



Open Liberty

# Masterclass: A Technical Deep Dive on Liberty

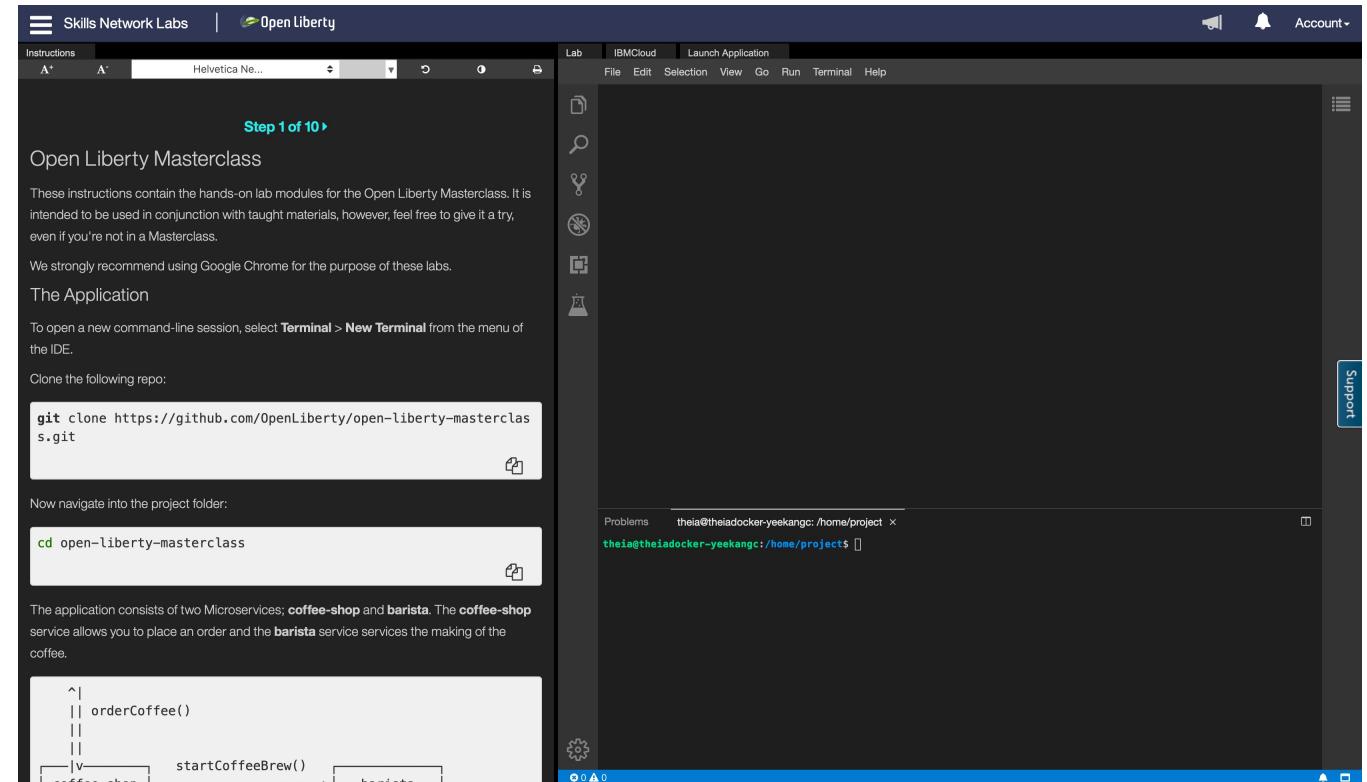
# Ready to get your hands dirty?



Online exercises:

[https://ibm.biz/Liberty  
Masterclass](https://ibm.biz/LibertyMasterclass)

Hosted environment  
with an editor and a  
terminal that you can  
try online through your  
browser



# Ready to get your hands dirty?



The Masterclass is *open sourced* at:

<https://github.com/OpenLiberty/open-liberty-masterclass>

Follow the instructions in the "Before you begin" section of the README.md to:

- Install Pre-requisites
- Prime Maven and Docker Caches

# Mastersclass hands-on content



- [The Application](#)
- [Module 1: Build](#)
- [Module 2: Dev Mode](#)
- [Module 3: Application APIs](#)
- [Module 4: Server Configuration](#)
- [Module 5: Externalizing Configuration](#)
- [Module 6: Integration Testing](#)
- [Module 7: Docker](#)
  - [Overriding Dev Server Configuration](#)
- [Module 8: Testing in Containers](#)
- [Module 9: Support Licensing](#)
- [Conclusion](#)

# Background



# 6 reasons why Liberty



*Lightweight, highly-efficient runtime*

*CI/CD optimized operational experience*

*Simple true-to-production developer experience*

- Just enough runtime → 80% disk and 56% memory saving
- Low operating cost → 4x increased density over Tomcat & Spring Boot
- Continuous delivery → Zero-effort security fixing & zero technical debt
- Zero migration → 100% v2v & fixpack migration saving
- Kubernetes optimized → Self-tuned optimal perf, production-ready, kube-native
- Developer experience → Container & kube-native experience, rapid inner loop

