

Java Knights in Varna

Eclipse MicroProfile Past, Present & Future



14.11.2019

BULPROS

Who am I?

Java Developer

I'm writing java code for the last 16 years

1

Software developer

For the last 30 years

2

Java Enthusiast

- Java.beer in Plovdiv
- Part of Bulgarian Java User Group board
- I'm one of the organizers of jPrime conference
- I'm the organiser of the conference JProfessionals in Plovdiv

3

Member of JCP

I try to influence the direction where Java goes by participating in the elections of new JCP EC members every year for the last 10 years.

4

Agenda

- What is MicroProfile
- Short history of Microprofile with examples
- More information about MicroProfile

What is Microprofile?

- Initially started as initiative in 2016 by



Red Hat

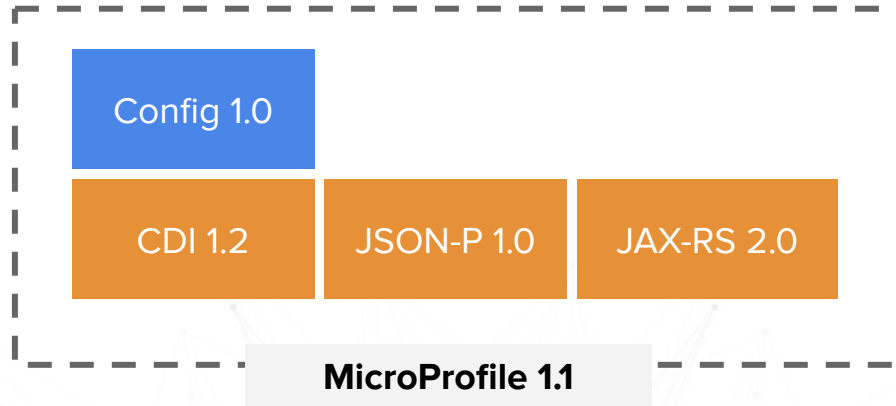




- Main goal was to try and leverage Java EE technologies to create vendor-neutral microservice framework



Microprofile history

- Later in December 2016 Microprofile became an Eclipse project
- And in August 2017 new version of the specification was released



 = New
 = No change from last release

What is Config 1.0

- An API that is used to configure the application using externally provided parameters
- Actual implementations are provided by multiple vendors

```
@Singleton
@Path("/hello")
public class HelloResource {

    @Inject
    @ConfigProperty(name = "who", defaultValue = "world")
    private String who;

    @GET
    public String hello() {
        return "Hello " + who;
    }
}
```

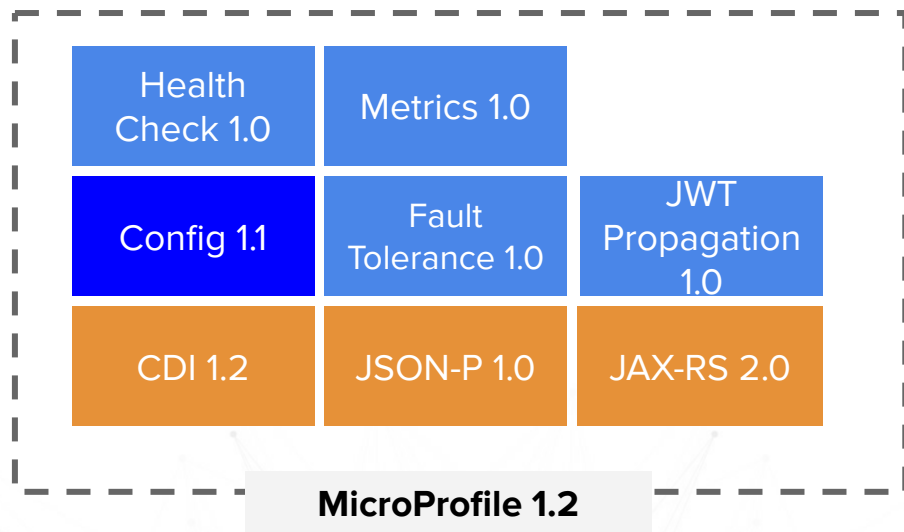
```
public class Config {

    public static void main(String[] args) {
        String who =
            ConfigProvider.getConfig().getOptionalValue("who", String.class).orElse("world");

        System.out.println("Hello " + who);
    }
}
```

