MicroProfile

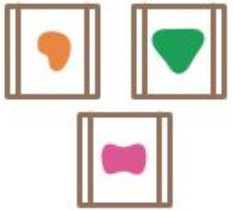
microservices made easy

whoami

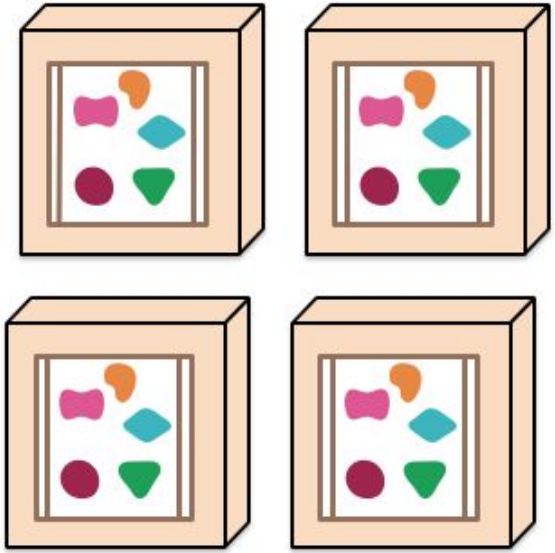
- Martin Štefanko
- Software engineer 3+ years, Red Hat
- MicroProfile contributor
-  @xstefank

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

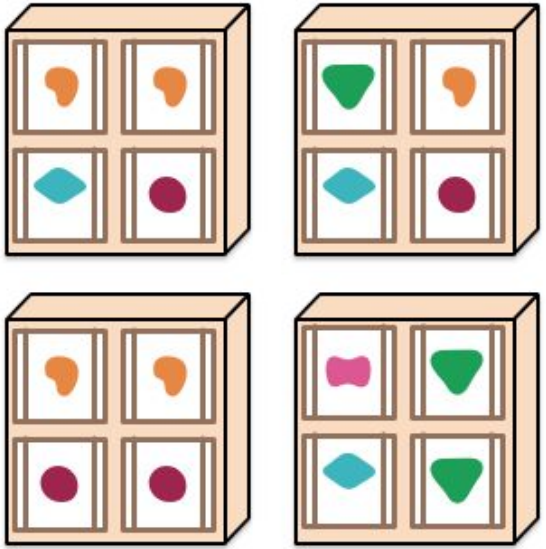
A microservices architecture puts each element of functionality into a separate service...



... 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)

+Under discussion

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

Community - individuals, organizations, vendors

HTTPS://MICROPROFILE.IO/ | HTTPS://PROJECTS.ECLIPSE.ORG/PROJECTS/TECHNOLOGY.MICROPROFILE

Current MicroProfile implementations

HTTPS://MICROPROFILE.IO/ | HTTPS://PROJECTS.ECLIPSE.ORG/PROJECTS/TECHNOLOGY.MICROPROFILE

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

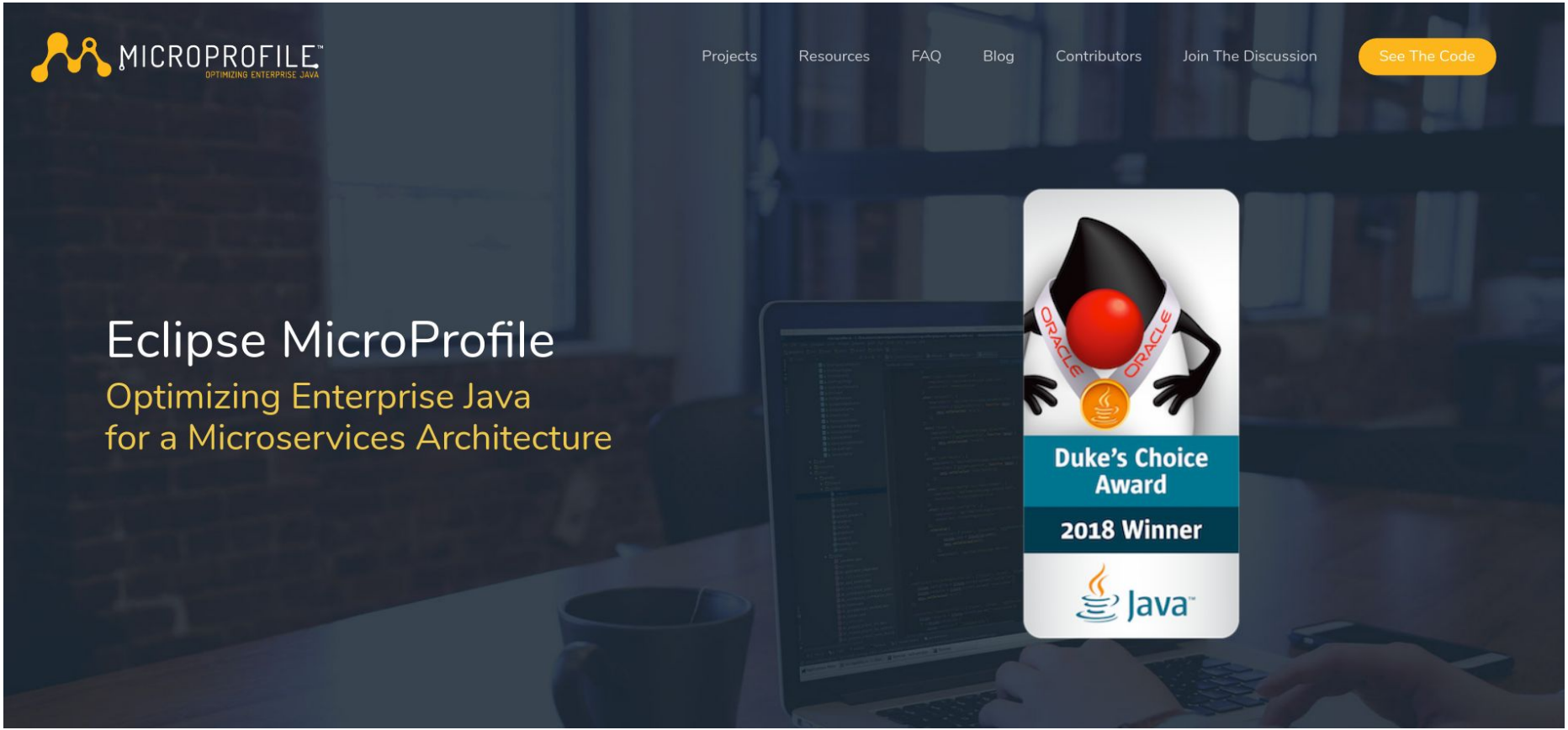
New issue

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.