# API Support

- First shipped in WAS 8.5 in 2012
  - Servlet + JSP + JPA
- Web Profile 6 in 2014
- Java EE 7 in 2016 – first commercial product to certify
- Java EE 8 in 2018 – first to certify
- Jakarta EE 8 in 2019 – first to certify
- Eclipse MicroProfile – first to deliver 1.0-1.4, 2.0-2.1, 3.0
- Jakarta EE 9 – first vendor product to certify



[Docs](#)[Get Started](#)[Support](#)[Fork the code](#)[openliberty.io](https://openliberty.io)

Launched September 2017

# Open Liberty

An IBM Open Source Project

**A lightweight open framework for building fast and efficient cloud-native Java microservices.**

Build cloud-native apps and microservices while running only what you need. Open Liberty is the most flexible server runtime available to Java™ developers in this solar system.

[Get Open Liberty](#)

# Liberty Architecture



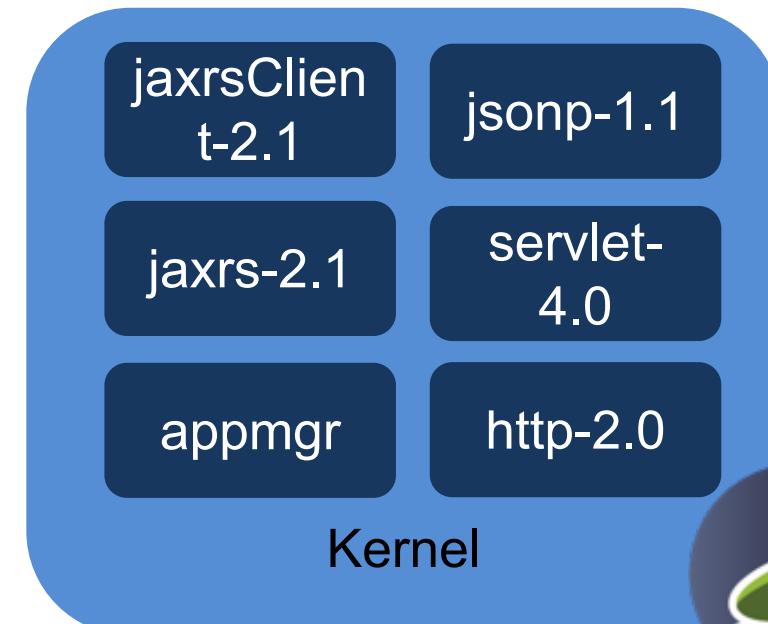
# Just Enough Application Server



- You control which features are loaded into each server instance



```
<feature>jaxrs-2.1</feature>
```



# Liberty Zero Migration



- Zero config migration
  - Write once, run forever
- Zero app migration
  - No behavior changes in existing features
  - New behaviors in new features
- Choose your Java
  - Java 15, 11, 8
  - AdoptOpenJDK
  - IBM
  - OpenJDK
  - Oracle



# Features

  
**Open Liberty**

# Periodic Table of Liberty (21.0.0.3)



**zOS**

**ND**

**Base**

**Core**

**Open Liberty**

**New in 4Q20**

**New in 3Q20**

**New in 2Q20**

**New in 1Q21**

	batchSMFLogging-1.0	zosLocalAdapters-1.0	zosTransaction-1.0	zosSecurity-1.0
collectiveController-1.0	dynamicRouting-1.0	healthManager-1.0	scalingController-1.0	
	clusterMember-1.0	healthAnalyzer-1.0	scalingMember-1.0	
cloudant-1.0	heritageAPIs-1.0	batchManagement-1.0		Security
javaee-7.0	sipServlet-1.1	wsAtomicTransaction-1.2		
javaee-8.0				Operations
jakartaee-8.0				
bells-1.0	microProfile-4.0	adminCenter-1.0	acmeCA-1.0	
concurrent-1.0	mpContextPropagation-1.0	collectiveMember-1.0	constrainedDelegation-1.0	
grpc-1.0	mpGraphQL-1.0	distributedMap-1.0	federatedRepository-1.0	
javaMail-1.6	mpReactiveMessaging-1.0	eventLogging-1.0	jwt-1.0	
jaxb-2.2	mpReactiveStreams-1.0	logstashCollector-1.0	jwtSso-1.0	
jdbc-4.3	opentracing-1.3	monitor-1.0	sessionDatabase-1.0	
jpaContainer-2.2	osgiConsole-1.0	openapi-3.1	webCache-1.0	
jsfContainer-2.3	springBoot-2.0	requestTiming-1.0		
json-1.0	webProfile-7.0	usageMetering-1.0		
jsonbContainer-1.0	webProfile-8.0	restConnector-2.0		
jsonpContainer-1.1		sessionCache-1.0		
	APIs			

