MicroProfile

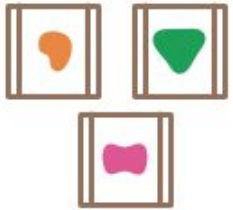
microservices made easy

whoami

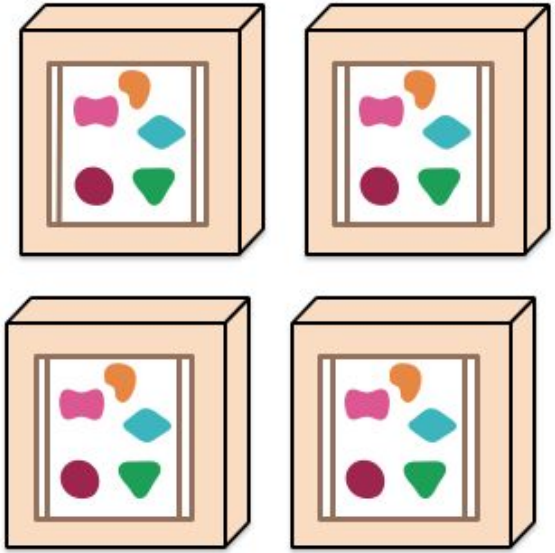
- Martin Štefanko
- Software engineer 3+ years, Red Hat
- MicroProfile contributor
- @xstefank

A monolithic application puts all its functionality into a single process...

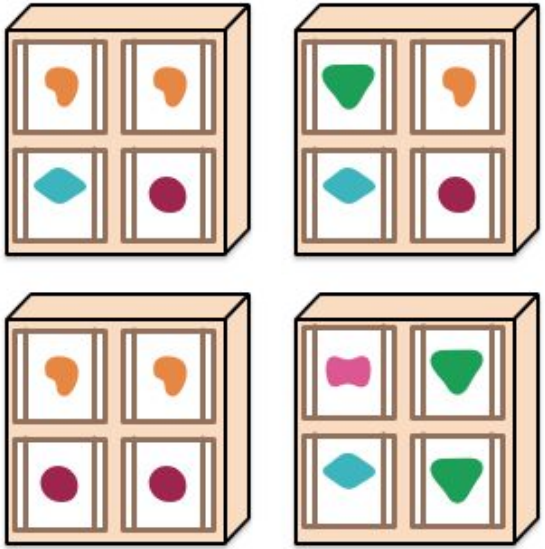
A microservices architecture puts each element of functionality into a separate service...



... and scales by replicating the monolith on multiple servers



... and scales by distributing these services across servers, replicating as needed.



https://martinfowler.com/articles/microservices.html

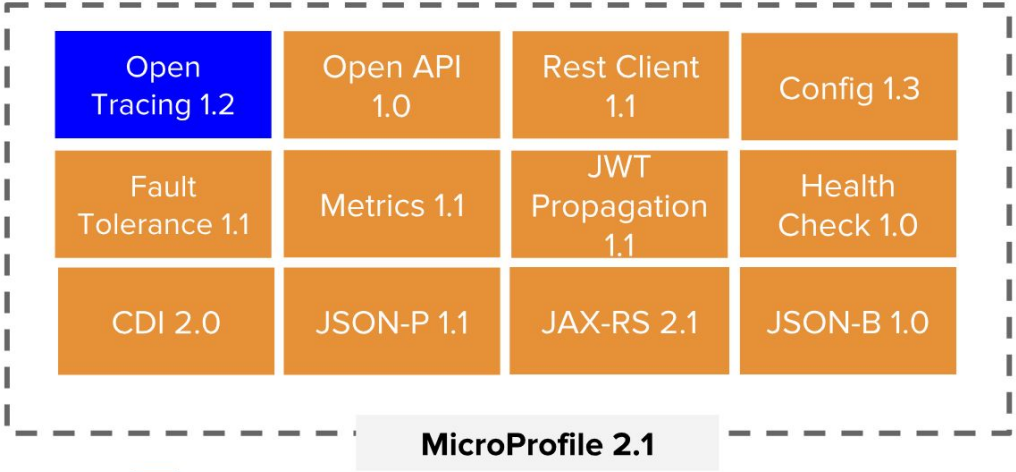
Enterprise Java in past 20 years

- Java EE (currently Jakarta EE)
  - Java EE 5 - May 11, 2006
  - Java EE 6 - December 10, 2009
  - Java EE 7 - June 12, 2013
  - Java EE 8 - August 31, 2017