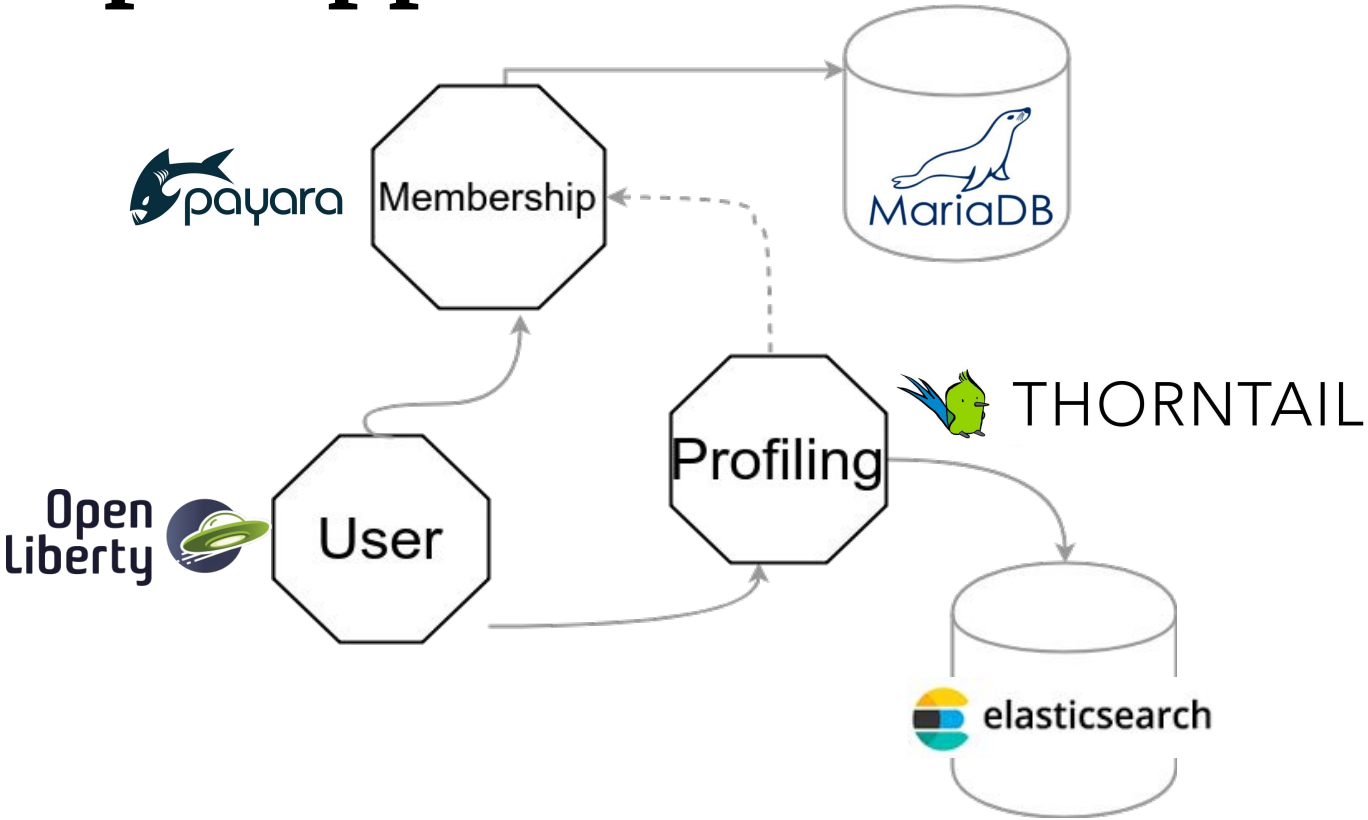
8 Open ✓ 0 Closed

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

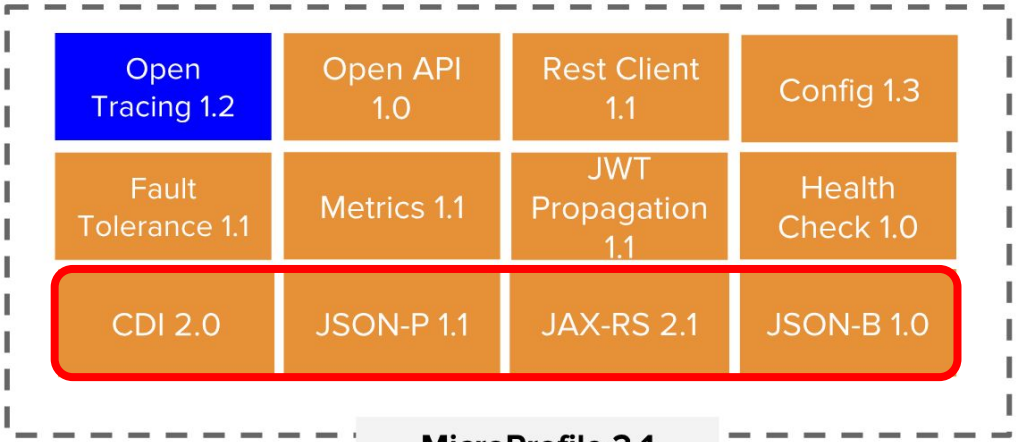


microprofile.io

Example application



Eclipse MicroProfile 2.1 (Oct, 2018)



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

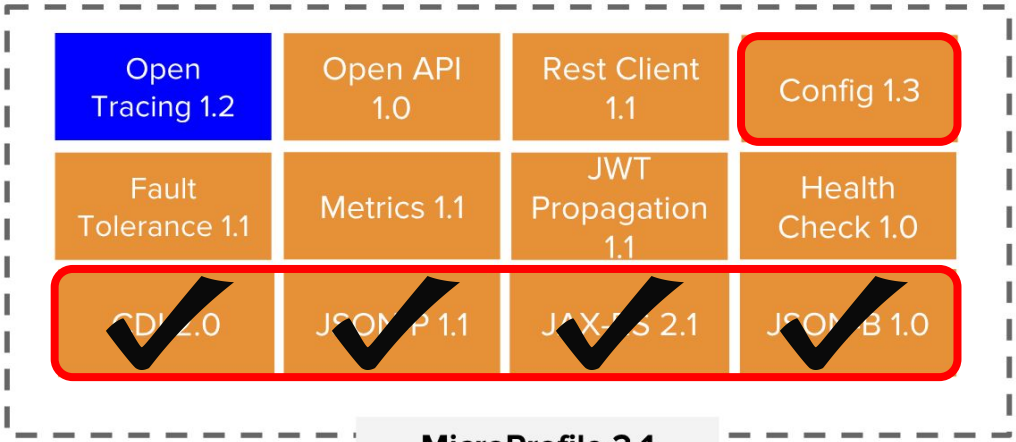
```
@RequestScoped
public class EventLogger {
    @Inject
    private Client client;

    @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

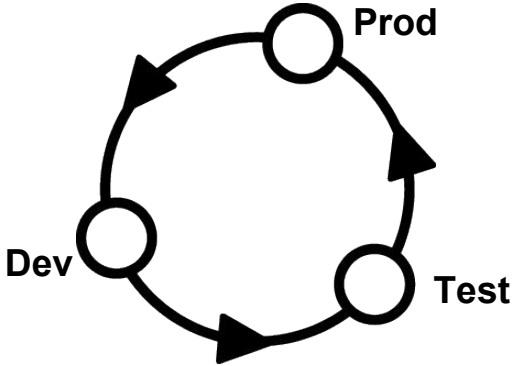
```java
@ApplicationScoped
public class BootstrapService {

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

Eclipse MicroProfile 2.1 (Oct, 2018)



Configuration

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

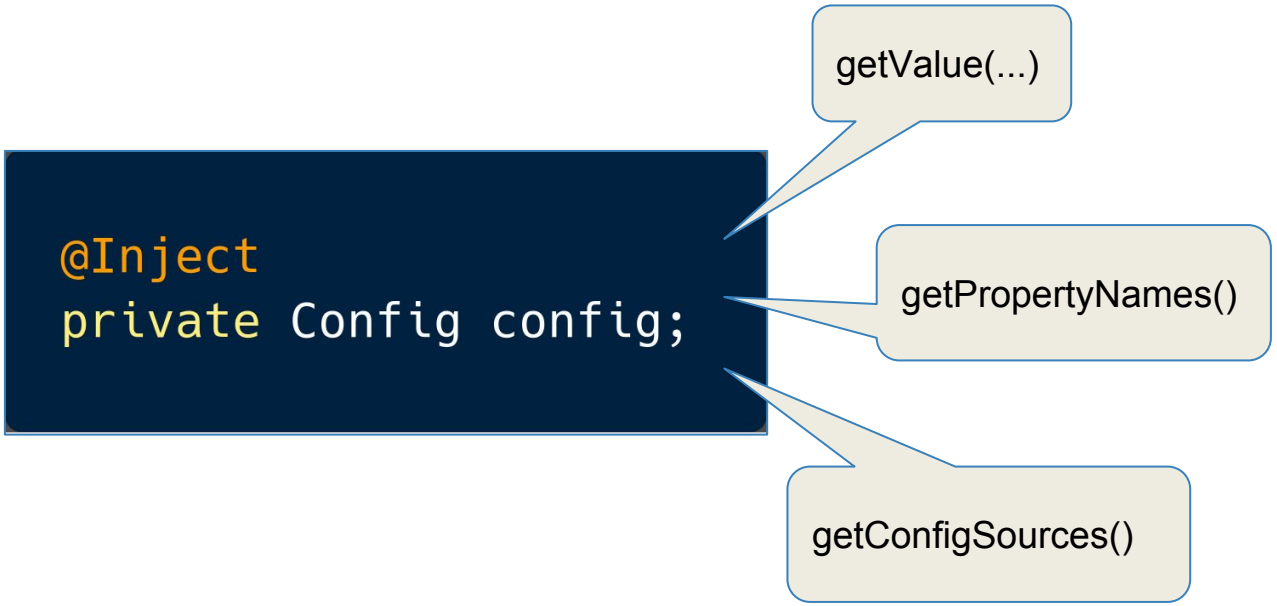


https://github.com/eclipse/microprofile-config

```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);


            ...
        }
    }
}
```

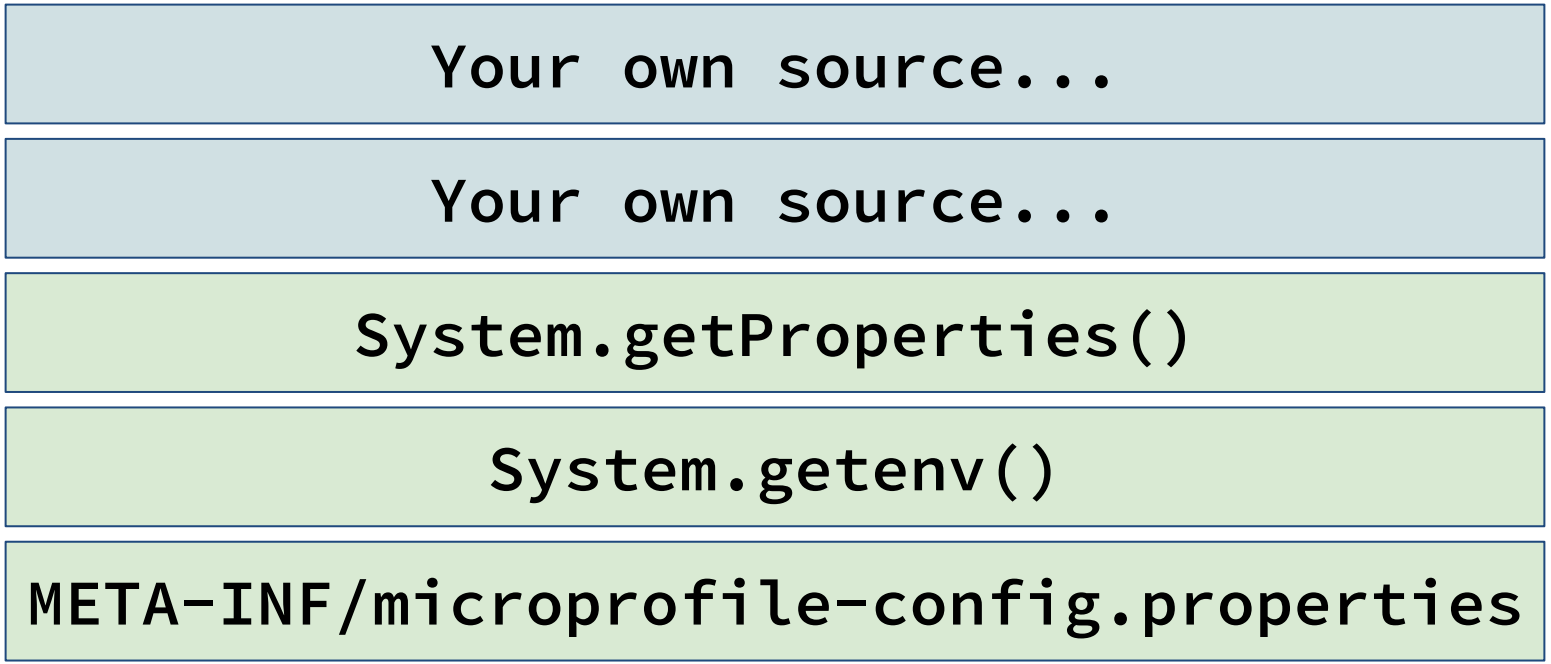
```java
@Inject
@ConfigProperty(name = "requiredProp", defaultValue = "default")
private String required;

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

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

Configuration



Configuration

By default there are 3 default config sources



