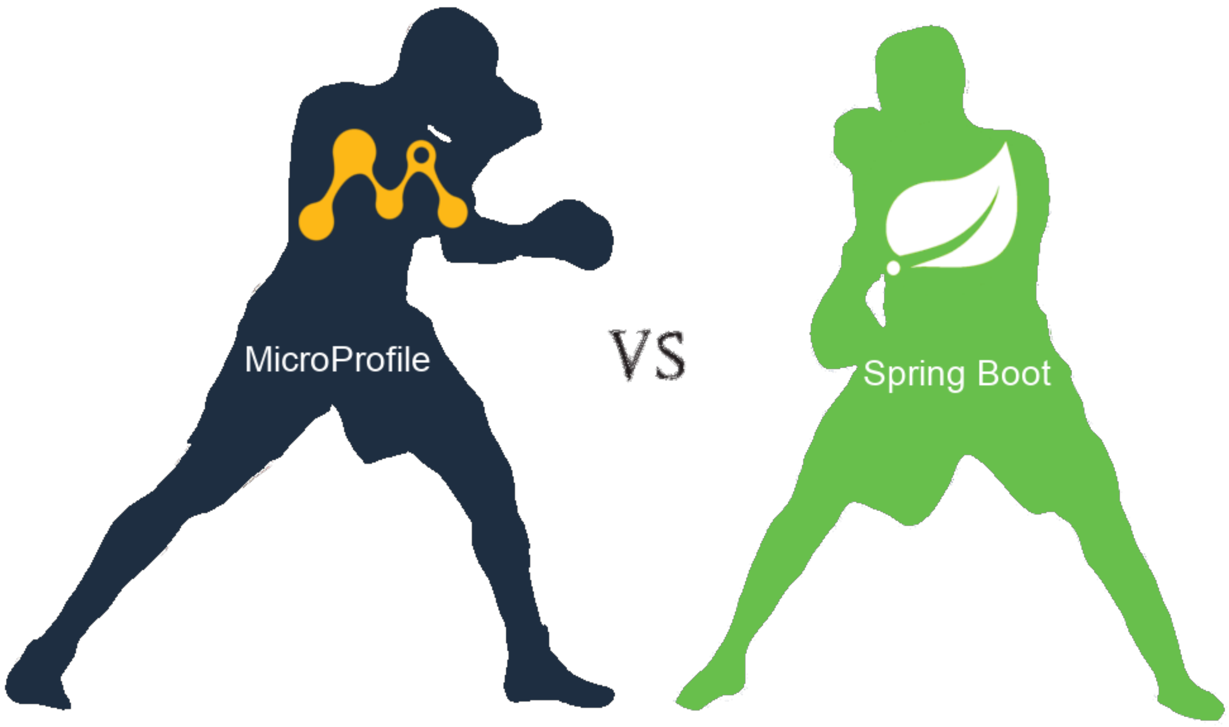
Java EE + MicroProfile - das bessere Spring Boot?



@anderScoreGmbH anderScore.company Java_Meetup_anderscore

Die Kontrahenten

- Der Champion: **Spring Boot**
    - Automagische Konfiguration
    - Starter für verschiedenste Anwendungsfälle
    - Embedded Tomcat, Jetty oder Undertow
    - Zugriff auf gesamtes Spring Ökosystem
- Der Herausforderer: **MicroProfile**
    - Robuste Standards und Implementierungen
    - Bulletproof
    - JEE/Jakarta Ökosystem

Die Kampfrichter

- **Daniel Krämer**
- Software-Entwickler, Architekt
- Integration und Migration
- Web Engineering
- Testautomatisierung
- dkraemer-anderscore

- **Maik Wolf**
- Software-Entwickler
- Fullstack & Devops
- JEE/Jakarta - Fanboy
- @da_mwolf
- maikwolf

Der Boxstall

- Standort: Köln (mit Rheinblick…)
- Individuelle Softwareentwicklung
- Consulting und Festpreis
- Gesamter Application Life Cycle
- Konferenzen und Artikel
- Öffentliche Trainings

- Technologien
  - JEE, Spring
  - Wicket, Angular
  - Docker, Kubernetes, Apache Kafka
  - …
- Goldschmiede@anderScore

Die Kriterien

1. Small runnable application
2. Externe Konfiguration
3. REST Endpoints
4. Health Check
5. Metriken

Auf in den Ring!

