Open Java EE and Eclipse
MicroProfile - A New Java
Landscape for Cloud Native Apps

Kevin Sutter – MicroProfile and Java EE Architect
@kwsutter

Emily Jiang – MicroProfile Development Lead
@emilyfhjiang
MicroProfile Sessions

- **Open Java EE and Eclipse MicroProfile – A New Java Landscape for Cloud Native Apps**
  - *(old title)* MicroProfile, Java EE, and the Application Server – Oh My!
  - Wednesday, Oct 25, 15:45, Burgersaal 2
- **Resilient Java Microservices with Eclipse MicroProfile**
  - *(old title)* Eclipse MicroProfile: What is it? Is it a Java EE Replacement?
  - Wednesday, Oct 25, 16:30, Silchersaal
- **MicroProfile BOF**
  - Wednesday, Oct 25, 18:30, Seminarraum 5
- **The Monitoring Aspects of Eclipse MicroProfile**
  - Thursday, Oct 26, 11:00, Seminarraum 5
- **Eclipse MicroProfile Config and OSGi Config Admin**
  - Thursday, Oct 26, 13:00, Schubartsaal
MICROPROFILE 1.2 - NEW ENTERPRISE CAPABILITIES FOR MICROSERVICES

- **Configuration**
  - Externalize configuration to improve portability

- **Health check**
  - Common format to determine service availability

- **Fault tolerance**
  - Build robust behavior to cope with unexpected failures

- **JWT Propagation**
  - Interoperable authentication and role-based access control

- **Metrics**
  - Common REST endpoints for monitoring service health

Eclipse MicroProfile 1.2:
- https://projects.eclipse.org/projects/technology.microprofile

Eclipse Enterprise for Java:
- https://projects.eclipse.org/projects/ee4j/charter
**MicroProfile 1.0 (Fall 2016)**
- jaxrs-2.0
- cdi-1.2
- jsonp-1.0

**MicroProfile 1.1 (August 2017)**
- microProfile-1.0
- mpConfig-1.0

**MicroProfile 1.2 (Sept 2017)**
- microProfile-1.1
- mpConfig-1.1
- mpFaultTolerance-1.0
- mpHealth-1.0
- mpMetrics-1.0
- mpJwt-1.0

**MicroProfile 1.3 (???)**
- MicroProfile 1.2
- mpTracing-1.0
- mpOpenApi-1.0
- mpRestClient-1.0

**MicroProfile 2.0 (???)**
- MicroProfile 1.3
- jaxrs-2.1 // Java EE 8
- cdi-2.0 // Java EE 8
- jsonp-1.1 // Java EE 8
- jsonb-1.0 // Java EE 8
Eclipse MicroProfile 1.2 (JavaOne / Sept 2017)

- Health Check 1.0
- Metrics 1.0
- Fault Tolerance 1.0
- JWT Propagation 1.0
- Config 1.1
- CDI 1.2
- JSON-P 1.0
- JAX-RS 2.0

Legend:
- **= New
- = Updated
- = No change from last release
Eclipse MicroProfile: Microservice innovation

- Vendor-neutral programming model, designed in the open, for Java microservices
- Provide core capabilities for building fault tolerant, scalable microservices
- Increasing the rate and pace of innovation beyond Java EE

<table>
<thead>
<tr>
<th>Config</th>
<th>Fault Tolerance</th>
<th>Health Check</th>
<th>Metrics</th>
<th>JWT Propagation</th>
</tr>
</thead>
<tbody>
<tr>
<td>externalize configuration</td>
<td>build robust behavior</td>
<td>common format to determine service</td>
<td>common REST endpoints for monitoring service</td>
<td>interoperable authentication and role-based access control</td>
</tr>
<tr>
<td>to improve portability</td>
<td>to cope with unexpected failures</td>
<td>availability</td>
<td>health</td>
<td></td>
</tr>
</tbody>
</table>

**MicroProfile v1.2 – Released September 2017**
http://microprofile.io/

New in Liberty 17.0.0.3
The property file, microprofile-config.properties, packaged in the application can be overwritten by:

- System variables (400 as the default priority)
- Environment variables (300 as the default priority) or
- a custom property files with a higher priority than microprofile-config.properties (100 as the default priority).

- Write **once**
- Containerize **once**
- Liberty **injects** config from outside the container
- **Portable** across environments
Application annotations for Timeout, Retry, CircuitBreaker, Bulkhead, Fallback policies
- Policies configured by properties
- Defined on methods invoking other services; handled by CDI interceptors.
  - Asynchronous: Invoke the method asynchronously
    - @Asynchronous
  - Retry: invoke same method multiple times
    - @Retry(maxRetries=3, delay=400, maxDuration=2000)
  - Timeout: Specify a timeout for the method invocation
    - @Timeout(300)
  - CircuitBreaker: open a circuit so that the following invocation will fail immediately
    - @CircuitBreaker(delay=200, failureRatio=0.2)
  - Bulkhead: limit the concurrent requests
    - @Bulkhead(20) – semaphore model, at most 20 concurrent requests
    - @Asynchronous @Bulkhead(value=20, waitingTaskQueue=30): thread pool style, at most 20 concurrent threads
  - Fallback: a secondary service when the primary service invocation fails
    - @Fallback(MyFallbackHandler.class) - invoke MyFallbackHandler.handle on exception
    - @Fallback(fallbackMethod="fbMethod") - invoke fbMethod on the same instance as the method with this annotation
### Eclipse MicroProfile: A Closer Look