```
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
```

```
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

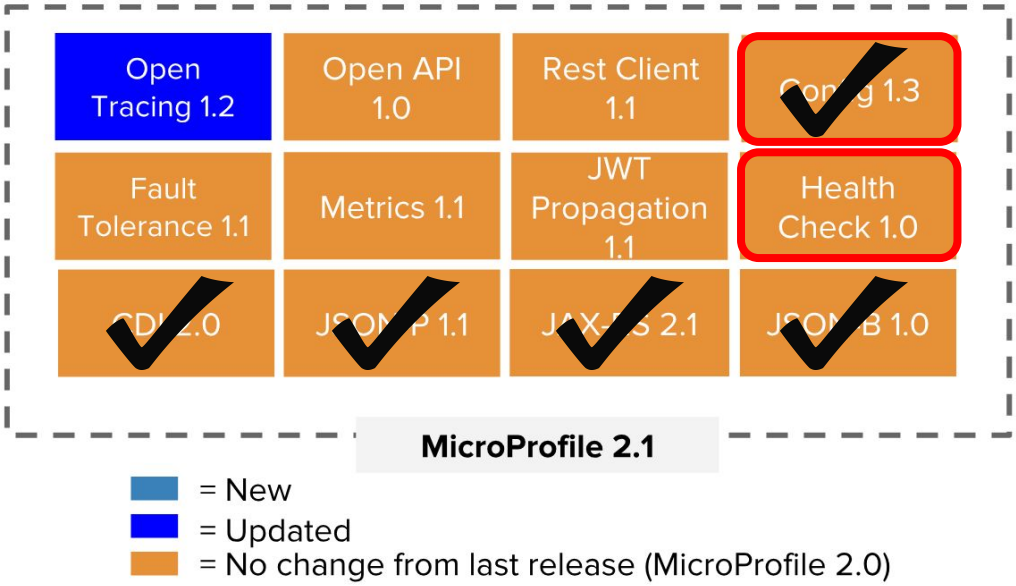
```
@Inject
@ConfigProperty(name="avengers")
private List<String> avengers;

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)



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

/health

```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

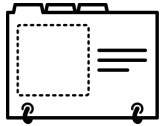
```
{
  "outcome": "UP",
  "checks": [
    {
      "name": "heap-memory",
      "state": "UP",
      "data": {
        "max %": "0.9",
        "max": "7365197824",
        "used": "185686984"
      }
    },
    {
      "name": "membership",
      "state": "UP",
      "data": {
        "databaseProductVersion": "1.4.196 (2017-06-10)",
        "databaseProductName": "H2",
        "driverVersion": "1.4.196 (2017-06-10)",
        "isValid": "true",
        "driverName": "H2 JDBC Driver"
      }
    },
    ...
  ]
}
```

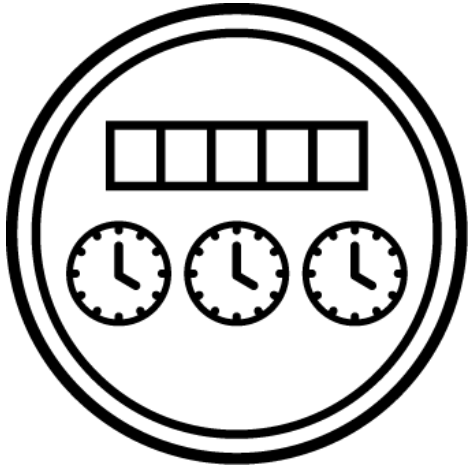
Eclipse MicroProfile 2.1 (Oct, 2018)

Metrics

To ensure reliable operation of software it is necessary to **monitor** essential system parameters. Metrics adds well-known **monitoring endpoints** and **metrics** for each process

Metric Registry





Required Base metrics
Application metrics
Vendor-specific metrics

Metrics - scopes

- **Base**
  - metrics that all MicroProfile vendors have to provide
- **Vendor**
  - vendor specific metrics (optional)
- **Application**
  - application-specific metrics (optional)

Metrics - base scope

General

- UsedHeapMemory
- CommittedHeapMemory
- MaxHeapMemory
- GCCount & GCTime
- JVM Uptime

ClassLoading

- LoadedClassCount
- TotalLoadedClassLoaded
- UnloadedClassCount

Thread

- ThreadCount
- DaemonThreadCount
- PeakThreadCount
- ActiveThreads
- PoolSize

Operating System

- AvailableProcessors
- SystemLoadAverage
- ProcessCpuLoad

Metrics - application

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

    @GET
    @Timed(name = "Memberships requests time",absolute = true,unit = MetricUnits.MICROSECONDS)
    public List<Membership> getAllMemberships() {
        ...
    }

    @GET
    @Path("{id}")
    @Counted(name = "Membership requests",absolute = true,monotonic = true)
    public Membership getMembership(@NotNull @PathParam(value = "id") int id) {
        ...
    }

    ...
}
```

Metrics - application

`@Timed`

`@Gauge`

`@Counted`

`@Metered`

`@Metric`

Metrics - @Metric

```
@Path("ping")
@ApplicationScoped
public class PingResource {

    @Inject
    @Metric(name = "metric counter", absolute = true)
    private Counter metricCounter;

    @GET
    public Response ping() {
        metricCounter.inc();
        return Response.ok(metricCounter.getCount()).build();
    }
}
```

Metrics - REST API

- /metrics

- /metrics/<scope>
  - /metrics/base
  - /metrics/vendor
  - /metrics/application

- /metrics/<scope>/<metric_name>

Metrics - REST API

/metrics/application/ping_request_counter

HTTP GET

```
{
  "ping_request_counter" : 3
}
```

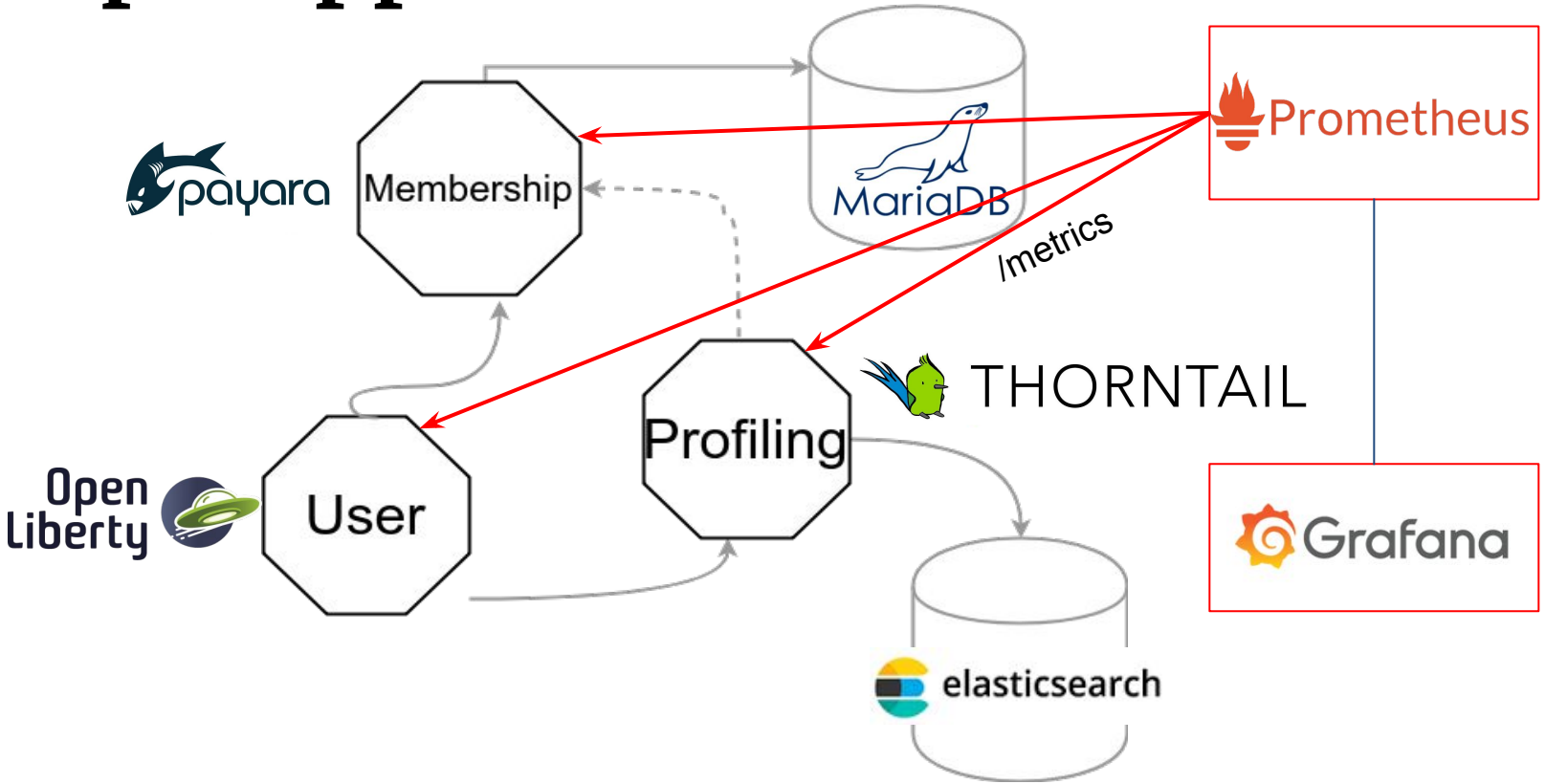
HTTP OPTIONS