Runde 1 - Small runnable application

Small runnable application

Runde 1 - Small runnable application - Spring

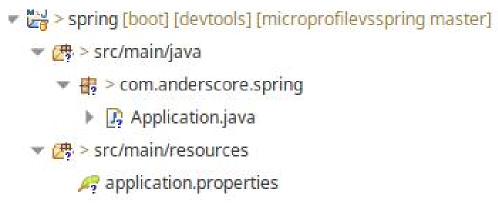
Philosophie:

> Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".
>
> We take an opinionated* view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.
>
> — https://spring.io/projects/spring-boot

*: eigensinnig, rechthaberisch

Runde 1 - Small runnable application - Spring

- **Aufsetzen eines Projektes**
    - Spring Initializr
    - CLI
    - IDE (Plugin)
- **Projektstruktur:**



```
> spring [boot] [devtools] [microprofilevsspring master]
    > src/main/java
        > com.anderscore.spring
            Application.java
    > src/main/resources
        application.properties
```

Runde 1 - Small runnable application - Spring

Starten der Anwendung:

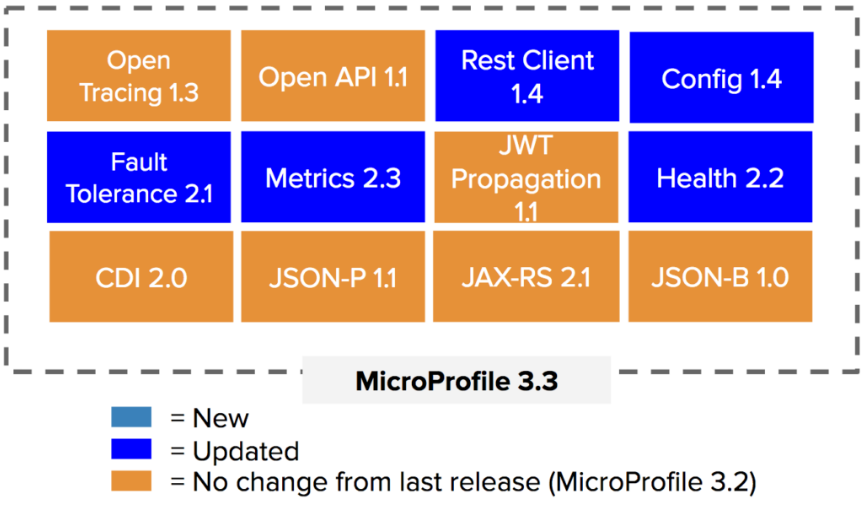
```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

```
mvn spring-boot:run
```

Runde 1 - Small runnable application - MicroProfile

MICROPROFILE™ OPTIMIZING ENTERPRISE JAVA

- Sammlung von Spezifikationen
- Fokus auf Microservice-Entwicklung



Runde 1 - Small runnable application - MicroProfile

THORNTAIL

Open Liberty

payara

helidon.io

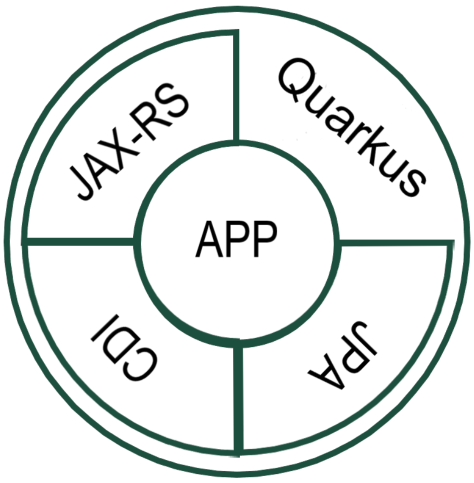
Runde 1 - Small runnable application - MicroProfile

> A Kubernetes Native Java stack tailored for OpenJDK HotSpot and GraalVM, crafted from the best of breed Java libraries and standards
>
> — https://quarkus.io/

Runde 1 - Small runnable application - MicroProfile

Quarkus

- Container first
- Einfach zu starten: `mvn quarkus:create`
- Live reload: `mvn compile quarkus:dev`
- Runner für JUnit 5
- MicroProfile 3.3
- Eine großes "Extension Ökosystem
- Java, Kotlin oder Scala



Runde 1 - Small runnable application - MicroProfile

- ECLIPSE VERT.X
- NETTY
- APACHE CAMEL
- INFINISPAN
- CAFFEINE
- KEYCLOAK

- KUBERNETES
- AWS LAMBDA
- AZURE FUNCTIONS
- APACHE TIKA
- ELASTICSEARCH
- KOGITO

Runde 1 - Small runnable application - MicroProfile

```
<project>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>${quarkus.platform.group-id}</groupId>
                <artifactId>${quarkus.platform.artifact-id}</artifactId>
                <version>${quarkus.platform.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-smallrye-health</artifactId>
        </dependency>
    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>io.quarkus</groupId>
                <artifactId>quarkus-maven-plugin</artifactId>
                <version>${quarkus-plugin.version}</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>build</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

Runde 1 - Small runnable application - MicroProfile

- **Aufsetzen eines Projektes**
  - MicroProfile Starter **oder** Quarkus Starter
  - CLI
- **Projektstruktur:**

```
.
├──── src
│     └──── main
│        ├──── docker
│        │     └──── Dockerfile.jvm
│        │     └──── Dockerfile.nativ
│        ├──── java
│        │     └──── de.anderscore.microprofile
│        │         └──── RestResource.java
│         └──── resources
│              └──── application.yml
└──── pom.xml
```

