**Hands-on Lab**

# Session Title: Hands-on Cloud-Native Java with Eclipse MicroProfile Workshop

**Abstract:**

Ever wondered what makes a cloud-native application "cloud-native"? Ever wondered what the unique challenges are and how best to address them on a fully-open Java technology stack?

In this workshop, you'll learn what it means to be cloud-native and how that impacts application development. You'll learn Eclipse MicroProfile, an industry collaboration defining technologies for the development and management of cloud-native microservices. With a full set of MicroProfile workshop modules available to you, you'll be able to start with the basics of REST services and progress to more advanced topics, or you can jump right in to fault tolerance, security, metrics and more.

Emily Jiang, Liberty Architect for MicroProfile and CDI, [emijiang@uk.ibm.com](mailto:emijiang@uk.ibm.com), twitter: @emilyfhjiang

Kevin Sutter, STSM, MicroProfile and Java EE architect

[sutter@us.ibm.com](mailto:sutter@us.ibm.com), twitter: @kwsutter

Andrew Rouse, Libery MicroProfile developer

[anrouse@uk.ibm.com](mailto:anrouse@uk.ibm.com)

Table of Contents

[Session Title: Hands-on Cloud-Native Java with Eclipse MicroProfile Workshop 1](#_Toc516564083)

[MicroProfile Introduction 3](#_Toc516564084)

[Eclipse MicroProfile 4](#_Toc516564085)

[MicroProfile 1.3 4](#_Toc516564086)

[Config 1.2 5](#_Toc516564087)

[Health Check 1.0 6](#_Toc516564088)

[Metrics 1.1 6](#_Toc516564089)

[Fault Tolerance 1.0 6](#_Toc516564090)

[JWT Propagation 1.0 6](#_Toc516564091)

[Open API 1.0 6](#_Toc516564092)

[Open Tracing 1.0 7](#_Toc516564093)

[Rest Client 1.0 7](#_Toc516564094)

[Lab Instructions 8](#_Toc516564095)

[openliberty.io Guides 8](#_Toc516564096)

[MicroProfile Guides 9](#_Toc516564097)

[Helpful Hints 12](#_Toc516564098)

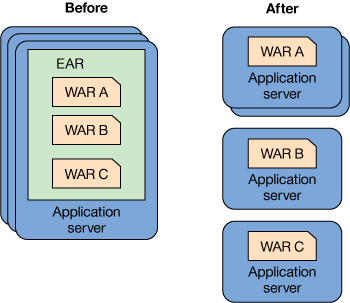
# MicroProfile Introduction

What is a microservice?[[1]](#footnote-1)

* Collection of loosely-coupled services, each representing one unique business function
* This approach is more modular and makes the application easy to understand, easy to develop, and easy to test
* Enables small, autonomous teams to develop, deploy, and scale their respective services independently

Monolith vs Microservice[[2]](#footnote-2)

* Monolith (on the left) – Packaging all aspects of the business functionality into a single application. All services are interdependent and deployed as a single unit.
* Microservice (on the right) – Each business function is separated into its own service. Each service is independent and developed, tested, and deployed independently.



What is [MicroProfile](http://microprofile.io/)?

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

With many partners….

* [Payara](https://www.payara.fish/)
* [Fujitsu](http://www.fujitsu.com/global/)
* [Tomitribe](http://www.tomitribe.com/)
* [IBM](https://www.ibm.com/cloud/websphere-application-platform)
* [Redhat](https://www.redhat.com/en)
* [Kumuluzee](https://ee.kumuluz.com/)
* [LJC](https://www.meetup.com/Londonjavacommunity/)
* [Hammock](https://github.com/hammock-project/hammock)
* [SouJava](https://soujava.org.br/)
* [SmartBear](https://smartbear.com/)
* [Hazelcast](https://hazelcast.com/)
* [Oracle](https://www.oracle.com/index.html)
* [Key Industry Experts](http://microprofile.io/contributors)

### Eclipse MicroProfile

Shortly after the initial MicroProfile 1.0 release, it was decided to move this open-source project to the Eclipse Foundation as the [Eclipse MicroProfile](https://projects.eclipse.org/projects/technology.microprofile) project. Since that time, we have released three updates to MicroProfile in 2017 and are working on more for 2018.