```
{
    "ping_request_counter": {
        "unit": "none",
        "type": "counter",
        "description": "Counter for the ping requests",
        "displayName": "App ping counter",
        "tags": "app=ping,deploy=new"
    }
}
```

Metrics - Prometheus

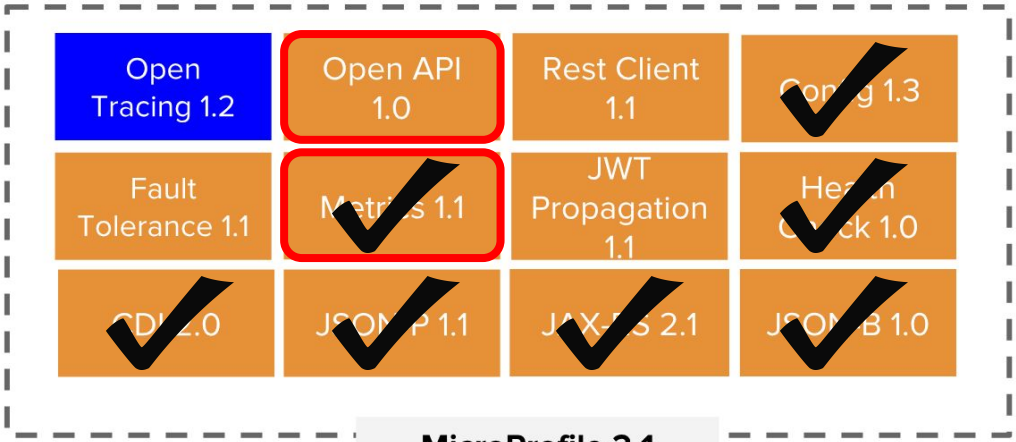
```
# HELP application:ping_request_counter Counter for the ping requests
# TYPE application:ping_request_counter counter
application:ping_request_counter{app="ping",deploy="new"} 3.0
```

Example application



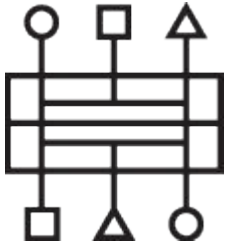

Prometheus Benchmark
Zoom Out
Last 6 hours
Refresh every 1m
Prometheus
localhost
Chunks
600 K
500 K
400 K
300 K
200 K
100 K
0
09:00
10:00
11:00
12:00
13:00
14:00
Chunks
To persist
Max chunks
Max to persist
Chunks Created
6
5
4
3
2
1
0
10m rate
1h rate
6h rate
Samples Ingested
700
600
500
400
300
200
100
0
samples/s
Urgency Score
0.030
0.025
0.020
0.015
0.010
0.005
0
Urgency score
Chunks Persisted
0.8
0.6
0.4
0.2
0
Modes
1.00
0.75
0.50
0.25
0
Rushed Mode Avg: 0
Checkpointing Avg: 0
Series Chunks Persisted
Timeseries Maintenance
0.20
0.15
0.10
Memory Maintenance Duration
7 ms
6 ms
5 ms
4 ms

Eclipse MicroProfile 2.1 (Oct, 2018)



Open API

Management of microservices in an MSA can become unwieldy as the number of microservices increases. Microservices can be managed via their APIs. Management, security, load balancing, and throttling are policies that can be applied to APIs fronting microservices. OpenAPI provides Java interfaces and programming models which allow Java developers to natively produce **OpenAPI v3 documents** from their **JAX-RS** applications.



Open API

- Enterprise Java Binding of the OpenAPI v3 specification
- Based on Swagger Core

- OpenAPI
  - Defines a standard, programming language-agnostic interface description for REST APIs
  - Understandable by humans and machines

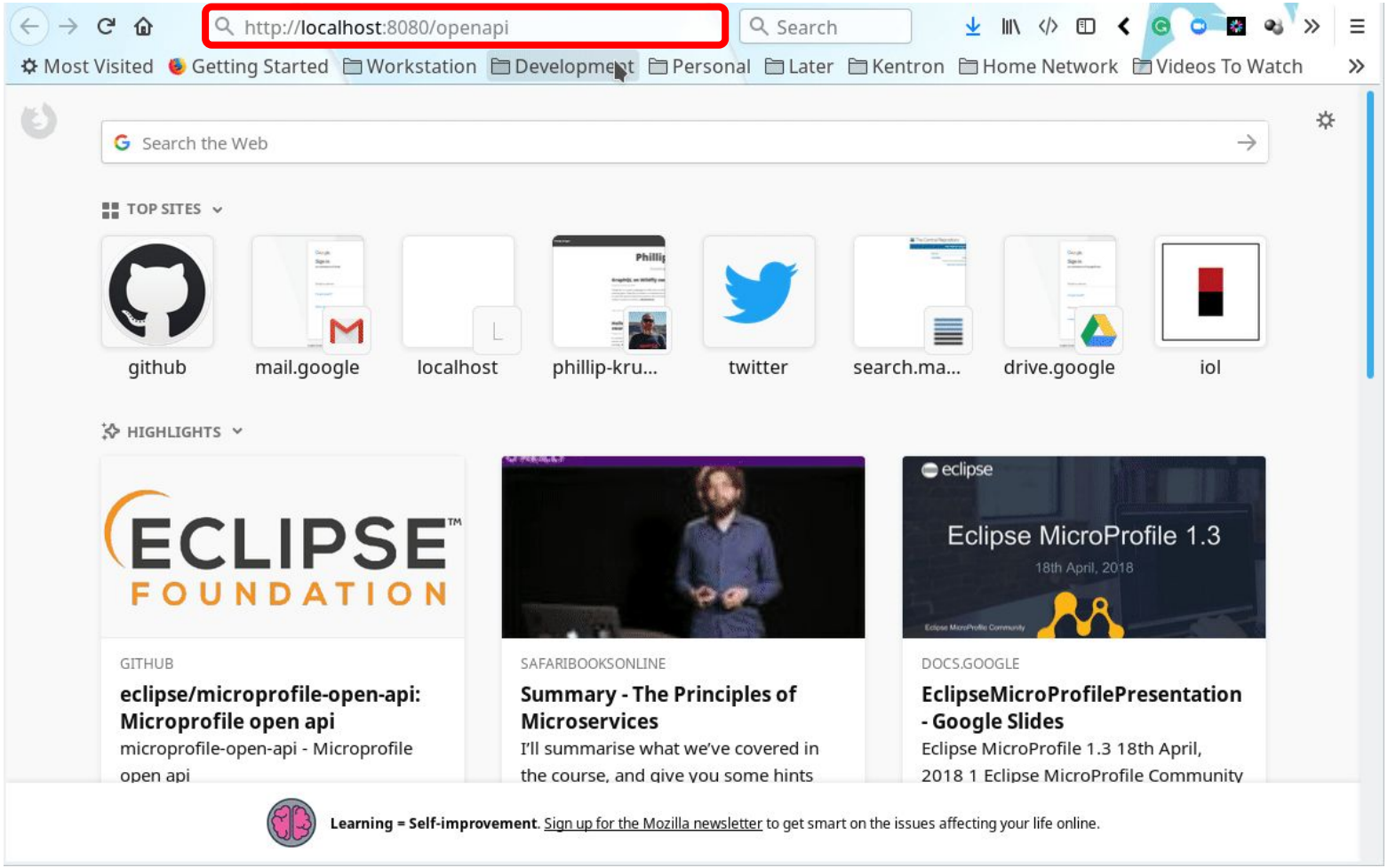
Open API

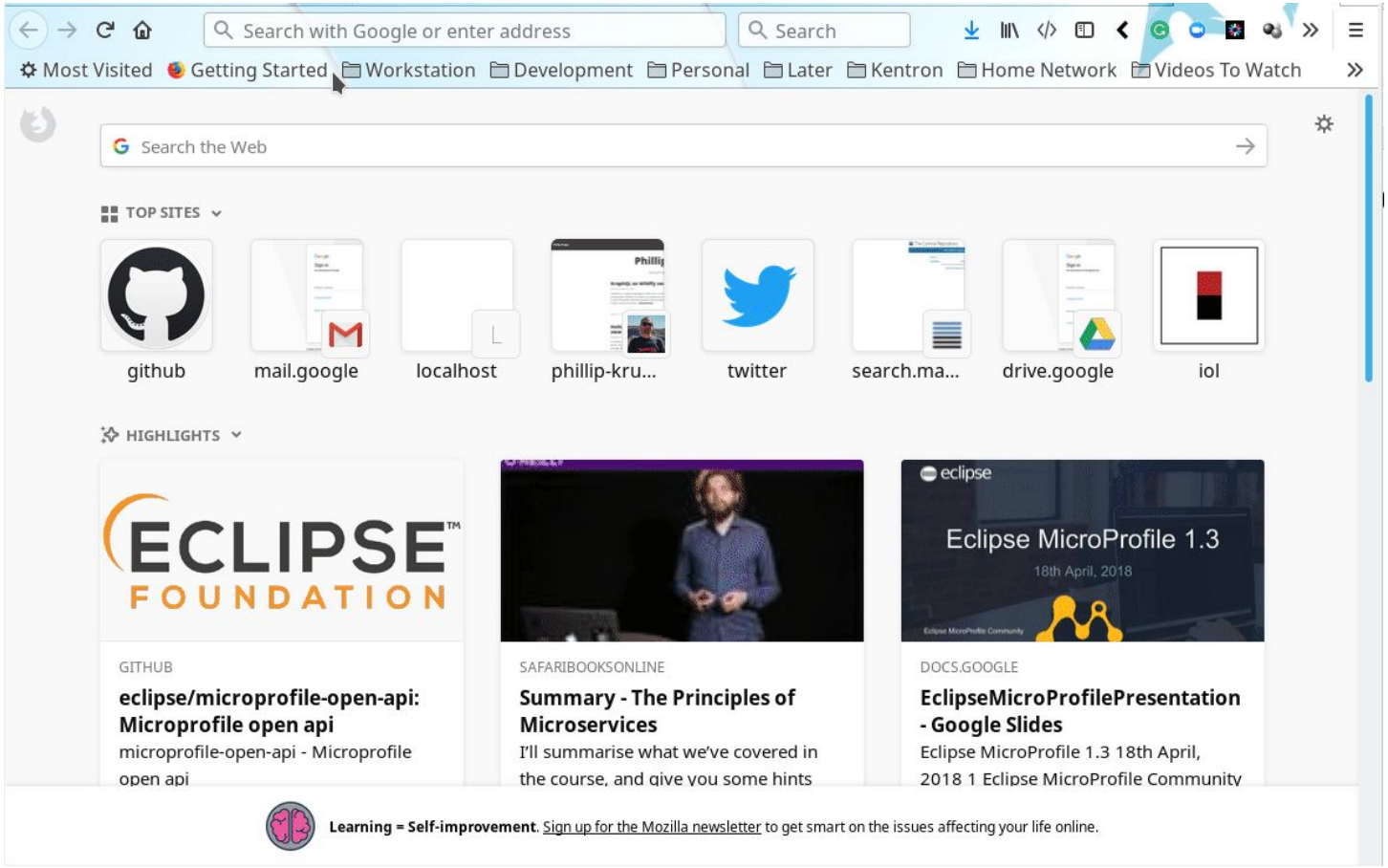
/openapi