# Build

Module 1



# Support for Maven and Gradle



- Manage full server and application lifecycle through the two most popular build technologies.
- Maven
  - liberty-maven-plugin
  - liberty-archetype-\*
  - Docs & Source: <https://github.com/OpenLiberty/ci.maven>
- Gradle
  - liberty-gradle-plugin
  - Docs & Source: <https://github.com/OpenLiberty/ci.gradle>
- All the artefacts you need in Maven Central
  - <https://search.maven.org/search?q=io.openliberty>
  - <https://search.maven.org/search?q=com.ibm.websphere>

# Dev Mode

Module 2



# dev mode

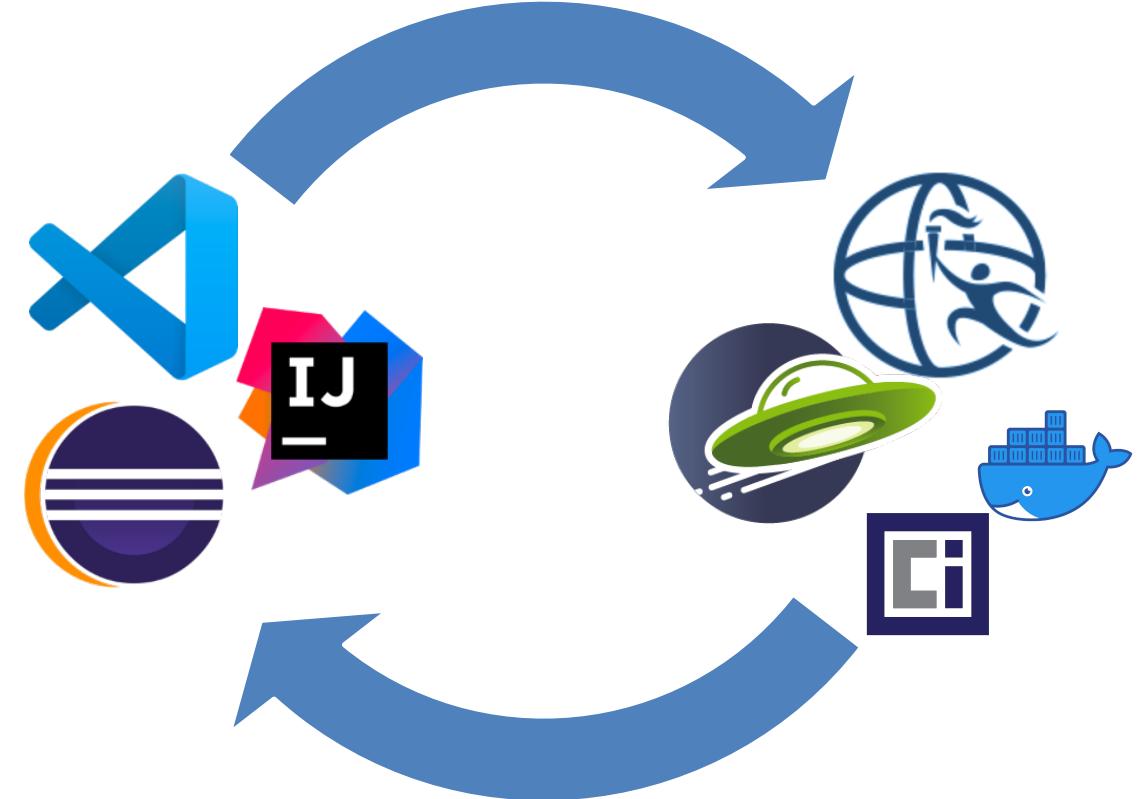
- Boosts developer productivity
- Immediate feedback for code and config changes
- No re-build necessary
- Hot deployment, testing and debugging
- In containers for near-production testing

*mvn liberty:dev*

*gradle libertyDev*

*mvn liberty:devc*

*gradle libertyDevc*



# Tools

- Open Liberty Tools for VS Code, IntelliJ & Eclipse
  - Start/Stop server
  - Run tests
  - View test reports
- MicroProfile Starter
  - Create a new custom Microservice project within the IDE
- MicroProfile REST Client Generator
  - Generate a REST client based on an OpenAPI definition