Runde 1 - Small runnable application - MicroProfile

Starten der Anwendung:

```
$ mvn package -Pnative && ./target/microprofile-quarkus
$ mvn quarkus:create
```

```
[io.quarkus] (main) Installed features: [
camel-core,
camel-microprofile-health,
camel-microprofile-metrics,
camel-support-common,
cdi,
rest-client,
resteasy,
smallrye-context-propagation,
smallrye-fault-tolerance,
smallrye-health,
smallrye-metrics
]
```

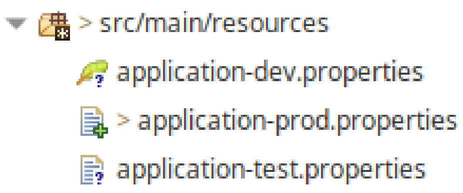
Runde 2 - Externe Konfiguration

Externe Konfiguration

Runde 2 - Externe Konfiguration - Spring

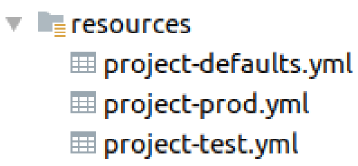
Profiles + Properties:

```
@Configuration
@PropertySource("classpath:application-${spring.profiles.active:dev}.properties")
@Import({PersistenceConfig.class, SecurityConfig.class})
public class AppConfig {
}
```


> src/main/resources
application-dev.properties
> application-prod.properties
application-test.properties

```
mvn spring-boot:run -Dspring.profiles.active=test
```

Runde 2 - Externe Konfiguration - MicroProfile


resources
project-defaults.yml
project-prod.yml
project-test.yml

Shell

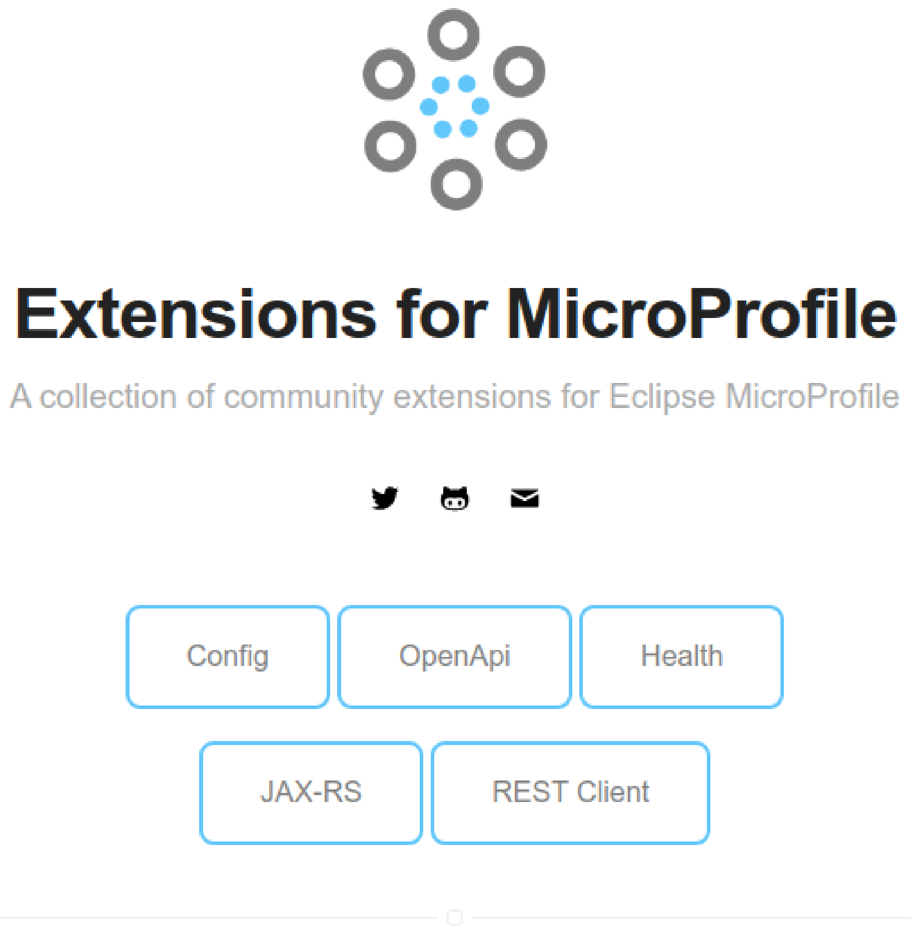
```
$ java -jar myapp-quarkus.jar -Stest
```

Runde 2 - Externe Konfiguration - MicroProfile

```
@Inject
@ConfigProperty(name="defaultEstimation")
private Long defaultEstimation;


@Inject
@ConfigProperty(name="defaultAssigne")
private Optional<Assigne> defaultAssigne;
```

- **Default ConfigSources**
  - System properties, Config file, etc.
- **Custom ConfigSources**
  - Config server, DB, etc.