```yaml
openapi: 3.0.1
info:
  title: Generated API
  version: "1.0"
paths:
  /ping:
    get:
      responses:
        200:
          description: OK
          content:
            '*/*': {}
```

```java
@ApplicationPath("/api")
@OpenAPIDefinition(info = @Info(
        title = "Membership service",
        version = "1.0.0",
        contact = @Contact(
                name = "Phillip Kruger",
                email = "phillip.kruger@phillip-kruger.com",
                url = "http://www.phillip-kruger.com")
        ),
        servers = {
            @Server(url = "/membership",description = "localhost"),
            @Server(url = "http://yellow:8080/membership",description = "Yellow Pi")
        }
)
public class ApplicationConfig extends Application {
}
```

```java
@RequestScoped
@Path("/")
@Consumes(MediaType.APPLICATION_JSON) @Produces(MediaType.APPLICATION_JSON)
@Tag(name = "Membership service", description = "Managing the membership")
public class MembershipService {

    @GET
    @Path("{id}")
    @Operation(description = "Get a certain Membership by id")
    @APIResponses({
            @APIResponse(responseCode = "200", description = "Successful, returning membership",
                    content = @Content(mediaType = MediaType.APPLICATION_JSON,
                            schema = @Schema(implementation = Membership.class))),
            @APIResponse(responseCode = "504", description = "Service timed out"),
            @APIResponse(responseCode = "401", description = "User not authorized")
    })
    public Membership getMembership(@NotNull @PathParam(value = "id") int id) {
        ...
    }
    ...
}
```


http://localhost:8080/openapi
Search
Most Visited
Getting Started
Workstation
Development
Personal
Later
Kentron
Home Network
Videos To Watch
Search the Web
TOP SITES
github
mail.google
localhost
phillip-kru...
twitter
search.ma...
drive.google
iol
HIGHLIGHTS
ECLIPSE FOUNDATION
GITHUB
eclipse/microprofile-open-api: Microprofile open api
microprofile-open-api - Microprofile open api
SAFARIBOOKSONLINE
Summary - The Principles of Microservices
I'll summarise what we've covered in the course, and give you some hints
eclipse
Eclipse MicroProfile 1.3
18th April, 2018
DOCS.GOOGLE
EclipseMicroProfilePresentation - Google Slides
Eclipse MicroProfile 1.3 18th April, 2018 1 Eclipse MicroProfile Community
Learning = Self-improvement. Sign up for the Mozilla newsletter to get smart on the issues affecting your life online.

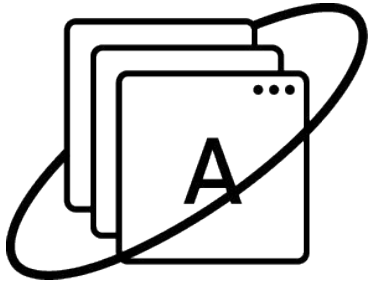


https://github.com/microprofile-extensions/openapi-ext/tree/master/swagger-ui

Eclipse MicroProfile 2.1 (Oct, 2018)

REST Client

In the Microservices world, we typically **talk REST** to other services. While the **JAX-RS specification** defines a **fluent API** for making calls, it is difficult to make it a **true type safe client**. Several JAX-RS implementations support the ability to take an interface definition and create a JAX-RS client from it (JBoss RestEasy, Apache CXF) as well as being supported by a number of service providers (Wildfly Swarm, OpenFeign). MicroProfile Rest Client API provides a **type-safe approach** to invoke RESTful services over HTTP in a **consistent** and **easy-to-reuse** fashion.



REST Client

```
Response response = ClientBuilder.newClient()
            .target("http://localhost:8080/membership/" + membershipId)
            .request()
                .header("Authorization", "Bearer " + token)

Membership membership = membershipProxy.getMembership("Bearer " + token, membershipId);

    throw new RuntimeException("Invalid return value - " + response.getStatus());
}

Membership membership = response.readEntity(Membership.class);
```

REST Client – definition

```
@RequestScoped
@RegisterRestClient
@Consumes(MediaType.APPLICATION_JSON) @Produces(MediaType.APPLICATION_JSON)
@RegisterProvider(RuntimeResponseExceptionMapper.class)
@Path("/api")
public interface MembershipProxy {

    @GET @Path("/{id}")
    public Membership getMembership(@HeaderParam("Authorization") String authorization,
                                    @NotNull @PathParam(value = "id") int id);
}
```

REST Client - injection