Microprofile history

- In September 2017 Microprofile 1.2 was released



- = New
- = Updated
- = No change from last release

What is Metrics

- Specification that defines well known endpoints to monitor essential system parameters
- It provides access to **base**, **vendor** and **application** sets of parameters

```
@Singleton
@Path("/hello")
public class HelloResource {

    @Inject
    @ConfigProperty(name = "who", defaultValue = "world")
    private String who;

    @GET
    @Counted(name="hello counter", absolute = true, monotonic = true)
    public String hello() {
        return "Hello " + who;
    }
}
```


What is Health

- Health checks are used to probe the state of a computing node from another machine like Kubernetes service controller

```
@ApplicationScoped
@Health
public class HelloHealthCheck implements HealthCheck {

    @Inject
    MetricRegistry metricRegistry;

    @Override
    public HealthCheckResponse call() {
        long hello_count = metricRegistry.getCounters((n, m) -> n.endsWith("hello counter"))
            .values()
            .stream()
            .map(Counter::getCount)
            .reduce(0L, Long::sum);

        return HealthCheckResponse.builder()
            .state(hello_count > 0)
            .name("hello health")
            .withData("total hello counter", hello_count)
            .build();
    }
}
```

What is Fault Tolerance

- Timeout, Fall back, Circuit Breakers, Retry

```
@Singleton
@Path("/hello")
public class HelloResource {

    private Random r = new Random(System.currentTimeMillis());

    @GET
    public String hello() {
        return "Fast hello world";
    }

    @GET
    @Path("/slow")
    @Timeout(500)
    @Fallback(fallbackMethod = "fallback")
    @Counted(name = "slow", monotonic = true)
    public String slowHello() {
        try { Thread.sleep(r.nextInt(1000)); } catch (InterruptedException ignored) {}
        return "Slow hello world";
    }
}
```

```
@Counted(name = "fallback", monotonic = true)
public String fallback() {
    return "fallback hello world";
}
```

What is JWT Propagation

- This specification outlines a proposal for using OpenID Connect(OIDC) based JSON Web Tokens(JWT) for role based access control(RBAC) of microservice endpoints.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}
```

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret  
) base64 encoded
```

What is JWT Propagation

```
@RequestScoped
@Path("/hello")
@Produces(MediaType.TEXT_PLAIN)
public class HelloResource {
```

```
    @Inject
    private JsonWebToken jsonWebToken;
```