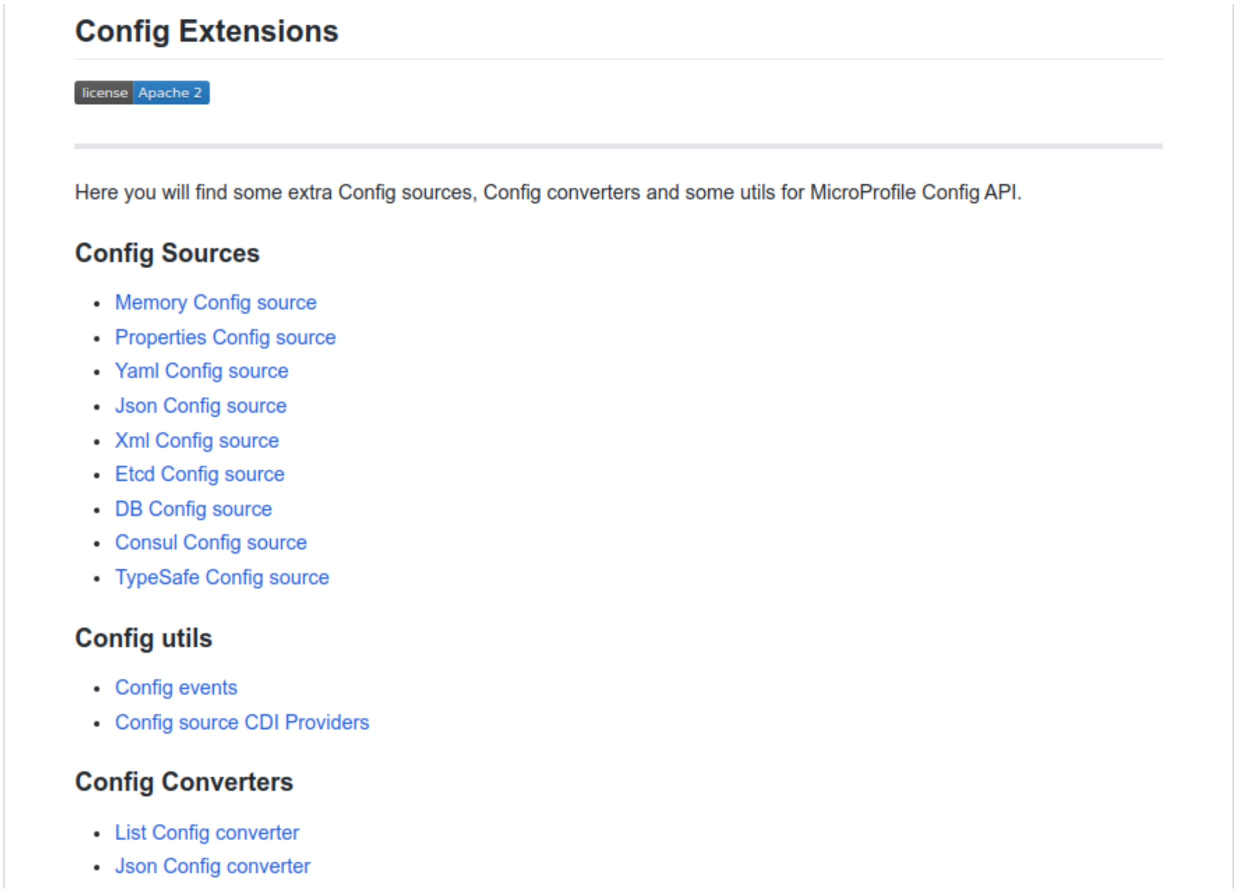
Runde 2 - Externe Konfiguration - MicroProfile



Extensions for MicroProfile

A collection of community extensions for Eclipse MicroProfile

https://microprofile-ext.org



Config Extensions

license Apache 2

Here you will find some extra Config sources, Config converters and some utils for MicroProfile Config API.

Config Sources

- Memory Config source
- Properties Config source
- Yaml Config source
- Json Config source
- Xml Config source
- Etcd Config source
- DB Config source
- Consul Config source
- TypeSafe Config source

Config utils

- Config events
- Config source CDI Providers

Config Converters

- List Config converter
- Json Config converter

Runde 3 - REST Endpoints

REST Endpoints

Runde 3 - REST Endpoints - Spring

REST Controller:

```
@RestController
@RequestMapping("/tasks")
public class TaskController {

    @Autowired
    private TaskRepository taskRepository;

    @GetMapping
    public List<Task> findAllTasks() {
        return taskRepository.findAll();
    }

    @GetMapping("/{id}")
    public Task findTask(@PathVariable long id) {
        return taskRepository.findById(id).orElseThrow(() -> new NotFoundException
(id));
    }

    @PostMapping
    @ResponseStatus(CREATED)
    public void createTask(@RequestBody Task task) {
        taskRepository.save(task);
    }

    @PutMapping("/{id}")
    public void updateTask(@PathVariable long id, @RequestBody Task task) {
        taskRepository.save(task);
    }

    @DeleteMapping("/{id}")
    @ResponseStatus(NO_CONTENT)
    public void deleteTask(@PathVariable long id) {
        taskRepository.deleteById(id);
    }
}
```