```
@Inject
@RestClient
private MembershipProxy membershipProxy;
```

<class_name>/mp-rest/url

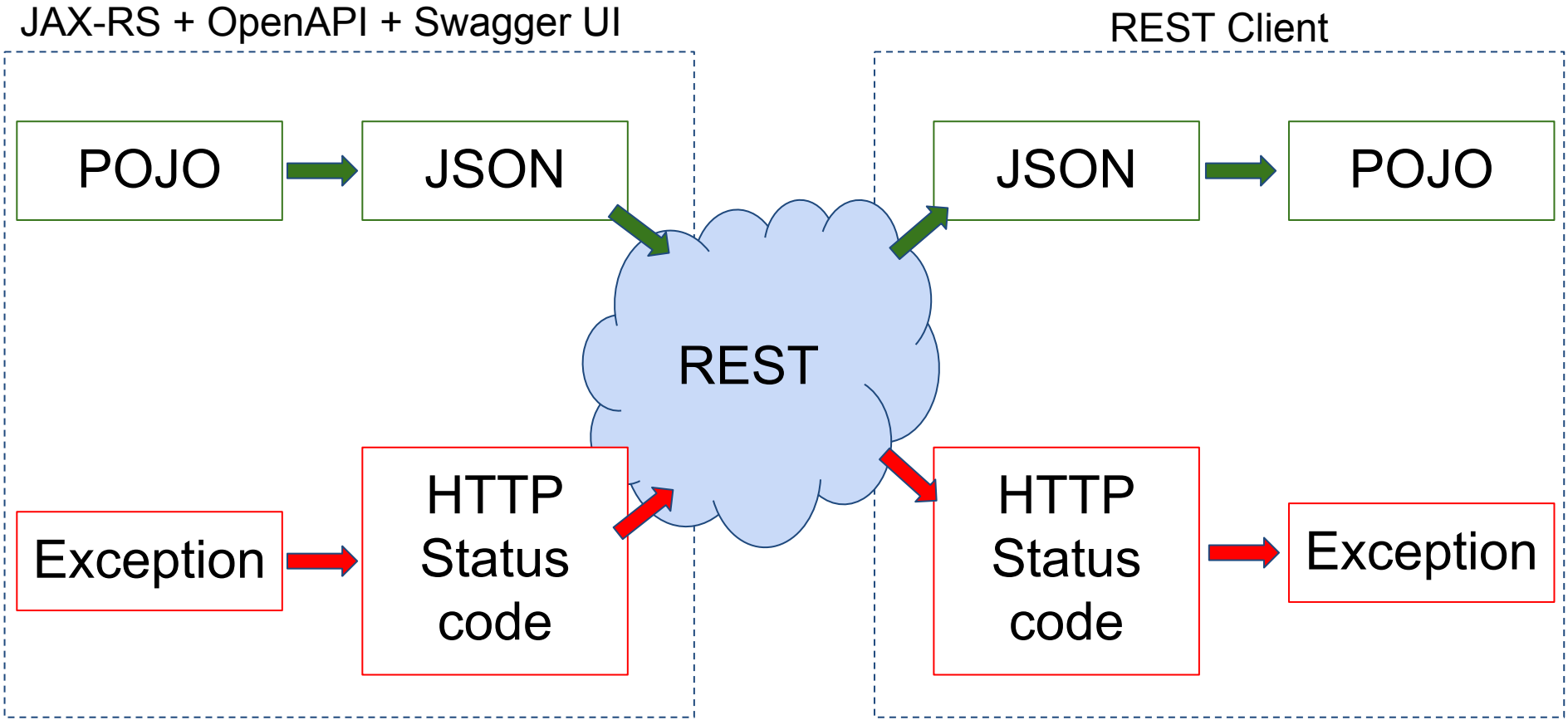
```
com.github.phillipkruger.profiling.membership.MembershipProxy/mp-rest/url=http://localhost:8080/membership
```

REST Client – builder

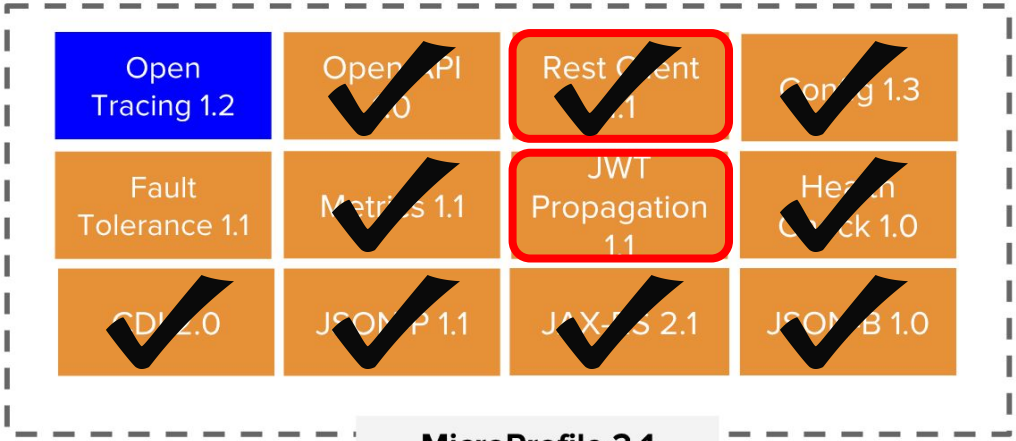
```
membershipProxy = RestClientBuilder.newBuilder()
                  .baseUrl(new URL("http://localhost:8080/membership"))
                  .build(MembershipProxy.class);
```

REST Client – exception mapping

```
@Provider
public class RuntimeResponseExceptionMapper implements ResponseExceptionMapper<RuntimeException>{

    @Override
    public boolean handles(int status, MultivaluedMap<String, Object> headers) {
        return codes.contains(status);
    }

    @Override
    public RuntimeException toThrowable(Response response) {
        ...
        return new RuntimeException(...);
    }
    ...
}
```


JAX-RS + OpenAPI + Swagger UI
REST Client
POJO
JSON
JSON
POJO
REST
Exception
HTTP Status code
HTTP Status code
Exception

Eclipse MicroProfile 2.1 (Oct, 2018)



JWT propagation

The security requirements that involve microservice architectures are strongly related with RESTful Security. In a RESTful architecture style, services are usually **stateless** and any **security state** associated with a client is sent to the target service on **every request** in order to allow services to **re-create** a security context for the caller and perform both **authentication** and **authorization** checks

JWT propagation

Multiple projects/standards directly influenced this proposal and acted as basis for this API, such as:

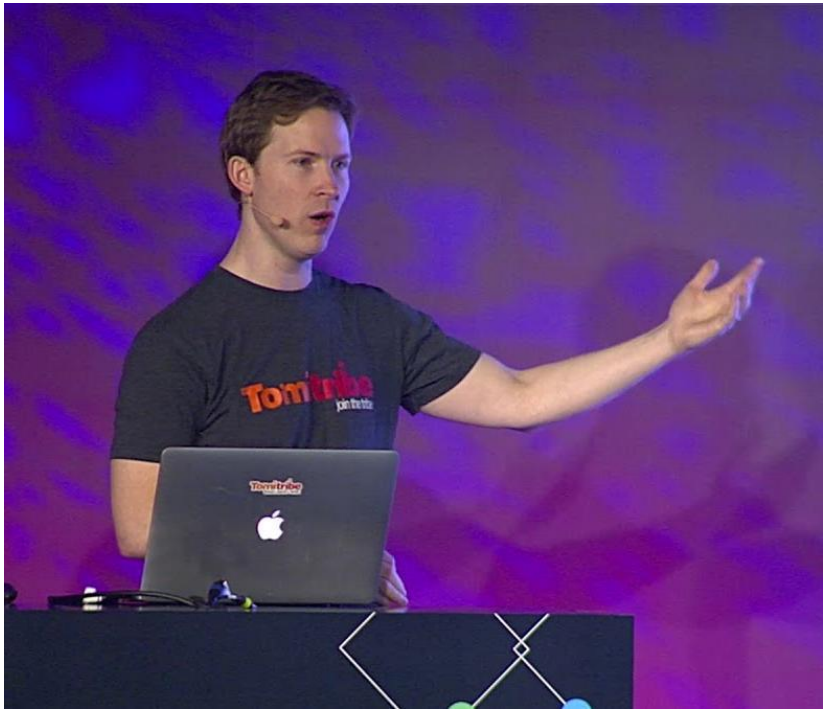
Goal:

- One of the main strategies to propagate the security state from clients to services or even from services to services involves the use of **security tokens**.
- For RESTful based microservices, security tokens offer a very lightweight and interoperable way to propagate identities across different services.

JWT propagation

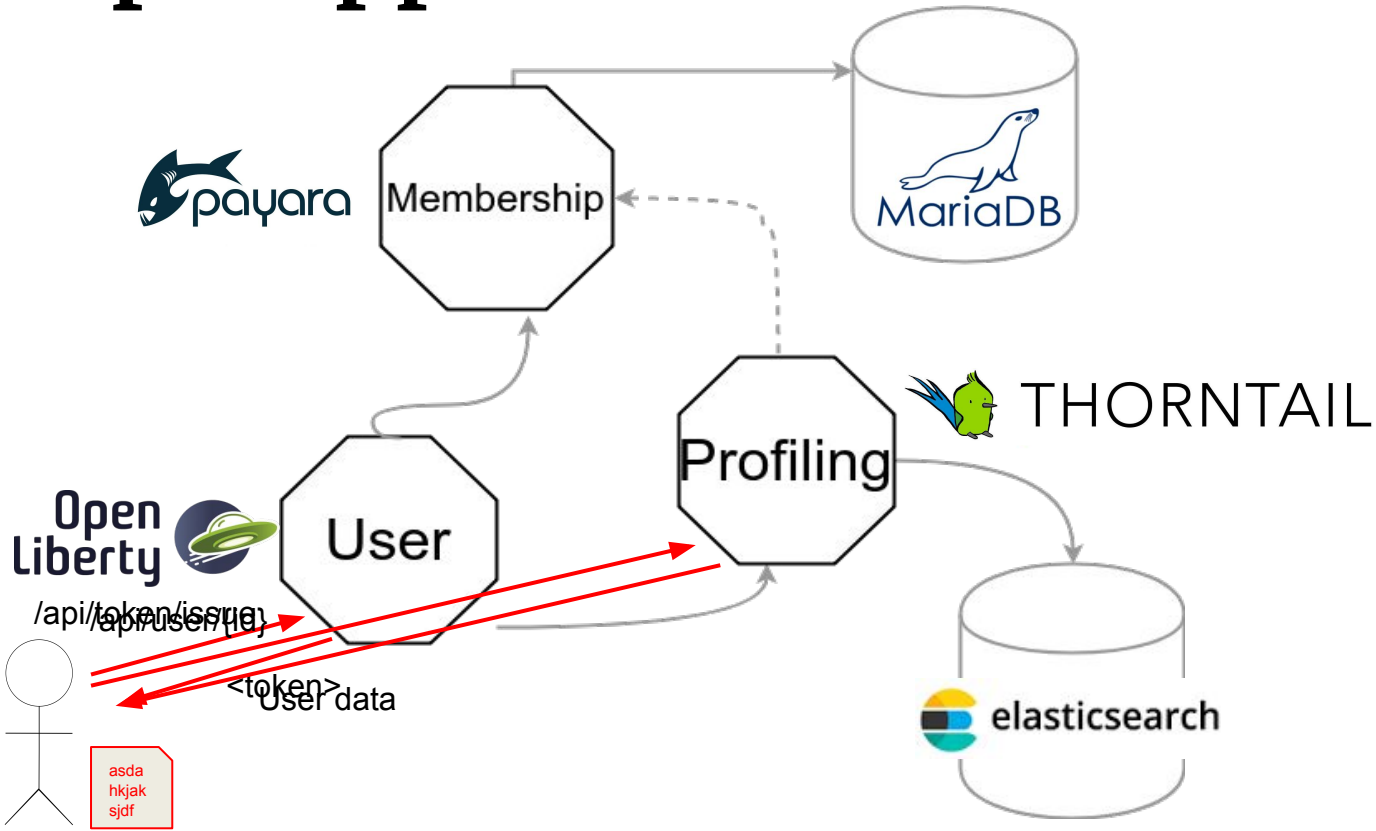
Deconstructing REST Security by David Blevins

https://www.youtube.com/watch?v=9CJ_BAeOmW0



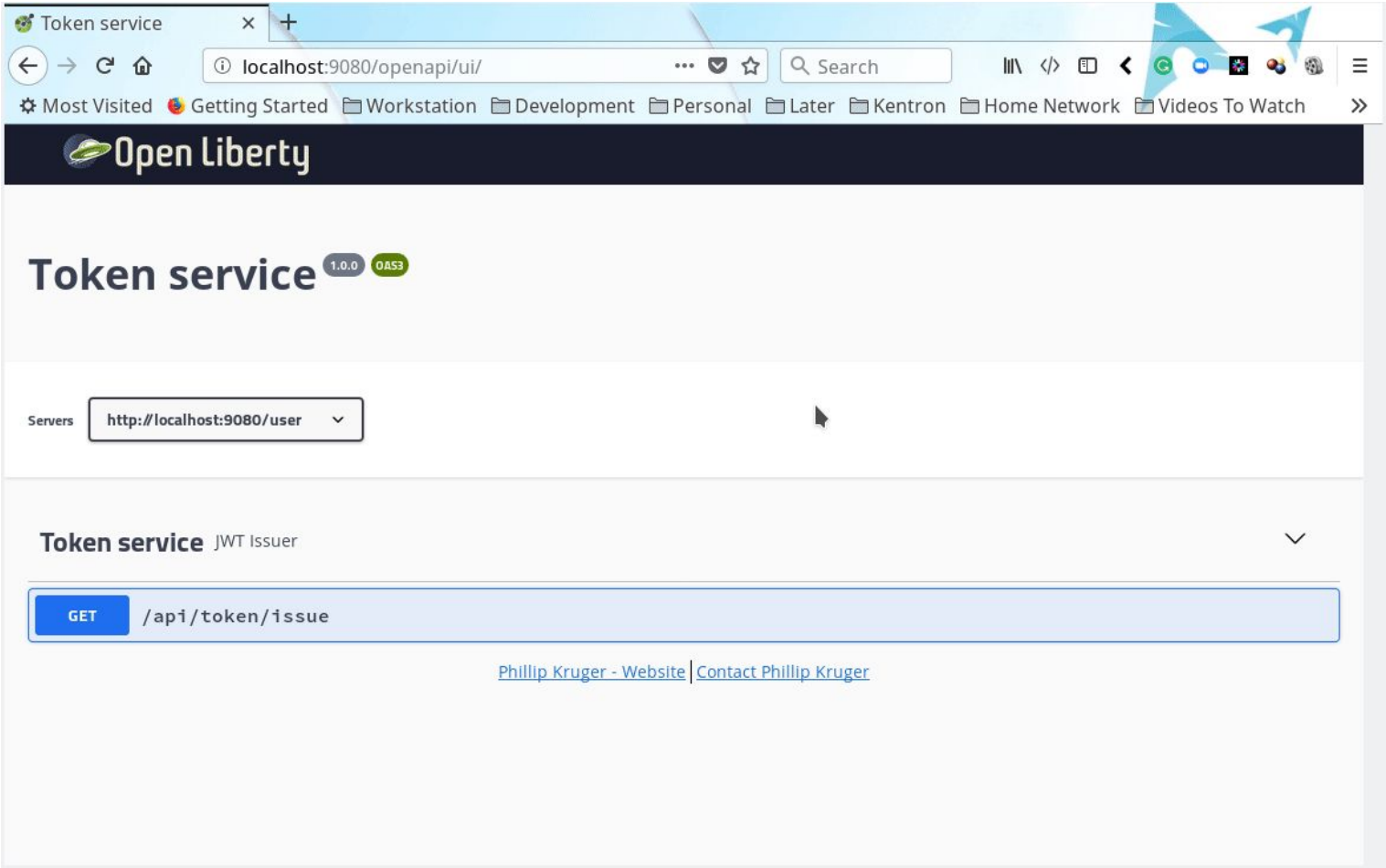
David Blevins - Tomitribe

Example application