The screenshot shows the Open Liberty Tools interface integrated into a dark-themed code editor. On the left, a file tree displays a directory structure under 'main' with files like 'SystemApplication.java' and 'SystemResource.java'. In the center, there's an 'OUTLINE' view and a 'LIBERTY DEV DASHBOARD' section which is currently active, showing a dropdown menu with options: Start, Start..., Stop, Run tests, View integration test report, and View unit test report. To the right, a terminal window shows the output of a Maven build for a 'Properties' module, indicating 1 test run, 0 failures, and 0 errors. The terminal also shows results for both integration and unit tests.

```
11 import org.eclipse.microprofile
12 import org.eclipse.microprofile
13
14 @RequestScoped
15 @Path("/properties")
16 public class SystemResource {
17     ...
18     @GET
19     @Produces(MediaType.APPLICATION_JSON)
20     @Timed(name = "getPropertyTime")
21     @Counted(absolute = true,
22             description = "Number of getProperty calls")
23     String getProperty(@PathParam("name") String name) {
24         return properties.get(name);
25     }
26 }
27
28 import it.io.openliberty.sample.Properties
29
30 import org.junit.Test;
31 import org.junit.runner.RunWith;
32 import org.junit.runners.Parameterized;
33 import org.junit.runners.Parameterized.Parameters;
34
35 import static org.junit.Assert.assertEquals;
36
37 import java.util.List;
38 import java.util.Map;
39 import java.util.stream.Collectors;
40
41 import io.restassured.RestAssured;
42 import io.restassured.http.ContentType;
43 import io.restassured.response.Response;
44
45 import static io.restassured.matcher.RestAssuredMatchers.*;
46 import static io.restassured.module.mockmvc.MockMvcRestAssured.*;
47
48 import static org.hamcrest.Matchers.*;
49
50 import static org.junit.Assert.*;
51
52
53 /**
54  * This test class demonstrates how to use the MicroProfile REST Client Generator to generate a Java client for a RESTful service.
55  */
56 @RunWith(Parameterized.class)
57 public class PropertiesTest {
58     private static final String PROPERTIES_URL = "/properties";
59
60     private static final String PROPERTY_NAME = "testProperty";
61
62     private static final String PROPERTY_VALUE = "Hello, Open Liberty!";
63
64     private static final String EXPECTED_RESPONSE = "{'name': 'testProperty', 'value': 'Hello, Open Liberty!'";
65
66     private static final Map<String, String> EXPECTED_HEADERS = Map.of(
67         "Content-Type", "application/json",
68         "Content-Length", "19"
69     );
70
71     private static final Map<String, String> EXPECTED_STATUS_CODE = Map.of(
72         "Status", "200 OK"
73     );
74
75     @Parameters
76     public static Object[] data() {
77         return new Object[][] {
78             {PROPERTY_NAME, PROPERTY_VALUE}
79         };
80     }
81
82     @Test
83     public void testGetProperty() {
84         Response response = given()
85             .when()
86             .get(PROPERTIES_URL + "?name=" + PROPERTY_NAME)
87             .then()
88             .body(equalTo(EXPECTED_RESPONSE))
89             .header("Content-Type", equalTo("application/json"))
90             .header("Content-Length", equalTo("19"))
91             .header("Status", equalTo("200 OK"))
92             .extract().response();
93
94         assertEquals(EXPECTED_RESPONSE, response.getBody().asString());
95         assertEquals(EXPECTED_HEADERS, response.getHeaders());
96         assertEquals(EXPECTED_STATUS_CODE, response.getStatusCode());
97     }
98 }
```



# Developer experience



## IDEs

The IDEs section contains three items: the Microsoft Visual Studio Code logo (blue 'X' icon), the JetBrains IntelliJ IDEA logo (purple and blue 'IJ' icon), and the 'Dev Mode' logo (a purple and orange circle with horizontal lines).

## Repositories

The Repositories section contains two items: 'The Central Repository' (represented by a blue vertical bar icon) and 'docker hub' (represented by the Docker logo, which is a blue whale icon next to the text 'docker hub').

## Build

The Build section contains two items: 'Maven™' (represented by the Maven logo, which is the word 'Maven' in a bold black font with a colorful feather icon above it) and 'Gradle' (represented by the Gradle logo, which is a dark blue elephant icon next to the word 'Gradle').

## APIs

The APIs section contains four items: 'MICROPROFILE™ OPTIMIZING ENTERPRISE JAVA' (represented by a yellow and orange 'M' icon), 'Java EE™' (represented by the Java EE logo, which is a blue coffee cup icon next to the text 'Java EE'), 'JAKARTA EE' (represented by an orange sailboat icon next to the text 'JAKARTA EE'), and 'Spring Boot®' (represented by a green hexagon icon with a white power symbol inside).

## Testing

The Testing section contains three items: 'microshed testing' (represented by a blue house icon next to the text 'microshed testing'), 'JUnit' (represented by the JUnit logo, which is the word 'JUnit' in red and green), and 'Arquillian' (represented by the Arquillian logo, which is a green alien head icon next to the word 'Arquillian').

Jesse Gallagher  
@Gidgerby

Have I mentioned lately how much of a delight @OpenLibertyIO is to work with? It's just thoroughly pleasant.