MicroProfile

- Eclipse MicroProfile is an **open-source** community **specification** for Enterprise Java microservices
- A community of **individuals**, **organizations**, and **vendors** collaborating within an open source (Eclipse) project to bring microservices to the Enterprise Java community

Eclipse MicroProfile 2.1 (Oct, 2018)



= New
= Updated
= No change from last release (MicroProfile 2.0)

+Under discussion

- Long Running Actions (LRA)
- Reactive Streams Operators
- Reactive messaging
- Service mesh
- Concurrency
- GraphQL
- ...

Community - individuals, organizations, vendors

Current MicroProfile implementations

Differences from Java EE

- open source and open community
- code first approach
- 3 releases per year (Feb, Jun, Oct)
  - MP 1.0 - Sep 2016
  - MP 1.1 - Aug 2017
  - MP 1.2 - Sep 2017
  - MP 1.3 - Jan 2018
  - MP 1.4 / MP 2.0 - Jun 2018
  - MP 2.1 - Oct 2018

MicroProfile 2.2

New issue

Due by February 06, 2019 **0%** complete

The MicroProfile 2.2 release is targeted for Feb 2019 (with additional releases in June and October of 2019). We will use this Milestone to help track the content for this platform release. The expected Release Announce date will be Tuesday, Feb 12.

8 Open 0 Closed

- **Update MicroProfile spec for 2.2 release**
  #77 opened on Nov 27, 2018 by kwsutter
- **Include Reactive Operators 1.0**
  #75 opened on Nov 27, 2018 by kwsutter — 1
- **Include OpenTracing 1.3**
  #72 opened on Nov 27, 2018 by kwsutter — 2
- **Include Metrics 2.0**
  #71 opened on Nov 27, 2018 by kwsutter — 2
- **Include Rest Client 1.2**
  #70 opened on Nov 27, 2018 by kwsutter — 2
- **Include Health Check 2.0**
  #69 opened on Nov 27, 2018 by kwsutter
- **Include OpenAPI 1.1**
  #68 opened on Nov 27, 2018 by kwsutter — 4
- **Include Fault Tolerance 2.0**
  #67 opened on Nov 27, 2018 by kwsutter — 2

MICROPROFILE™
OPTIMIZING ENTERPRISE JAVA

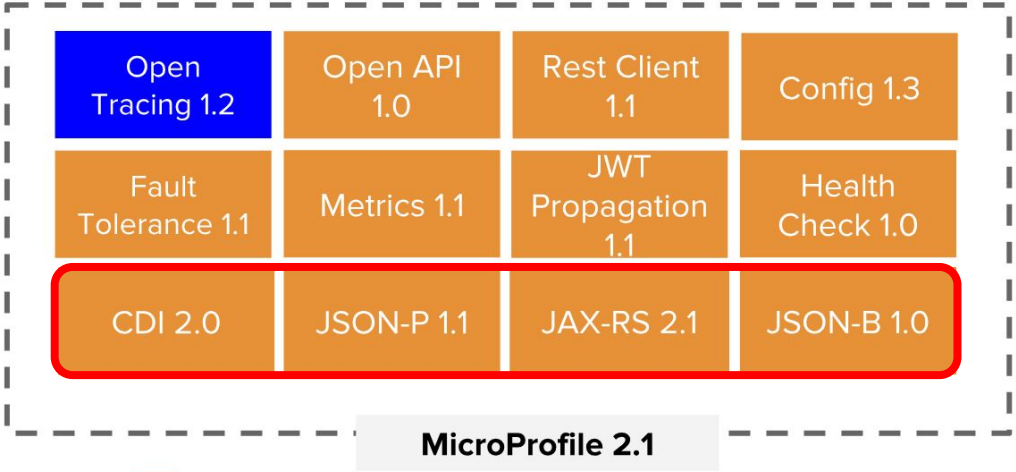
Projects Resources FAQ Blog Contributors Join The Discussion See The Code