```
<!-- To get the token from REST -->
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>default</realm-name>
</login-config>

<!-- Add Security for RESTful Web Services Using Basic Authentication  -->
<security-constraint>
    <display-name>Secure REST Area</display-name>
    <web-resource-collection>
        <web-resource-name>Secure REST</web-resource-name>
        <url-pattern>/api/token/*</url-pattern>
        <http-method>POST</http-method>
        <http-method>GET</http-method>
    </web-resource-collection>

    <auth-constraint>
        <role-name>admin</role-name>
        <role-name>user</role-name>
    </auth-constraint>
</security-constraint>

<security-role>
    <role-name>admin</role-name>
</security-role>
<security-role>
    <role-name>user</role-name>
</security-role>
```

```java
@RequestScoped
@Path("/token")
@Produces(MediaType.TEXT_PLAIN)
@Tag(name = "Token service",description = "JWT Issuer")
@DeclareRoles({"user", "admin"})
@DenyAll
public class TokenService {

    @Context
    private SecurityContext securityContext;

    @Inject
    private TokenIssuer tokenIssuer;

    @Inject
    private TokenSigner tokenSigner;

    @GET
    @Path("/issue")
    @Operation(description = "Issue a JWT Token for the logged in user")
    @APIResponse(responseCode = "200", description = "Get the signed token")
    @RolesAllowed({"admin","user"})
    public Response issueToken(){

        Principal userPrincipal = securityContext.getUserPrincipal();
        String username = userPrincipal.getName();
        List<String> roles = getUserRoles();
        String token = tokenIssuer.issue(username, roles);
        String signToken = tokenSigner.signToken(token);

        return Response.ok(signToken).build();
    }
```


Token service
localhost:9080/openapi/ui/
Search
Most Visited
Getting Started
Workstation
Development
Personal
Later
Kentron
Home Network
Videos To Watch
Open Liberty
Token service
1.0.0
OAS3
Servers
http://localhost:9080/user
Token service
JWT Issuer
GET
/api/token/issue
Phillip Kruger - Website
Contact Phillip Kruger

JWT - token


eyJraWQiOiJcL3ByaXZhdGVLZXkucGVtIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.eyJhdWQiOiJyYXNwYmVycnktcGkiLCJzdWIiOiJwaGlsbGlwLmtydWdlckBwaGlsbGlwLWtydWdlci5jb20iLCJ1cG4iOiJwaGlsbGlwLmtydWdlckBnbWFpbC5jb20iLCJpc3MiOiJodHRwOlwvXC9sZWdvbGFuZC5waGlsbGlwLWtydWdlci5jb20iLCJncm91cHMiOlsidXNlciIsImFkbWluIl0sImV4cCI6MTU0ODI3NDc1OSwiaWF0IjoxNTQ4MjcyOTU5LCJqdGkiOiJmYTQxNjU2OS1jODNlLTRmYTUtOWE5MC1kOGUwMGQ5YzFhNzcifQ.DB6TwtL3s6zwz_HASaO_TrnvvAM8--TqHbP545ibeBATpP4bkxs5WEvwmMnFadyPzbMECKeBHIjYA1Gc5v5kVof1No1KbexB9fbmxaA9tL-ZBl7dfugV9eTwskVnIcG6iqIx5jCaZD2i8q8C7In3Waq2O7TRcOi1XXPGotgP3ss-XYJ_GQSKWbQ8HMwSdQFmBS2t57dkdPl6LP9Zc062IAINUDiHEEyPI6rTWw9IIDXPIdkpjeJvSKJw950Pv5SkaGbb8NsZSiBh0hVAvaMoQjqr-jmpurJ_7Pcp1KhrgAHoC8fxQ0h5gHkhjTjsGhVWQRVtHgJtk34QfBrLK1Y2XQ

JWT - token


```
{
    "aud": "raspberry-pi",
    "sub": "phillip.kruger@phillip-kruger.com",
    "upn": "phillip.kruger@gmail.com",
    "iss": "http://legoland.phillip-kruger.com",
    "groups": [
        "user",
        "admin"
    ],
    "exp": 1548274759,
    "iat": 1548272959,
    "jti": "fa416569-c83e-4fa5-9a90-d8e00d9c1a77"
}
```