Tim Zöller  
@javahippie

The @OpenLibertyIO dev mode is one of the best hot-reload features I have ever worked with, I am seriously impressed!

# Application APIs

Module 2



# Java EE 7



appClientSupport-1.0	ejbPersistentTimer-3.2	jaspic-1.1	managedBeans-1.0
batch-1.0	ejbRemote-3.2	jaxb-2.2	mdb-3.2
concurrent-1.0	j2eeManagement-1.1	jaxws-2.2	wasJmsClient-2.0
ejb-3.2	javaMail-1.5	jca-1.7	webProfile-7.0
ejbHome-3.2	jacc-1.5	jms-2.0	wmqJmsClient-2.0

appSecurity-2.0	jaxrsClient-2.0	jsp-2.3
beanValidation-1.1	jdbc-4.2	managedBeans-1.0
cdi-1.2	jndi-1.0	servlet-3.1
ejbLite-3.2	jpa-2.1	ssl-1.0
el-3.0	jsonp-1.0	websocket-1.1
jaxrs-2.0	jsf-2.2	

# Jakarta EE or Java EE 8



appClientSupport-1.0	ejbHome-3.2	jacc-1.5	managedBeans-1.0
appSecurityClient-1.0	ejbPersistentTimer-3.2	jaxb-2.2	mdb-3.2
batch-1.0	ejbRemote-3.2	jaxws-2.2	wasJmsClient-2.0
concurrent-1.0	j2eeManagement-1.1	jca-1.7	webProfile-8.0
ejb-3.2	javaMail-1.6	jms-2.0	wmqJmsClient-2.0

appSecurity-3.0	jaxrs-2.1	jsonp-1.1	websocket-1.1
beanValidation-2.0	jaxrsClient-2.1	jsf-2.3	
cdi-2.0	jdbc-4.2	jsp-2.3	
ejbLite-3.2	jndi-1.0	managedBeans-1.0	
el-3.0	jpa-2.2	servlet-4.0	
jaspic-1.1	jsonb-1.0	ssl-1.0	



# Open Liberty beta is Jakarta EE 9 compatible

Open Liberty beta releases have satisfied the Jakarta EE 9 TCK and have been added to the Jakarta EE website



Jared Anderson on Mar 5, 2021

announcements, Java EE, beta, Jakarta EE, community, ...



<https://openliberty.io/blog/2021/03/05/jakarta-ee-9-compatibility.html>

<https://jakarta.ee/compatibility/#tab-9>

**javax.\* -> jakarta.\***

Jakarta EE Compatible Products    Jakarta EE 9    Jakarta EE 8

Jakarta EE 9 Platform Compatible Products

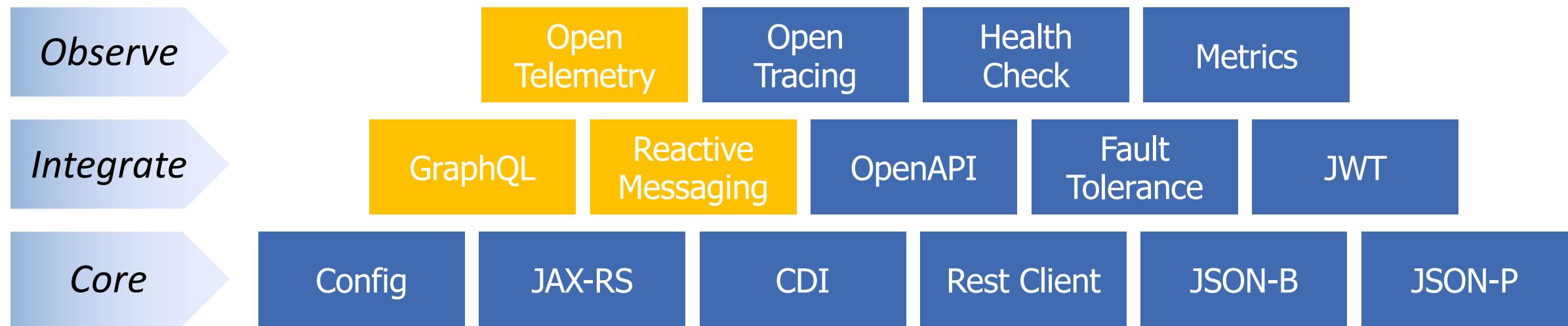
 Eclipse GlassFish Eclipse Foundation 6	 Open Liberty IBM Corporation <u>21.0.0.3-beta</u>
---	--

Jakarta EE 9 Web Profile Compatible Products

 Eclipse GlassFish Eclipse Foundation 6	 Open Liberty IBM Corporation <u>21.0.0.2-beta</u>	 WildFly Red Hat 23.0.0
---	--	---