Eclipse MicroProfile

Optimizing Enterprise Java for a Microservices Architecture

microprofile.io

Eclipse MicroProfile 2.1 (Oct, 2018)



= New
= Updated
= No change from last release (MicroProfile 2.0)

JAX-RS

```
@ApplicationPath("/api")
public class ApplicationConfig extends Application {

}
```

host:port/api/path1/path2/…

```java
@Path("/")
@Consumes(MediaType.APPLICATION_JSON) @Produces(MediaType.APPLICATION_JSON)
public class ProfileService {

    @POST
    public Response logEvent(...) {
        return Response.accepted(event).build();
    }

    @GET
    @Path("user/{userId}")
    public Response getUserEvents @PathParam("userId") int userId) {

        try {
            validateMembership(userId);
        } catch (NotFoundException nfe){
            return Response.status(Status.PRECONDITION_FAILED).header(REASON, "Membership [" + userId + "] does not
exist").build();
        }
        return eventSearcher.search(UserEventConverter.USER_ID,userId,size);
    }

  ...
}
```

JSON-P

```
JsonObject json = Json.createObjectBuilder()
                  .add("name", "Iron Man")
                  .add("realName", "Tony Start")
                  .add("alive", "true")
                  .build();
```

JSON-P

- Creating, reading and writing JSON
- Parsing JSON
- Stream support
- JSON pointers
- JSON patching

JSON-B

```java
public class Membership implements Serializable {
    private int membershipId;
    private Person owner;
    private Type type;
}

public class Person implements Serializable {
    private int id;
    private List<String> names;
    private String surname;
    private String email;
}
```

```json
{
    "membershipId": 4,
    "owner": {
        "email": "minki@gmail.com",
        "id": 4,
        "names": [
            "Minki"
        ],
        "surname": "van der Westhuizen"
    },
    "type": "FREE"
}
```

JSON-B

```
Jsonb jsonb = JsonbBuilder.create();
String json = jsonb.toJson(Avenger.name("Iron Man")
        .realName("Tony Stark")
        .alive(true)
        .build());

Avenger ironMan = jsonb.fromJson(json, Avenger.class);
```

```java
@Path("/")
@Consumes(MediaType.APPLICATION_JSON) @Produces(MediaType.APPLICATION_JSON)
public class MembershipService {

    @GET
    public List<Membership> getAllMemberships() {
        ...
    }

    @GET
    @Path("{id}")
    public Membership getMembership(@NotNull @PathParam(value = "id") int id) {
        ...
    }
}
```

CDI

```
@RequestScoped
public class EventLogger {
    @Inject
    private Client client;

    @Inject @Successful
    private Event<UserEvent> successfulBroadcaster;

    public Future<Void> logEvent(String token, @NotNull UserEvent event){
        IndexResponse response = client.prepareIndex(IndexDetails.PROFILING_INDEX,
IndexDetails.TYPE).setSource(json.toString(), XContentType.JSON).get();

        if(response.status().getStatus() == 201){
            successfulBroadcaster.fire(event);
        }

        return CompletableFuture.completedFuture(null);
    }
}
```

CDI

- @RequestScoped
- @ApplicationScoped
- @SessionScoped
- @Dependent
- @ConversationScoped

```java
@ApplicationScoped
public class BootstrapService {

    @Produces
    public Client getClient() throws ClientNotAvailableException{
        return node.client();
    }
}
```

Eclipse MicroProfile 2.1 (Oct, 2018)