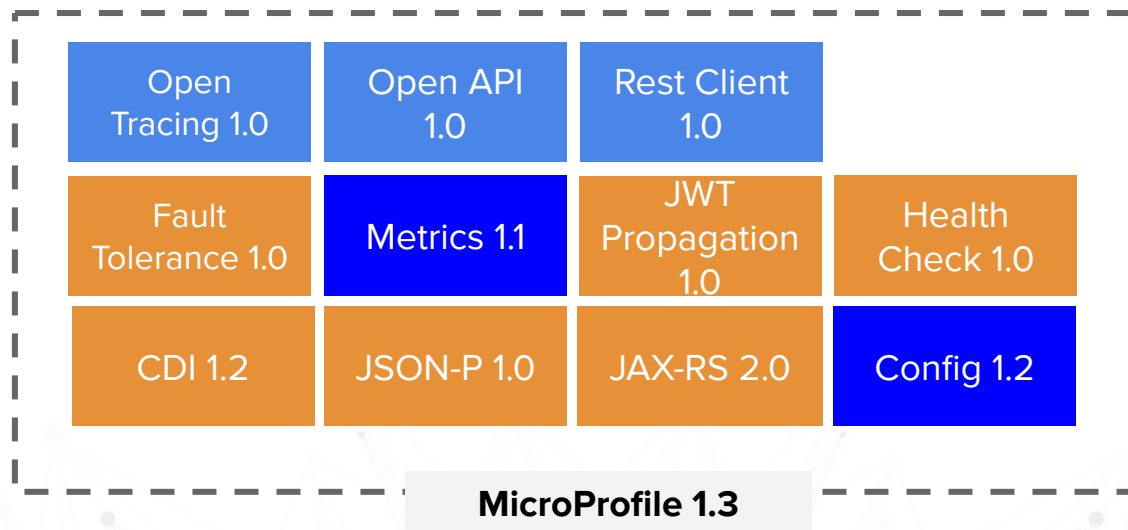
```
    @GET
    @Counted(name="hello counter jwt", absolute = true, monotonic = true)
    @RolesAllowed({"authenticated"})
    public String jwtHello() {
        return "Hello " + jsonWebToken.getName();
    }
}
```

```
    @GET
    @Path("/nojwt")
    @Counted(name="hello counter no jwt", absolute = true, monotonic = true)
    @PermitAll
    public String noAuthentication() {
        return "Hello unauthenticated";
    }
}
```

```
@ApplicationPath("/")
@loginConfig(authMethod = "MP-JWT", realmName = "jwt-app")
@DeclareRoles("authenticated")
public class JWTApplication extends Application {}
```

Microprofile history

- In January 2018 Microprofile 1.3 was released
- And in June 2018 1.4 was released - no new API, only updates to some of the existing one



- = New
- = Updated
- = No change from last release

What's new in 1.3

- Open Tracing

```
@Singleton
@Path("/hello")
public class HelloResource {

    @Inject
    @ConfigProperty(name = "who", defaultValue = "world")
    private String who;

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    @Traced
    public String hello() {
        return "Hello " + who;
    }
}
```

- Open API

```
.....

<dependency>
  <groupId>org.microprofile-ext.openapi-ext</groupId>
  <artifactId>swagger-ui</artifactId>
  <version>1.0.2</version>
</dependency>

.....
```

```
@GET
@Counted(name="hello counter", absolute = true, monotonic = true)
@Produces(MediaType.TEXT_PLAIN)
@Operation(description = "returns \"Hello {who}\" where {who} is provided by
configuration or \"world\" if who is not provided")
public String hello() {
    return "Hello " + who;
}
```

What is REST Client

- The MicroProfile Rest Client provides a type-safe approach to invoke RESTful services over HTTP

```
@Path("/movies")
@registerRestClient
public interface MovieReviewService {

    @GET
    Set<Movie> getAllMovies();

    @GET
    @Path("/{movieId}/reviews")
    Set<Review> getAllReviews(@PathParam("movieId") String movieId);

    @GET
    @Path("/{movieId}/reviews/{reviewId}")
    Review getReview(@PathParam("movieId") String movieId, @PathParam("reviewId") String reviewId);

    @POST
    @Path("/{movieId}/reviews")
    String submitReview(@PathParam("movieId") String movieId, Review review);

    @PUT
    @Path("/{movieId}/reviews/{reviewId}")
    Review updateReview(@PathParam("movieId") String movieId, @PathParam("reviewId") String reviewId, Review review);
}
```

What is REST Client

Use plain API

```
private void consumeRestService() throws URISyntaxException {  
    URI apiUri = new URI("http://localhost:9080/movieReviewService");  
    MovieReviewService reviewSvc = RestClientBuilder.newBuilder()  
        .baseUri(apiUri)  
        .build(MovieReviewService.class);  
    Review review = new Review(3 /* stars */, "This was a delightful comedy, but not terribly realistic.");  
    reviewSvc.submitReview( movieId, review );  
}
```