## MicroProfile 1.3

[MicroProfile 1.3](https://github.com/eclipse/microprofile-bom/releases/tag/1.3) is our latest release that contains the following technologies:

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

\* These three Java EE technologies (CDI, JAX-RS, and JSON-P) provide the base for MicroProfile and were the designated content for MicroProfile 1.0.

Each of the other technologies listed are currently beyond the scope of Java EE. They are defining additions or extensions to the programming model for microservice development. Each component is responsible for developing a specification, an API, and a TCK. MicroProfile does not produce an RI (reference implementation), per se. Instead, each member of the MicroProfile community is invited to develop their own implementation and use the TCK to verify its correctness.

[](https://openliberty.io/)

For this lab, we are using [Open Liberty](https://openliberty.io/) – WebSphere’s open-source version of their flagship Liberty application server – as the MicroProfile implementation. All of the build, deployment, and configuration steps in this lab are targeted for use with Open Liberty. But, the actual sample code is not specific to Open Liberty. The code samples could be used with any [MicroProfile-compliant application server.](https://wiki.eclipse.org/MicroProfile/Implementation)

[](http://microprofile.io/)

Now, let’s just briefly introduce each of the other technologies included in Eclipse MicroProfile. Several hotlinks to additional information are also provided.



### Config 1.2

The [Config component](https://github.com/eclipse/microprofile-config) externalizes configuration from the application to improve portability. A core principal of the Config component is to be able to override configuration at deployment time using system properties and environment variables. Config 1.0 was the key component for the first Eclipse MicroProfile 1.1 release, and has had two iterations since.

The Config technology was evolved as JSR 382 [Configuration API 1.0 JSR](https://www.jcp.org/en/jsr/detail?id=382) through the JCP process with Emily Jiang as co-spec lead.

### Health Check 1.0

The [Health Check component](https://github.com/eclipse/microprofile-health) provides a common REST endpoint format to determine microservice availability. This is a simple check to determine if a microservice is alive or not. Contrast with [Metrics](https://github.com/eclipse/microprofile-metrics), which provides much more detail on the running microservice.

### Metrics 1.1

The [Metrics component](https://github.com/eclipse/microprofile-metrics)  provides common REST endpoints for monitoring a microservice’s health. Similar in nature to JMX, but a much simpler API via JAX-RS. Both built-in and application-defined metrics are accessible, with the output in either Json or Prometheus text formats. Again, a popular feature with one update on the books and more to come. Note this API provides more extensive detail than the simple up/down provided by [Health](https://github.com/eclipse/microprofile-health).

### Fault Tolerance 1.0

The [Fault Tolerance component](https://github.com/eclipse/microprofile-fault-tolerance) provides an API and annotations for building robust behavior to cope with unexpected failures in your microservice. Aspects of fault-tolerant execution include timeouts, retries, fallbacks, bulkhead processing, and circuit breakers. This is proving to be one of our most popular features of MicroProfile, and is frequently referenced in conference sessions.

### JWT Propagation 1.0

The [JWT Propagation](https://github.com/eclipse/microprofile-jwt-auth) component provides for interoperable authentication and role-based access control in Java. This MicroProfile feature allows for an authenticated JWT token to be shared across multiple microservices – even if these services are running on other vendors’ implementations. This was one of the first ideas proposed when MicroProfile was first being formed and we delivered it as part of [MicroProfile 1.2](https://github.com/eclipse/microprofile-bom/releases/tag/1.2).