MICROPROFILE
OPTIMIZING ENTERPRISE JAVA

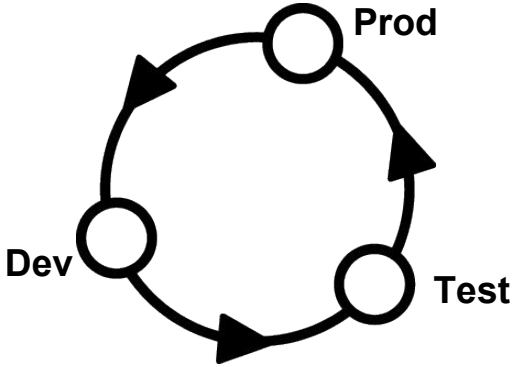
Open Tracing 1.2	Open API 1.0	Rest Client 1.1	Config 1.3
Fault Tolerance 1.1	Metrics 1.1	JWT Propagation 1.1	Health Check 1.0
CDI 2.0	JSON-P 1.1	JAX-RS 2.1	JSON-B 1.0

MicroProfile 2.1

= New
= Updated
= No change from last release (MicroProfile 2.0)

Configuration

Applications need to be **configured** based on a **running environment**. It must be possible to **modify** configuration data from **outside** an application so that the application itself does not need to be repackaged



https://github.com/eclipse/microprofile-config

```java
@ApplicationScoped
public class BootstrapService {

    @Inject @ConfigProperty(name = "java.io.tmpdir", defaultValue = "/tmp")
    private String tempDir;

    @Inject @ConfigProperty(name = "elasticsearch.cluster.name", defaultValue = IndexDetails.CLUSTER_NAME)
    private String clusterName;

    private void startElastic(){
        if(!isRunning){

            String homePath = tempDir + SLASH + appName + SLASH;

            Settings.Builder settingsBuilder = Settings.builder()
                    .put("path.home", homePath)
                    .put("cluster.name", clusterName)
                    .put("node.name", "internal")
                    .put("client.transport.sniff", true)
                    .put("node.max_local_storage_nodes",3);

            ...
        }
    }
}
```

```java
@Inject
@ConfigProperty(name = "requiredProp", defaultValue = "default")
private String required;

@Inject
@ConfigProperty(name = "optionalProp", defaultValue = "default")
private Optional<String> optional;

@Inject
@ConfigProperty(name = "alwaysReloadedProp", defaultValue = "default")
private Provider<String> alwaysReloaded;
```

Configuration

```
@Inject
private Config config;
```

getValue(...)

getPropertyNames()

getConfigSources()

Configuration

By default there are 3 default config sources

Your own source...

Your own source...

System.getProperties()

System.getenv()

META-INF/microprofile-config.properties

```java
public class MemoryConfigSource implements ConfigSource {

        private static final Map<String,String> PROPERTIES = new HashMap<>();

        @Override
        public int getOrdinal() {
            return 900;
        }

        @Override
        public Map<String, String> getProperties() {
            return PROPERTIES;
        }

        @Override
        public String getValue(String key) {
            if(PROPERTIES.containsKey(key)){
                return PROPERTIES.get(key);
            }
            return null;
        }

        @Override
        public String getName() {
            return "MemoryConfigSource";
        }
    }
```

Configuration

- META-INF/services/org.eclipse.microprofile.config.spi.ConfigSource

```
org.microprofileext.config.source.memory.MemoryConfigSource
```

```java
public class DynamicConfigSourceProvider implements ConfigSourceProvider {
    @Override
    public Iterable<ConfigSource> getConfigSources(ClassLoader forClassLoader) {
        List<ConfigSource> configSources = new ArrayList<>();

        Map<String, String> memoryMap = new HashMap<>();
        memoryMap.put("test-prop", "test new memory prop");

        configSources.add(new MemoryConfigSource(memoryMap));

        return configSources;
    }
}
```

Configuration - converters

- Boolean
- Byte
- Short
- Int
- Long
- Float
- Double
- Character
- Class

- Array
- List
- Set

```
@Inject
@ConfigProperty(name="avengers")
private List<String> avengers;

myProp=Iron Man,Captain America,Thor
```