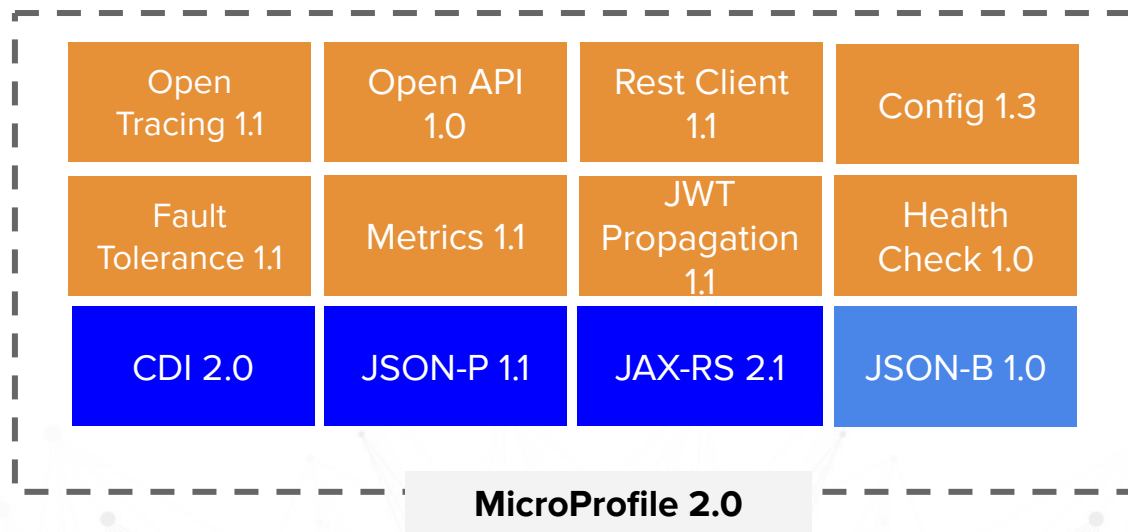
```
@Inject  
@RestClient  
private MovieReviewService movieReviewService;
```

```
private void consumeRestService() {  
    Review review = new Review(3 /* stars */, "This was a delightful comedy, but not terribly realistic.");  
    movieReviewService.submitReview( movieId, review );  
}
```

Use with CDI

Microprofile history

- In June 2018 also 2.0 was released - Upgrade to Java EE 8 versions of the core API's



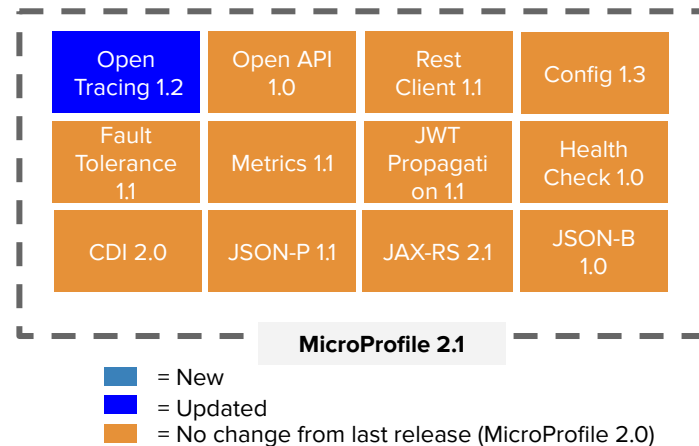
-  = New
-  = Updated
-  = No change from last release (MicroProfile 1.4)