### Open API 1.0

The [Open API component](https://github.com/eclipse/microprofile-open-api) provides a Java API for the [OpenAPI v3 specification](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md) that developers can use to expose their API documentation. Java developers can natively produce OpenAPI v3 documents from their JAX-RS applications. FYI, the OpenAPI specification started off as the Swagger specification.[[3]](#footnote-3)

### Open Tracing 1.0

The [Open Tracing component](https://github.com/eclipse/microprofile-opentracing) allows services to easily participate in a distributed tracing environment. The specification defines behaviors and an API for accessing an [OpenTracing](http://opentracing.io/) compliant Tracer object within your microservice. These trace logs can then be consumed by a third-party distributed tracing facility such as [Zipkin](https://zipkin.io/) or [Jaeger](https://github.com/jaegertracing/jaeger).

### Rest Client 1.0

The [Rest Client component](https://github.com/eclipse/microprofile-rest-client) provides a type-safe approach for invoking RESTful services over HTTP. This API greatly simplifies the client-side API as defined by JAX-RS. The underlying MicroProfile implementation also handles the communication between the client and service.

# Lab Instructions

This lab is a little unique since the actual coding instructions for the lab are contained on the Open Liberty site: <https://openliberty.io/guides/>

Here you will find a collection of Guides tailored for Open Liberty and MicroProfile. (We will focus on the Guides that highlight [MicroProfile](http://microprofile.io/) functionality.) Each of the Guides will take anywhere from 25 – 50 minutes, depending on your experience level. They are all self-sufficient and do not require any particular order. This allows you to tailor the lab to your liking. For example, if you are interested in learning about MicroProfile Config, then you can try out that guide. If you have no interest in learning about MicroProfile Metrics, you can skip it and try something else. The other benefit is that these guides will be available to you after the conference – and there are new ones being updated on a regular basis.

## openliberty.io Guides

Your Firefox browser should be pointing at <https://openliberty.io/guides/>. In the “Search all guides” box, enter “microprofile”.



You will find two types of Guides. Interactive and “static”. The “static” guides are indicated by the lack of an “Interactive” graphic. You will also notice an estimated time allotment. Your mileage may vary… For the lab, I would recommend the use of “static” guides since they will provide a more hands-on experience with coding, building, and deploying the applications into Open Liberty.

Examples of available guides are displayed here:



## MicroProfile Guides

MicroProfile guides in Open Liberty can also be accessed from [**https://github.com/OpenLiberty/tutorial-microprofile**](https://github.com/OpenLiberty/tutorial-microprofile)



You are now encouraged to page through this Guide, following the instructions, to learn how to develop a MicroProfile-based application using the features defined by the selected Guide.

After the initial introductory material, the first step is to clone the repo containing the code for this Guide. Do this step from the Command Prompt (in the “git” directory). There are handy “copy/paste” icons in each of the code snippet windows to make it easy for transferring the code to either the Command Prompt or the Atom editor.

Now, if you go to your Atom window, you will see the Guide’s directory structure and the ability to navigate to the proper location for starting the coding exercises.



When the Guides ask you to create new files, navigate to the proper directory, right mouse click, and select New File:



You will then be prompted to fill in the name of your new file at the directory location of your right mouse click:



And, now you can cut-and-paste from your Guide to the Atom or VS Code editor:



That’s about it. The rest of the lab is working through as many of the Guides as your time allows. If you have any questions, raise your hand and somebody will try to help out.

Good luck and Enjoy!

### Helpful Hints

* Remember each Guide is stand-alone. There is no defined order. You can start (or end) with any of the guides.
* Each Guide has a “start” and “finish” directories. The “start” directories are clean slates ready for your coding efforts. The “finish” directory has the completed application with all code already done. If you have issues during the Guide instructions, you could always reference the “finish” directory for what it’s supposed to look like.
* In Atom, remember to Save (Ctrl-S) when done with each source part. Also, cut (Ctrl-C) and paste (Ctrl-V) are helpful both in Atom and the Firefox.

1. https://en.wikipedia.org/wiki/Microservices [↑](#footnote-ref-1)
2. https://www.ibm.com/developerworks/cloud/library/cl-refactor-microservices-bluemix-trs-1/index.html [↑](#footnote-ref-2)
3. https://swagger.io/blog/difference-between-swagger-and-openapi/ [↑](#footnote-ref-3)