<https://microprofile.io/>



Open cloud-native Java APIs

# MicroProfile

- Builds on Java EE
- By the Java EE community
- Open Source at Eclipse
- Multiple Implementations

```
@Path("/")
public class RestEE {

    @GET
    @Counter
    @Traced
    public String hello() {
        return "Hello MicroProfile!!";
    }
}
```

# Server Configuration

Module 4



# Simple Config



```
<server>
  <featureManager>
    <feature>jaxrs-2.1</feature>
  </featureManager>

  <webApplication location="myweb.war" contextRoot="/" />

  <applicationManager autoExpand="true"/>
</server>
```

server.xml

-Xmx1g  
-Dsystem.prop=value

jvm.options

WLP\_OUTPUT\_DIR=/usr/wlp-out/

server.env

# Composing Config



```
<server>
  <httpEndpoint id="defaultHttpEndpoint" host="${host}"
                httpPort="${http}"
                httpsPort="${https}"/>
</server>
```

configDropins/defaults/common-http.xml

```
<server>
  <include location="https://myHost/ports.xml" />
  <variable name="host" value="${my.host}" />
  <variable name="http" value="${my.host.http}" />
  <variable name="https" value="${my.host.https}" />
</server>
```

configDropins/overrides/ports.xml

# Externalizing Configuration

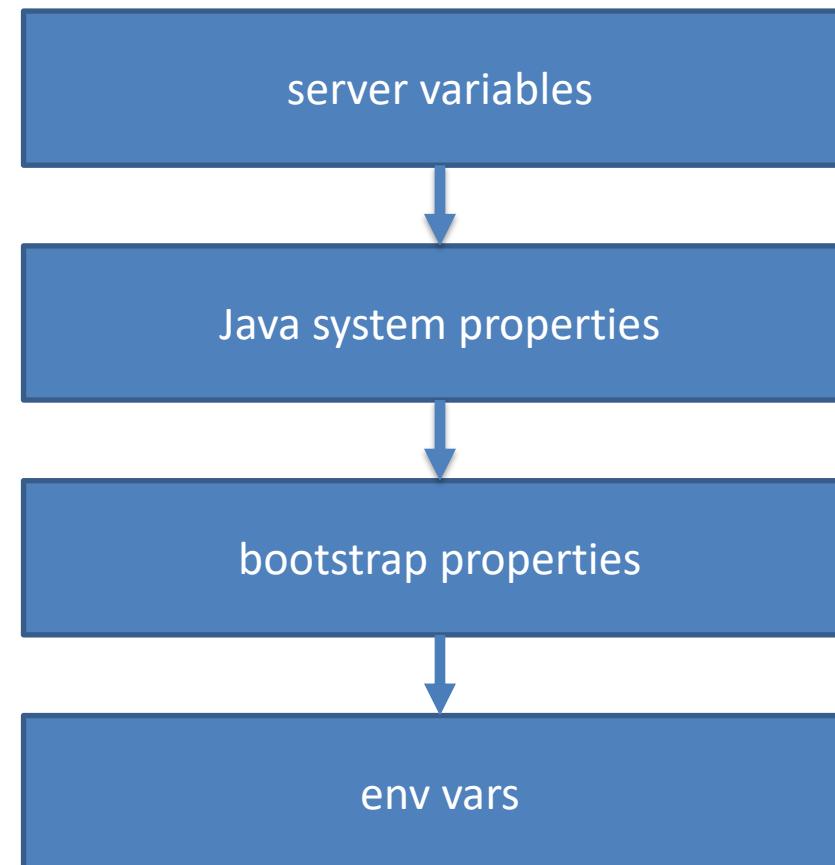
Module 5



# Variable Resolution



- Liberty variable resolution happens top to bottom



# Externalizing Config

```
<server>
  <httpEndpoint id="defaultHttpEndpoint" host="*"
    httpPort="${env.SERVER_HTTP_PORT}"
    httpsPort="${env.SERVER_HTTPS_PORT}"/>
</server>
```

configDropins/defaults/common-http.xml

# MicroProfile

- Inject Configuration
- Sourced from
  - Properties file
  - Environment
  - Java system properties

```
@Path("/")
public class RestEE {

    @Inject
    @ConfigProperty("name")
    private String name;

    @GET
    @Counter
    @Traced
    public String hello() {
        return "Hello " + name;
    }
}
```