Microprofile history

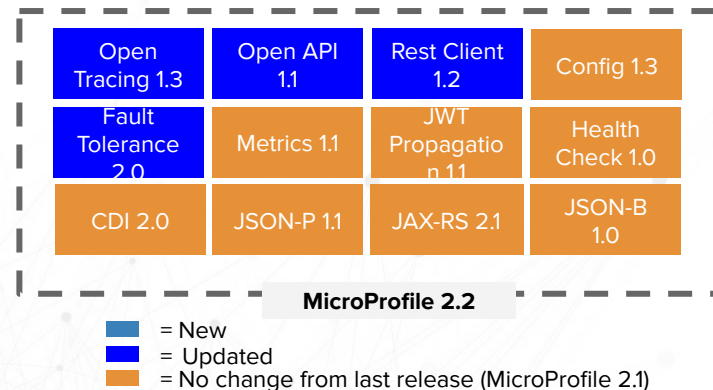
- Main changes in Microprofile 2.0
 - Align with Java EE 8 API's
 - Set minimum Java SE 8

Microprofile history

- In Oct 2018 2.1 was released

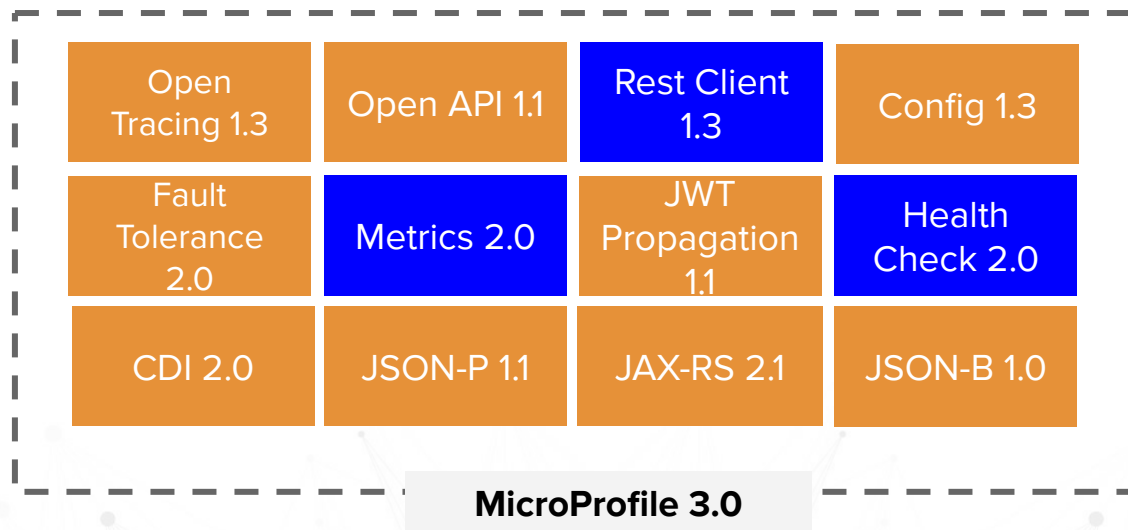


- In Feb 2019 2.2 was released



Microprofile history

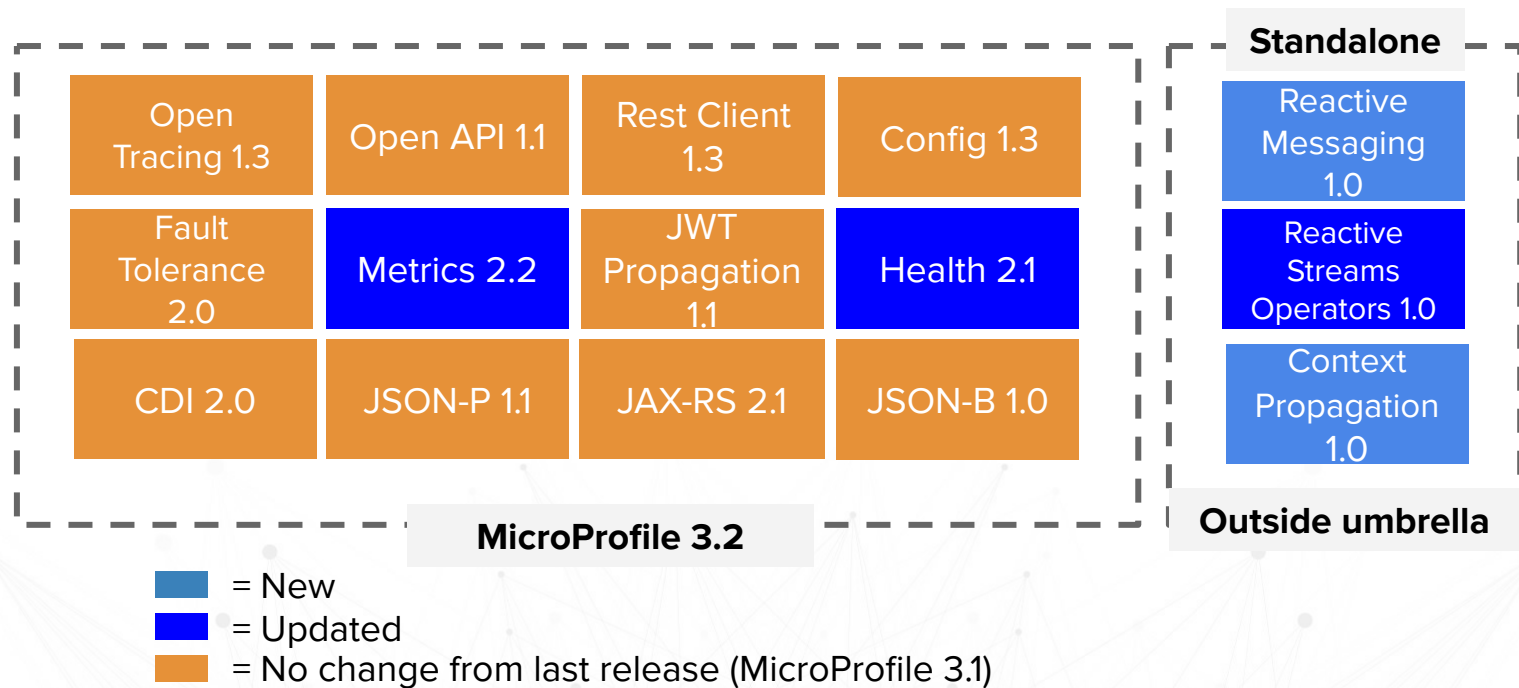
- In June 2019 3.0 was released. Main reason is that some of the API's contain breaking changes



- = New
- = Updated
- = No change from last release (MicroProfile 2.2)

Microprofile history

- In November 2019 version 3.2 will be released



Microprofile Reactive Capabilities

MicroProfile Reactive Streams Operators

A set of operators to create new reactive streams, process the transiting data and consume them with ease

MicroProfile Reactive Messaging

Defines a development model for declaring CDI *beans* producing, consuming and processing messages. It relies on Reactive Streams Operators and CDI

MicroProfile Context Propagation

APIs for propagating contexts across units of work that are thread-agnostic

Current Microprofile implementations



QUARKUS

BULPROS

Roadmap

- Long Running Actions
- GraphQL
- Reactive Relational Database Access
- Event Data
- Service Meshes

How you can contribute?

- Review individual specifications
- Propose changes in the SPEC or fix bugs
- Participate in the discussions
 - <https://groups.google.com/forum/#!forum/microprofile>

Microprofile Resources

- MicroProfile web site - <https://microprofile.io>
- MicroProfile Starter - <https://start.microprofile.io/>
- Wiki page - <https://wiki.eclipse.org/MicroProfile>
- GitHub repository - <https://github.com/eclipse/microprofile>

You can also check individual repositories of the MicroProfile specifications at GitHub

- <https://github.com/eclipse>
- Demo projects - <https://github.com/doychin/java-knights>
- BG JUG Hands-On-Lab - <https://github.com/bgjug/microprofile-hol-1x>

Q & A