```java
@ApplicationPath("/api")
@OpenAPIDefinition(info = @Info(
        title = "Profile service",
        version = "1.0.0",
        contact = @Contact(
                name = "Phillip Kruger",
                email = "phillip.kruger@phillip-kruger.com",
                url = "http://www.phillip-kruger.com")
        ),
        servers = {
            @Server(url = "/profiling",description = "localhost"),
            @Server(url = "http://red:7080/profiling",description = "Red Pi")
        }
)
@SecurityScheme(description = "The JWT from User service",
        securitySchemeName = "Authorization",
        in = SecuritySchemeIn.HEADER,
        type = SecuritySchemeType.HTTP,
        scheme = "bearer",
        bearerFormat = "JWT")
@LoginConfig(authMethod = "MP-JWT",realmName = "jwt-domain")
@DeclareRoles({"user", "admin"})
public class ApplicationConfig extends Application {
}
```

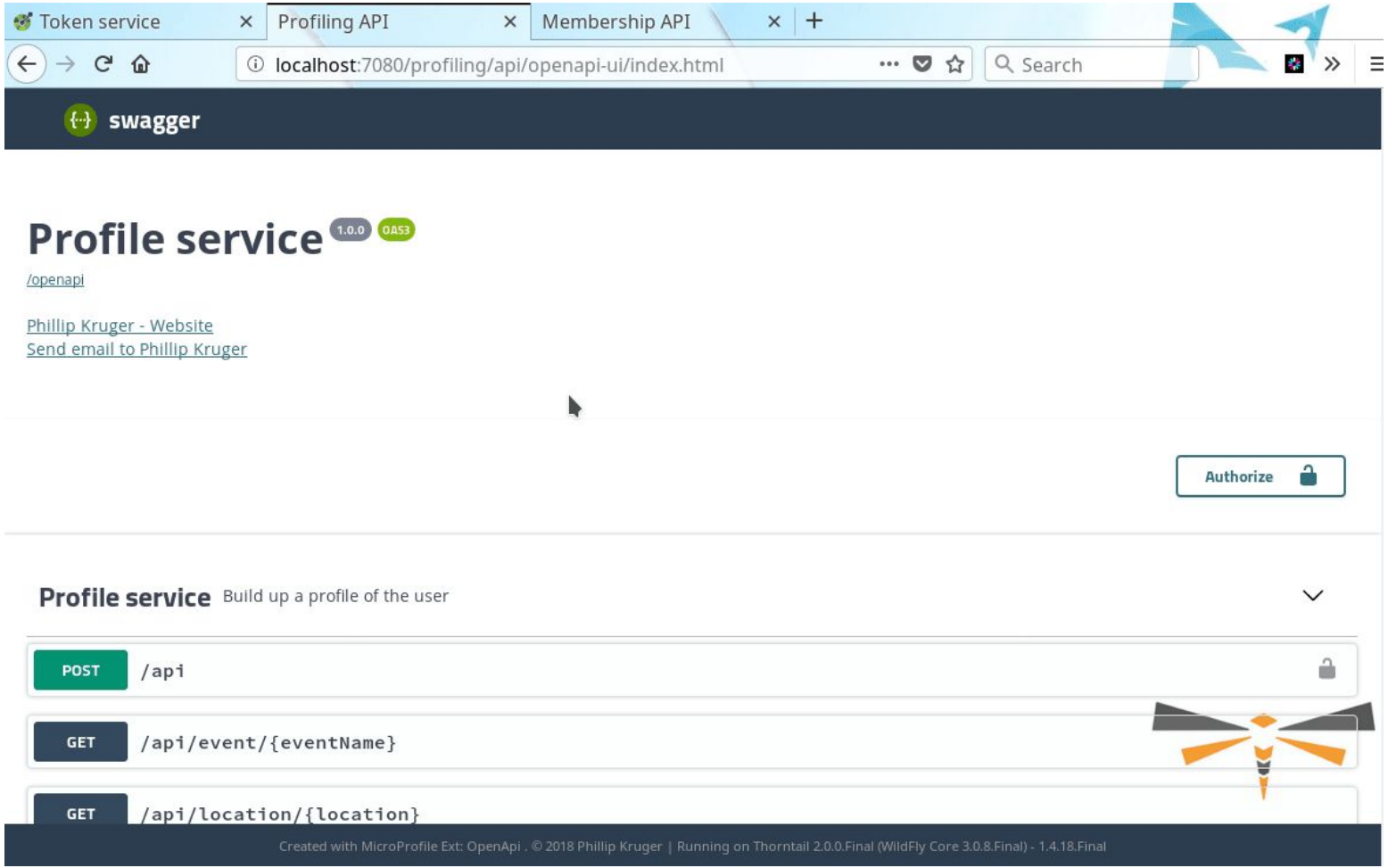
OpenAPI

JWT

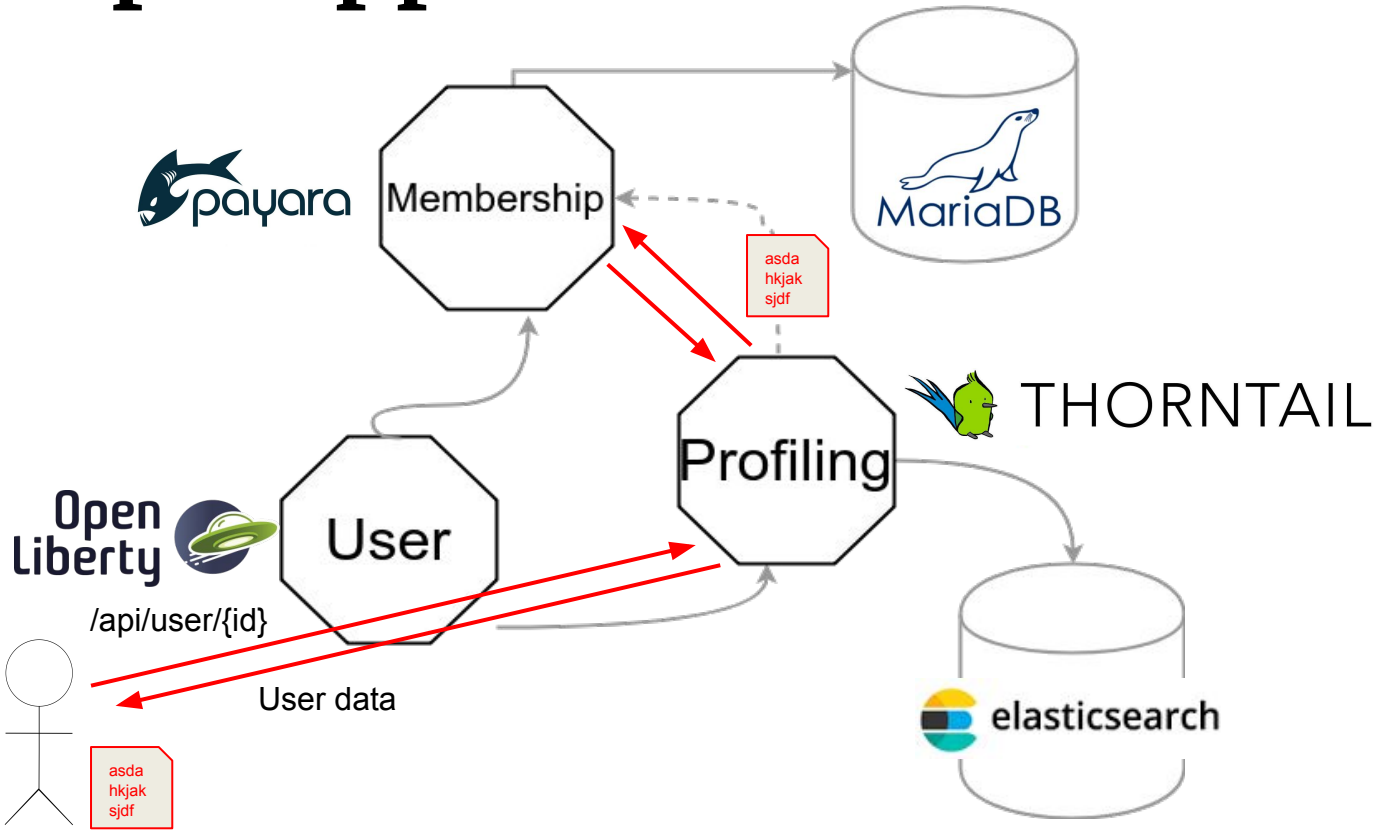
```
@GET @Path("user/{userId}")
@Operation(description = "Getting all the events for a certain user")
@APIResponses({
        @APIResponse(responseCode = "200", description = "Successfull, returning events",
                content = @Content(schema = @Schema(implementation = UserEvent.class))),
        @APIResponse(responseCode = "401", description = "User not authorized"),
        @APIResponse(responseCode = "412", description = "Membership not found, invalid userId",
                headers = @Header(name = REASON)),
        @APIResponse(responseCode = "503", description = "Propl      ection to a downstream service",
                headers = @Header(name = REASON))
})
@SecurityRequirement(name = "Authorization")
@RolesAllowed({"admin","user"})
public Response getUserEvents(
    ...
}
```

OpenAPI

JWT


Token service
Profiling API
Membership API
localhost:7080/profiling/api/openapi-ui/index.html
Search
swagger
Profile service
1.0.0
OAS3
/openapi
Phillip Kruger - Website
Send email to Phillip Kruger
Authorize
Profile service
Build up a profile of the user
POST
/api
GET
/api/event/{eventName}
GET
/api/location/{location}
Created with MicroProfile Ext: OpenApi . © 2018 Phillip Kruger | Running on Thorntail 2.0.0.Final (WildFly Core 3.0.8.Final) - 1.4.18.Final

Example application



Eclipse MicroProfile 2.1 (Oct, 2018)