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Config</td>
<td>Fault Tolerance</td>
<td>Health Check</td>
<td>Metrics</td>
<td>JWT Propagation</td>
</tr>
<tr>
<td>externalize configuration to improve portability</td>
<td>build robust behavior to cope with unexpected failures</td>
<td>common format to determine service availability</td>
<td>common REST endpoints for monitoring service health</td>
<td>interoperable authentication and role-based access control</td>
</tr>
</tbody>
</table>

- **Exposes** `/health` default endpoint for the server/container if feature enabled
  - Standard API for optional application-specific implementation
  - Can be used with Kubernetes liveness check yaml
- **Exposes** `/metrics` endpoint for the server/container if feature enabled
  - Exposes system, vendor and app-specific metrics
    - OOB metrics include stats about:
      - JVM memory
      - Garbage Collection
      - JVM uptime
      - Threads
      - Thread Pools (stretch goal)
      - ClassLoading
      - CPU usage and availability
  - Response in JSON (for collection from collectd or other JSON-friendly tools) and Prometheus text formats
  - App metrics can be provided using Dropwizard-based API or new CDI-enabled annotations

```java
@Timed(name="thinkTime", absolute=true)
void someImportantThing() {
    // method logic here...
}

@Gauge(name="myGauge", absolute=true)
double somethingToTrack() {
    return myValue;
}
```

GET `/metrics`

# HELP base:cpu_availableProcessors number of processors available to the Java virtual machine
# TYPE base:cpu_availableProcessors gauge
base:cpu_availableProcessors 8

# HELP base:memory_committedHeap amount of memory in bytes that is committed for the JVM to use
# TYPE base:memory_committedHeap gauge
base:memory_committedHeapMemory 41287680

# TYPE application:thinkTime_count counter
application:thinkTime_count 944534

# TYPE application:myGauge gauge
application:myGauge 0.1834432
**MicroProfile** : A Closer Look

<table>
<thead>
<tr>
<th>Config</th>
<th>Fault Tolerance</th>
<th>Health Check</th>
<th>Metrics</th>
<th>JWT Propagation</th>
</tr>
</thead>
<tbody>
<tr>
<td>externalize configuration to improve portability</td>
<td>build robust behavior to cope with unexpected failures</td>
<td>common format to determine service availability</td>
<td>common REST endpoints for monitoring service health</td>
<td>interoperable authentication and role-based access control</td>
</tr>
</tbody>
</table>

### Flow Diagram:

**Client**

- Any http client or authn proxy

- Request service with JWT

- Make authz decision based on JWT claims

- Service verifies JWT & create JsonWebToken & subject

- Service authorizes request with JsonWebToken

<table>
<thead>
<tr>
<th>Client</th>
<th>Micro Services (MS 1)</th>
<th>Micro Services (MS 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Liberty or other Java EE containers</td>
<td>Liberty or other Java EE container</td>
</tr>
</tbody>
</table>

- JsonWebToken is accessible via CDI or jax-rs SecurityContext. Use JWT for additional authorization, or propagate jwt to another service

**Notes:**

1. Client has JWT, and use it to request service over http header
2. Service verifies JWT & create JsonWebToken & subject
3. Service authorizes request with JsonWebToken
Eclipse MicroProfile 1.3 (4Q 2017 ???)

Roadmap

- Open Tracing 1.0
- Fault Tolerance 1.0
- CDI 1.2
- Open API 1.0
- Metrics 1.0
- JSON-P 1.0
- Rest Client 1.0
- JWT Propagation 1.0
- JAX-RS 2.0
- Health Check 1.0
- Config 1.1

MicroProfile 1.3

- New
- No change from last release
Eclipse MicroProfile 2.0 (1Q 2018 ?)

Roadmap

- Open Tracing 1.0
- Open API 1.0
- Rest Client 1.0
- JSON-B 1.0
- Fault Tolerance 1.0
- Metrics 1.0
- JWT Propagation 1.0
- Health Check 1.0
- CDI 2.0
- JSON-P 1.1
- JAX-RS 2.1
- Config 1.1

**MicroProfile 2.0**

- New
- No change from last release
Eclipse Enterprise for Java (EE4J)
Moving Java EE to Eclipse Foundation

Technology

Sponsorship

- Nimble
- Flexible
- Open
- Compatible

Community and Vendors
Eclipse Enterprise for Java (EE4J)
Moving Java EE to Eclipse Foundation

• Open process
• Collaboration: community, vendors, Eclipse
• Transition to EE4J in CY2018
  • GlassFish 5.0/Java EE 8 RIs, TCKs, product docs
  • Process for existing and new specs
  • Compatibility process
• Technology evolution, MicroProfile innovation
• Oracle Java EE Support through Java EE 8
  • Continuity for Java EE community
Accelerating Adoption of Microservices
Open Liberty

A lightweight open source server runtime ideal for building Java™ microservices and cloud-native apps

- Easy to consume
- Deploy on any cloud for Java™
- Seamlessly transition to WebSphere

https://openliberty.io/
• Resources

• http://microprofile.io/

• https://openliberty.io/

• https://www.eclipse.org/community/eclipse_newsletter/2017/september/

• https://github.com/NottyCode/fault-tolerance-examples

• https://liberty-app-accelerator.wasdev.developer.ibm.com/start/
Thank you
OpenAPI (MicroProfile 1.3 proposed content)
OpenTracing (MicroProfile 1.3 proposed content)

Minikube or ICP environment

MicroserviceBuilder microserviceA
Distributed tracing instrumentation

MicroserviceBuilder microserviceB
Distributed tracing instrumentation

Zipkin service

MicroserviceBuilder fabric

Zipkin console
  microserviceA
  microserviceB