Runde 3 - REST Endpoints - MicroProfile

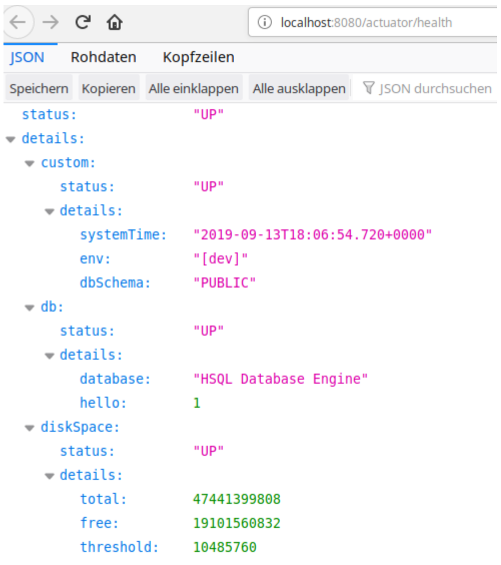
```
@Path("/tasks")
@Produces(MediaType.APPLICATION_JSON)
public interface TaskResource {

    @GET
    @Path("")
    List<Task> findAllTasks();

    @GET
    @Path("/{id}")
    Task findTask(
            @PathParam("id") Long id
    );

    @POST
    @Path("/{id}")
    void createTask(Task task);

    @PUT
    @Path("/{id}")
    void updateTask(
            @PathParam("id") Long id,
            Task task
    );

    @DELETE
    @Path("/{id}")
    void deleteTask(
            @PathParam("id") Long id
    );

}
```

Runde 4 - Health Check

Health Check

Runde 4 - Health Check - Spring

Actuator Health Endpoint:



```
status:            "UP"
details:
  custom:
    status:        "UP"
    details:
      systemTime:  "2019-09-13T18:06:54.720+0000"
      env:         "[dev]"
      dbSchema:    "PUBLIC"
  db:
    status:        "UP"
    details:
      database:    "HSQL Database Engine"
      hello:       1
  diskSpace:
    status:        "UP"
    details:
      total:       47441399808
      free:        19101560832
      threshold:   10485760
```

Runde 4 - Health Check - Spring

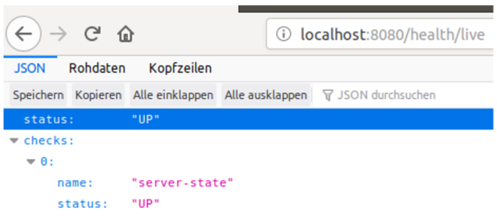
Eigene Health Indicators:

```
@Component
public class CustomHealthIndicator extends AbstractHealthIndicator {

    @Autowired
    private Environment environment;

    @Autowired
    private DataSource dataSource;

    @Override
    protected void doHealthCheck(Builder builder) throws Exception {
        builder.up()
                .withDetail("systemTime", new Date())
                .withDetail("env", Arrays.toString(environment.getActiveProfiles()))
                .withDetail("dbSchema", dataSource.getConnection().getSchema());
    }
}
```

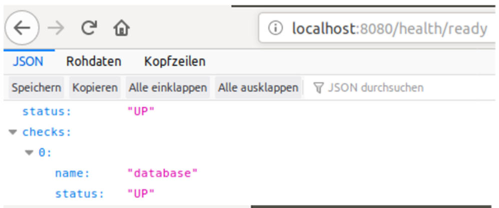
Runde 4 - Health Check - MicroProfile

MicroProfile Health Endpoint:

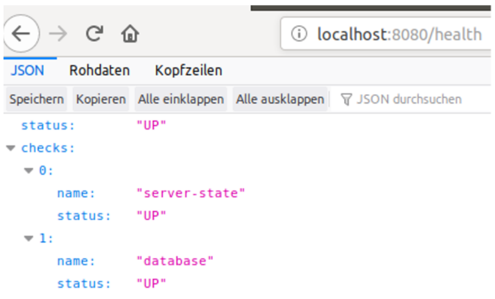
- `health/live` ⇒ **@Liveness**
- `health/ready` ⇒ **@Readiness**
- `health` ⇒ **@Liveness** & **@Readiness**



```
status:        "UP"
▼ checks:
  ▼ 0:
      name:     "server-state"
      status:   "UP"
```



```
status:        "UP"
▼ checks:
  ▼ 0:
      name:     "database"
      status:   "UP"
```