# Testing

Module 6



# Testing



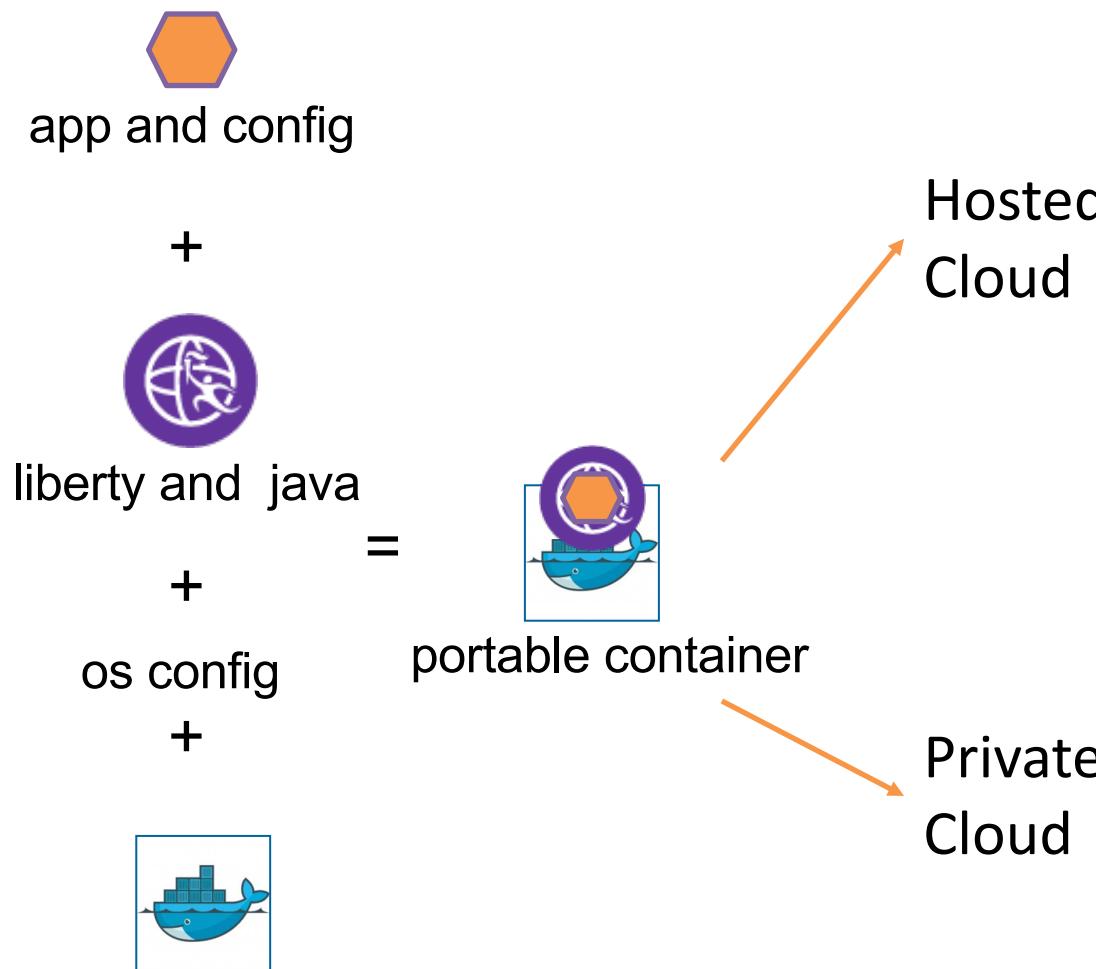
- Use maven-surefire-plugin to run unit test
  - Use CDI dependency injection to make your code easy to mock & unit test
- Use maven-failsafe-plugin to run integration test
- Use Liberty Arquillian integration to run tests on a Liberty server
- Use MicroShed Testing for system testing in Containers (Module 8)

# Docker

Module 7



# Liberty in Containers



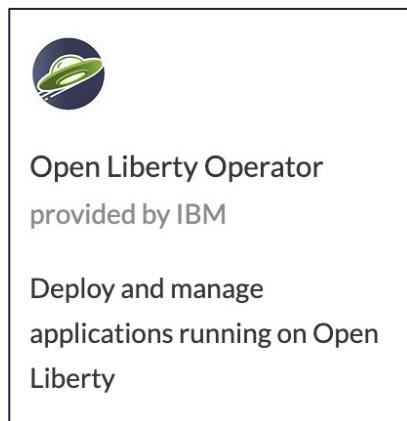
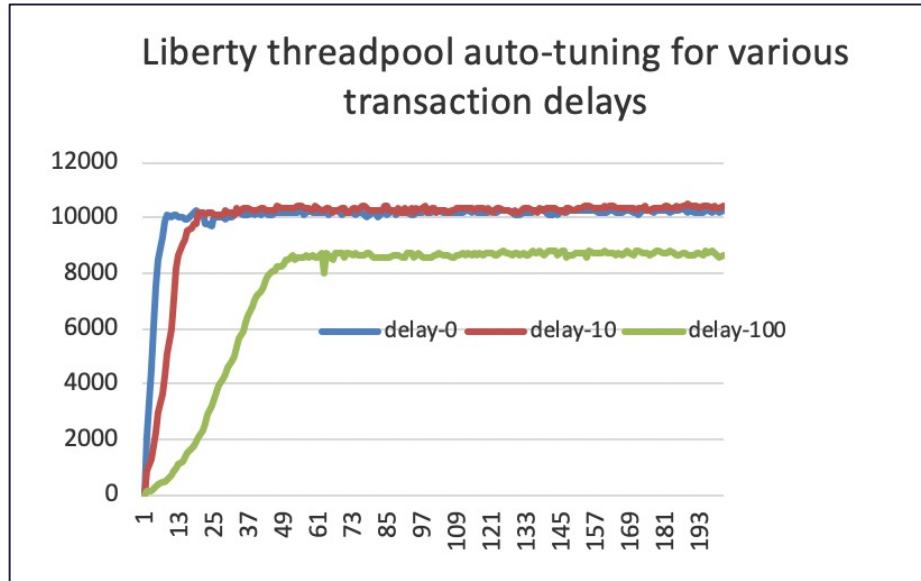
- IBM Cloud Kubernetes Service
- Azure Kubernetes Service
- Google Kubernetes Engine
- Amazon Elastic Kubernetes Service
- Jelastic
  
