Micro Profile Hands on Lab

Dmitry Alexandrov, Ivan St. Ivanov

Version 0.9, 15 November 2016

Table of Contents

Introduction

The Java MicroProfile

Magazine Manager

To showcase the MicroProfile we'll develop together a sample app. Let's suppose that it is used by an owner of a magazine to manage the various parts of their business. It consists of four microservices, each of which can be scaled and upgraded separately.

The responsibility of the *Authors* microservice is to keep track of the authors that are writing for the magazine. An author has the following properties:

```
public class Author {
    private Long id;
    private String firstName;
    private String lastName;
    private String email;
    private boolean isRegular;
    private int salary;
}
```

In the initial version the microservice should support the following operations:

- Retrieving all the authors
- Finding author by its names
- Finding author by its ID
- Adding an author

The *Content* microservice takes care of the articles submitted by the authors as well as the comments on those articles. There are two domain objects in it:

```
public class Article {
    private Long id;
    private String title;
    private String content;
    private String author;
    private List<Comment> comments = new ArrayList<>();
}

public class Comment {
    private Long id;
    private String author;
    private String content;
}
```

Its initial functionality covers things like:

- Retrieving all the articles along with their comments
- Finding an article by its ID
- Retrieving all the articles for an author
- Creating an articles
- Adding a comment to an existing article

We'll go into more details in the Persistence chapter why in `Article` we don't refer directly to the `Author` entity from the *Authors* microservice. For now let's assume that we want to explore the Database per service pattern.

The third microservice in our list is the *Advertisers*. As its name implies it contains operations around the magazine's advertisers:

```
public class Advertiser {
    private Long id;
    private String name;
    private String website;
    private String contactEmail;
    private SponsorPackage sponsorPackage;
}

public enum SponsorPackage {

    GOLD(1000), SILVER(500), BRONZE(100);

    private int price;

    SponsorPackage(int price) {
        this.price = price;
    }

    public int getPrice() {
        return price;
    }
}
```

The common functionality that we'll provide with this microservice are these:

- Get all the advertisers
- Find advertiser by name
- Find all the advertisers with a certain sponsor package
- Add an advertiser

Last but not least comes the *Subscriber* microservice. Its domain model is as simple as that:

```
public class Subscriber {
    private Long id;
    private String firstName;
    private String lastName;
    private String email;
    private String address;
    private LocalDate subscribedUntil;
}
```

And the operations that we are going to implement are:

- Get the list of all subscribers
- Find subscriber by ID
- Retrieve all the expiring subscriptions
- Add a subscriber

These four microservices at the end will be configured to run on the four runtimes supporting the MicroProfile (one microservice per server). Basically you will be able to deploy each separate service on every runtime. However, to keep things simple, we did the following breakdown:

- *Authors* running on Apache TomEE
- *Content* running on IBM Liberty Profile
- *Advertisers* running on WildFly Swarm
- *Subscribers* running on Payara Micro

In order to follow better the next steps in this lab, we would suggest that you pick the same setup.

Initial project

We've sketched for you an initial maven project containing the four microservices as Maven subprojects. It is located under the `lab/magman` directory of this lab. Use your favorite IDE to import that project. Make sure that it is imported as Maven project for the IDEs that it is not the default structure.

You'll notice that besides the four microservices there is a simple pom project that groups the three specs:

```
<properties>
    <cdi-version>1.2</cdi-version>
    <jaxrs-version>2.0.1</jaxrs-version>
    <jsonp-version>1.0</jsonp-version>
</properties>

<dependencies>
    <dependency>
        <groupId>javax.enterprise</groupId>
        <artifactId>cdi-api</artifactId>
        <version>${cdi-version}</version>
    </dependency>
    <dependency>
        <groupId>javax.ws.rs</groupId>
        <artifactId>javax.ws.rs-api</artifactId>
        <version>${jaxrs-version}</version>
    </dependency>
    <dependency>
        <groupId>javax.json</groupId>
        <artifactId>javax.json-api</artifactId>
        <version>${jsonp-version}</version>
    </dependency>
</dependencies>
```

By depending on that project instead of `javaee-api`, we will make sure that the microservices we develop will not leak a non-MicroProfile technology like EJB.

The we simply depend on the project by adding the following dependency in the four microservices:

```
<dependency>
    <groupId>bg.jug</groupId>
    <artifactId>microprofile-dependencies</artifactId>
    <version>${project.version}</version>
    <type>pom</type>
    <scope>provided</scope>
</dependency>
```

Thus they can use the three MicroProfile specs. The only thing you need to do is to run in the `lab/magman` directory:

```
mvn clean install
```

After that, you can go to the target directory of each microservice and you'll notice one little war each. This is the standard format to deliver web applications in Java EE. So far.

How this lab works

In the next few chapters we'll guide you through the process of building a CDI, JAX-RS, JSON-P application, packing it in fat jar instead of war and dealing with persistence. In each chapter we'll show you the different implementation aspects of two of the microservices (*Author* and *Content*), while the other two we'll leave to you (with some hints from our side).

You've may also noticed the `sources` directory in the root of this repository. You can always consult it if the hints are not helpful enough and you don't have whom to ask. Besides the solution for the current version of the lab, it contains other features that will probably enter in future extensions that we plan to provide.

Core Micro Profile technologies

Dependency Injection and object lifecycle

Exposure to clients

Communication format

Micro Services Enablement

Make jar, not war

WildFly Swarm

Apache TomEE

IBM Liberty Profile

Payara Micro

Running on non-default HTTP port

WildFly Swarm

Apache TomEE

IBM Liberty Profile

Payara Micro

What about persistence?

Other interesting features