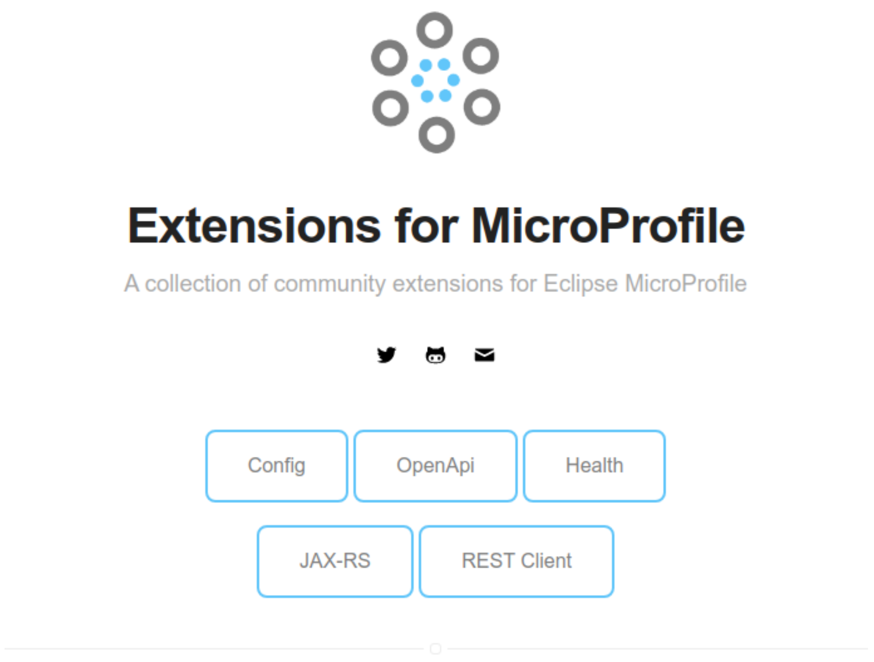


```
status:        "UP"
▼ checks:
  ▼ 0:
      name:     "server-state"
      status:   "UP"
  ▼ 1:
      name:     "database"
      status:   "UP"
```

Runde 4 - Health Check - MicroProfile

```java
@Liveness
@ApplicationScoped
public class LivenessChecks implements HealthCheck {

    @Override
    public HealthCheckResponse call() {
        ModelNode op = new ModelNode();
        op.get("address").setEmptyList();
        op.get("operation").set("read-attribute");
        op.get("name").set("suspend-state");

        try (ModelControllerClient client = ModelControllerClient.Factory.create(
"localhost", 9990)) {
            ModelNode response = client.execute(op);

            if (response.has("failure-description")) {
                throw new Exception(response.get("failure-description").asString());
            }

            boolean isRunning = response.get("result").asString().equals("RUNNING");
            if (isRunning) {
                return HealthCheckResponse.named("server-state").up().build();
            } else {
                return HealthCheckResponse.named("server-state").down().build();
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

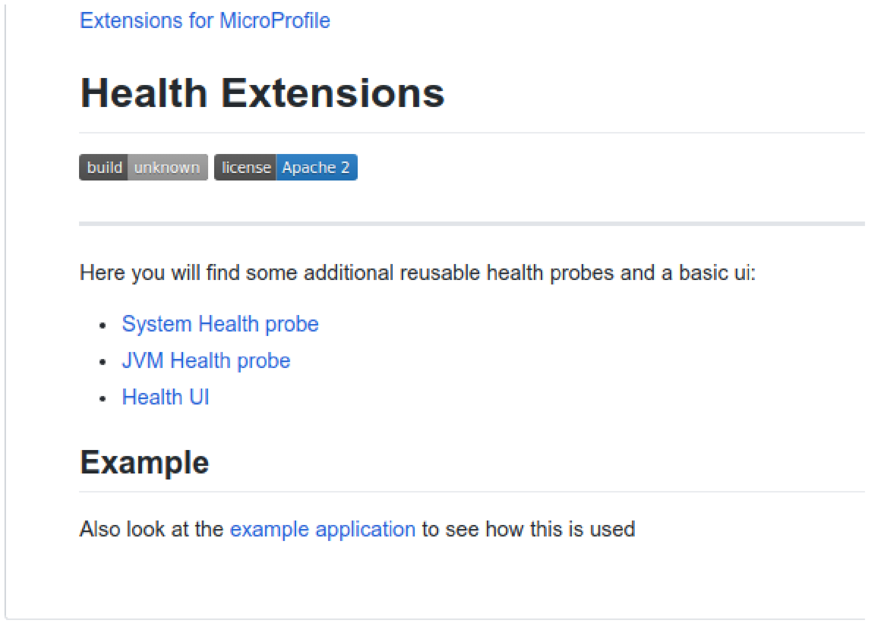
Runde 4 - Health Check - MicroProfile



Extensions for MicroProfile

A collection of community extensions for Eclipse MicroProfile

https://microprofile-ext.org



Extensions for MicroProfile

Health Extensions

Here you will find some additional reusable health probes and a basic ui:

- System Health probe
- JVM Health probe
- Health UI

Example

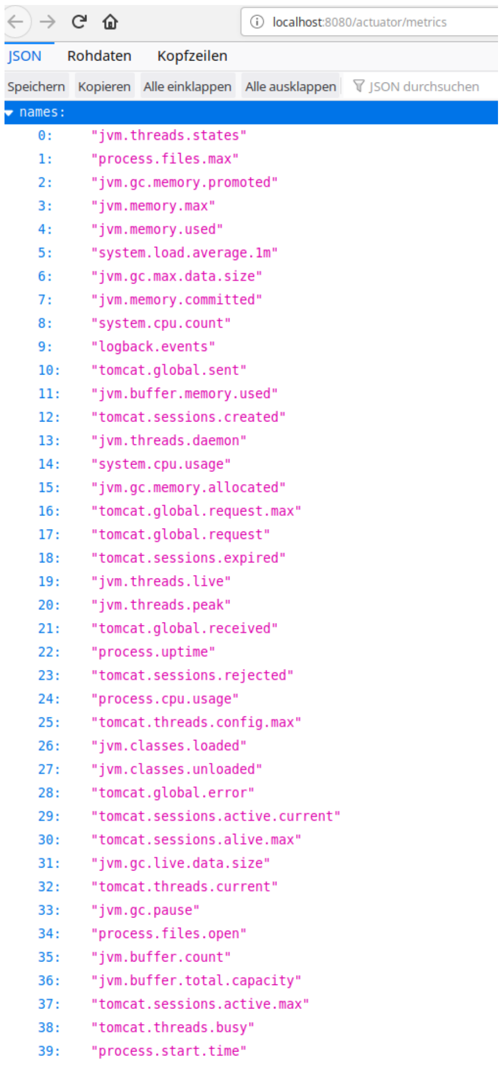
Also look at the example application to see how this is used

Runde 5 - Metriken

Metriken

Runde 5 - Metriken - Spring

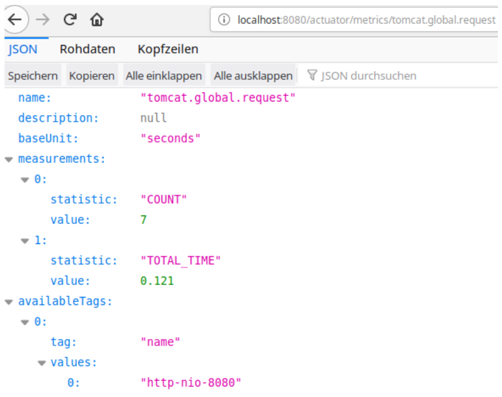
Actuator Metrics Endpoint:



```
names:
  0:   "jvm.threads.states"
  1:   "process.files.max"
  2:   "jvm.gc.memory.promoted"
  3:   "jvm.memory.max"
  4:   "jvm.memory.used"
  5:   "system.load.average.1m"
  6:   "jvm.gc.max.data.size"
  7:   "jvm.memory.committed"
  8:   "system.cpu.count"
  9:   "logback.events"
  10:  "tomcat.global.sent"
  11:  "jvm.buffer.memory.used"
  12:  "tomcat.sessions.created"
  13:  "jvm.threads.daemon"
  14:  "system.cpu.usage"
  15:  "jvm.gc.memory.allocated"
  16:  "tomcat.global.request.max"
  17:  "tomcat.global.request"
  18:  "tomcat.sessions.expired"
  19:  "jvm.threads.live"
  20:  "jvm.threads.peak"
  21:  "tomcat.global.received"
  22:  "process.uptime"
  23:  "tomcat.sessions.rejected"
  24:  "process.cpu.usage"
  25:  "tomcat.threads.config.max"
  26:  "jvm.classes.loaded"
  27:  "jvm.classes.unloaded"
  28:  "tomcat.global.error"
  29:  "tomcat.sessions.active.current"
  30:  "tomcat.sessions.alive.max"
  31:  "jvm.gc.live.data.size"
  32:  "tomcat.threads.current"
  33:  "jvm.gc.pause"
  34:  "process.files.open"
  35:  "jvm.buffer.count"
  36:  "jvm.buffer.total.capacity"
  37:  "tomcat.sessions.active.max"
  38:  "tomcat.threads.busy"
  39:  "process.start.time"
```

Runde 5 - Metriken - Spring

Actuator Metrics Endpoint (#Requests):



```
name:            "tomcat.global.request"
description:     null
baseUnit:        "seconds"
measurements:
  0:
    statistic:   "COUNT"
    value:       7
  1:
    statistic:   "TOTAL_TIME"
    value:       0.121
availableTags:
  0:
    tag:         "name"
    values:
      0:         "http-nio-8080"
```

Runde 5 - Metriken - MicroProfile

Metrics Endpoint:

- /metrics/base
- /metrics/vendor
- /metrics/application

Runde 5 - Metriken - MicroProfile

/base

```
{
"gc.total;name=G1 Young Generation1": 15,
"gc.total;name=G1 Old Generation1": 0,
"cpu.systemLoadAverage": 1.42,
"thread.count": 54,
"classloader.loadedClasses.count": 15789,
"classloader.unloadedClasses.total": 8,
"jvm.uptime": 529828,
"gc.time;name=G1 Young Generation1": 207,
"gc.time;name=G1 Old Generation1": 0,
"thread.max.count": 90,
"memory.committedHeap": 557842432,
"classloader.loadedClasses.total": 15797,
"cpu.availableProcessors": 8,
"thread.daemon.count": 9,
"memory.maxHeap": 4139778048,
"memory.usedHeap": 107376120
}
```

Runde 5 - Metriken - MicroProfile

/vendor