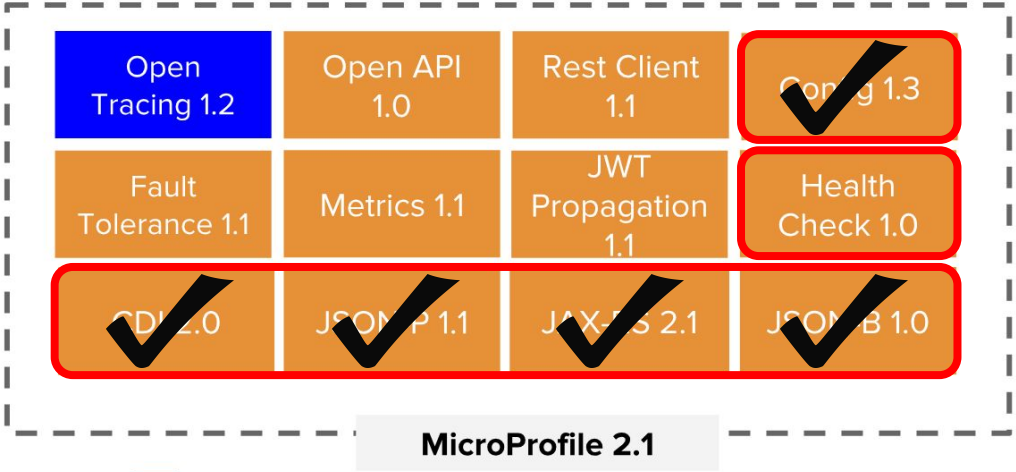
Configuration - converters

```
@Inject
@ConfigProperty(name = "avenger")
private Avenger avenger;

avenger=Iron Man,Tony Stark,true
```

```java
public class AvengerConverter implements Converter<Avenger> {
    @Override
    public Avenger convert(String value) {
        String[] split = value.split(",");
        return Avenger.name(split[0])
                .realName(split[1])
                .alive(Boolean.valueOf(split[2]))
                .build();
    }
}
```

Eclipse MicroProfile 2.1 (Oct, 2018)



= New
= Updated
= No change from last release (MicroProfile 2.0)

Health

Health checks are used to **probe** the **state** of a computing node from another machine (i.e. kubernetes service controller) with the primary target being cloud infrastructure environments where **automated** processes **maintain the state** of computing nodes

Health

- MUST be compatibility with well known **cloud** platforms (i.e. http://kubernetes.io/docs/user-guide/liveness/)
- MUST be appropriate for **machine-to-machine** communication
- SHOULD give enough information for a **human** administrator

```java
@Health
@ApplicationScoped
public class MembershipHealthCheck implements HealthCheck {

    @Override
    public HealthCheckResponse call() {

        HealthCheckResponseBuilder responseBuilder = HealthCheckResponse.named("membership");
        try {
            Connection connection = datasource.getConnection();
            boolean isValid = connection.isValid(timeout);
            DatabaseMetaData metaData = connection.getMetaData();

            responseBuilder = responseBuilder
                    .withData("databaseProductName", metaData.getDatabaseProductName())
                    .withData("driverName", metaData.getDriverName())
                    .withData("isValid", isValid);

            return responseBuilder.state(isValid).build();

        } catch(SQLException  e) {
            responseBuilder = responseBuilder.withData("exceptionMessage", e.getMessage());
            return responseBuilder.down().build();
        }
    }
}
```

Health – output

```
{
  "outcome": "UP",
  "checks": [
    {
      "name": "heap-memory",
      "state": "UP",
      "data": {
        "max %": "0.9",
        "max": "7365197824",
        "used": "185686984"
      }
    },
```

```
    {
      "name": "membership",
      "state": "UP",
      "data": {
        "databaseProductVersion": "1.4.196 (2017-06-10)",
        "databaseProductName": "H2",
        "driverVersion": "1.4.196 (2017-06-10)",
        "isValid": "true",
        "driverName": "H2 JDBC Driver"
      }
    },
    ...
  ]
}
```