- Red Hat Open Shift Container Platform
- Pivotal Kubernetes Service
- Pivotal Cloud Foundry

```
FROM open-liberty  
ADD myapp.war /config/dropins/myapp.war
```

# Making the most of Docker



# Kubernetes optimized



- **Deliver faster** without costly tuning exercises
- Get **optimal performance** even as the environment changes
- **Simple Operator-based deploy** and day-2 operations experience
- Supported **production-ready images**
- **APIs** for Kubernetes integration
- Container-based **usage tracking**

# Testing in Containers

Module 8



# MicroShed Testing

- Integration tests that are easy to setup, write, and run
- Test your apps the same way they run in production...in Containers
- Can run multiple containers on same network (e.g. test DB integration)
- <https://microshed.github.io/>

```
@MicroShedTest
public class MyTest {

    // Search for Dockerfile.
    // Start app in Container.
    // Wait for Container before running tests.
    @Container
    public static MicroProfileApplication app
        = new MicroProfileApplication()
            .withAppContextRoot("/myservice");

    // Inject JAX-RS REST Client
    @Inject
    public static MyService mySvc;

    // A test method like any other
    @Test
    public void testMyService() {
        ...
    }
}
```

# Support Licensing

Module 9



# How to get Support

## WebSphere



z/OS  
ND  
Base  
Core



## WebSphere Hybrid Edition

### IBM Integrated Application Runtimes

Java:

- WebSphere
- Liberty
- MicroProfile
- Jakarta EE
- OpenJ9



**Cloud Foundry Migration Runtime**

**Transformation Advisor**

**Mono2Micro**

 **Red Hat OpenShift**

# Wrap-up



# Mastersclass hands-on content



- [Open Liberty Masterclass](#)
  - [Table of Contents](#)
  - [Before you begin](#)
    - [Install Pre-requisites](#)
    - [Prime Maven and Docker Caches](#)
  - [The Application](#)
  - [Module 1: Build](#)
  - [Module 2: Dev Mode](#)
  - [Module 3: Application APIs](#)
  - [Module 4: Server Configuration](#)
  - [Module 5: Externalizing Configuration](#)
  - [Module 6: Integration Testing](#)
  - [Module 7: Docker](#)
    - [Overriding Dev Server Configuration](#)
  - [Module 8: Support Licensing](#)
  - [Conclusion](#)

[Get Started](#)[Guides](#)[Docs](#)[Support](#)[Blog](#)

# Guides

The quickest way to learn all things Open Liberty, and beyond!

[Filter guides](#)

X

## DEVELOP (37 guides)

[Getting started](#)[RESTful service](#)[Reactive service](#)[Configuration](#)[Fault tolerance](#)[Observability](#)[Security](#)[Persistence](#)[Client side](#)

## BUILD AND TEST (10 guides)

[Build](#)[Test](#)[Containerize](#)

## DEPLOY (9 guides)

[Kubernetes](#)[Cloud deployment](#)

## Developing your cloud-native application

### Getting started

#### Getting started with Open Liberty

Learn how to develop a Java application on Open Liberty with Maven and Docker.

⌚ 25 minutes

#### Injecting dependencies into microservices

Learn how to use Contexts and Dependency Injection (CDI) to manage and inject dependencies into microservices.

⌚ 15 minutes

### RESTful service

#### Creating a RESTful web service

Learn how to create a REST service with JAX-RS, JSON-B, and Open Liberty.

#### Consuming RESTful services with template interfaces

Learn how to use MicroProfile Rest Client to invoke RESTful services over HTTP in a type-safe way.

#### Consuming a RESTful web service

Explore how to access a simple RESTful web service and consume its resources in Java using JSON-B and JSON-P.

#### Documenting RESTful APIs

Explore how to document and filter RESTful APIs from code or static files by using MicroProfile OpenAPI.

# Thank You