```
{
"bufferPool.usedMemory;name=mapped1": 0,
"bufferPool.usedMemory;name=direct1": 368640,
"memoryPool.usage.max;name=CodeHeap 'profiled nmethods'1": 16832896,
"memoryPool.usage.max;name=Compressed Class Space1": 11092256,
"memoryPool.usage.max;name=G1 Eden Space1": 333447168,
"memoryPool.usage.max;name=G1 Old Gen1": 57122512,
"memoryPool.usage.max;name=CodeHeap 'non-profiled nmethods'1": 6189184,
"memoryPool.usage.max;name=Metaspace1": 91788704,
"memoryPool.usage.max;name=G1 Survivor Space1": 25165824,
"memoryPool.usage.max;name=CodeHeap 'non-nmethods'1": 1357952,
"memoryPool.usage;name=CodeHeap 'non-profiled nmethods'1": 6189184,
"memoryPool.usage;name=Metaspace1": 91788704,
"memoryPool.usage;name=Compressed Class Space1": 11092256,
"memoryPool.usage;name=G1 Old Gen1": 46558712,
"memoryPool.usage;name=G1 Survivor Space1": 20971520,
"memoryPool.usage;name=CodeHeap 'profiled nmethods'1": 16832896,
"memoryPool.usage;name=CodeHeap 'non-nmethods'1": 1295872,
"memoryPool.usage;name=G1 Eden Space1": 39845888,
"loadedModules": 327
}
```

Runde 5 - Metriken - MicroProfile

/application

- @Counted
- @Gauge
- @Metered
- @Timed

Runde 5 - Metriken - MicroProfile

@Counted

```
@Counted(unit = MetricUnits.NONE,
        name = "tasksCreated",
        absolute = true,
        displayName = "Created items",
        description = "Metrics to show how many times createTask method was called.
",
        tags = {"tasks=create"}
        )
@POST
@Path("/{id}")
public void createTask(Task task) {
    ...
}
```

/application/tasksCreated

```
{
"tasksCreated;tasks=create": 53
}
```

Runde 5 - Metriken - MicroProfile

@Gauge

```
@Inject
@ConfigProperty(name="defaultEstimation")
private Long defaultEstimation;

@Gauge(unit = "Hour", name = "defaultEstimation", absolute = true)
public Long getDefaultEstimation() {
    return defaultEstimation;
}
```

/application/defaultEstimation

```
{
 "defaultEstimation": 5
}
```

Runde 5 - Metriken - MicroProfile

@Metered

```
@Metered(
        name = "findTask",
        unit = MetricUnits.MINUTES,
        description = "Metrics to monitor findTask method.",
        absolute = true
)
@GET
@Path("/{id}")
public Task findTask(
        @PathParam("id") Long id
) {
    ....
};
```

/application/findTask

```
{
    "findTask": {
        "count": 8,
        "meanRate": 0.10400404006688957,
        "oneMinRate": 0.11417125483023463,
        "fiveMinRate": 0.025847358928386722,
        "fifteenMinRate": 0.00879681999435735
    }
}
```

Runde 5 - Metriken - MicroProfile

@Timed

```
@Timed(name = "findAllTasks",
        description = "Metrics to monitor the times of findAllTasks method.",
        unit = MetricUnits.SECONDS,
        absolute = true)
@GET
@Path("")
public List<Task> findAllTasks() {
    ....
}
```

/application/findAllTasks

```
{
  "findAllTasks" : {
    "min": 3.62E-6,
    "mean": 1.3103301859534476E-5,
    "max": 1.66379E-4,
    "stddev": 2.893381453028447E-5,
    "count": 41,
    "meanRate": 0.23552131518346484,
    "oneMinRate": 0.5193839909881481,
    "fiveMinRate": 0.12930613497839835,
    "fifteenMinRate": 0.044721333247577794
  }
}
```

Kurzes Fazit

MicroProfile

- Sammlung erprobter Enterprise-Standards
- Gesammeltes Wissen und Know-How
- Speziell auf Microservices zugeschnitten

- Kostenersparnis
- Teilweise unflexibel
- Eine Liebes/Hass Beziehung

Spring

- Extrem mächtiges Ökosystem
- Minimale Konfiguration
- Vorreiter
- Bewährte Technologie (auch) für Microservices
- Leichte Integration anderer Frameworks (Starter)…
- … aber auch Abhängigkeit davon

Wer ist der Sieger?

> Ja gut, es gibt nur eine Möglichkeit: Sieg, Unentschieden oder Niederlage
>
> — Franz Beckenbauer

Wie seht ihr das?

Links

- MicroProfile Dokumentation: https://microprofile.io
- Quarkus Dokumentation: https://quarkus.io/
- Spring Dokumentation: https://spring.io
- Folien: https://github.com/goldschmiede/2020-06-26-MicroProfile-vs-Spring

Ende

Vielen Dank!

@anderScoreGmbH anderScore.company Java_Meetup_anderscore